



DESIGN OF NEURAL NETWORK FILTERS

JAN LARSEN

Department of Mathematical Modelling, Building 349
Technical University of Denmark
DK-2800 Lyngby, Denmark

© 1993, 1996 by Jan Larsen

SYNOPSIS

Emnet for nærværende licentiatafhandling er *design af neurale netværks filtre*. Filtre baseret på neurale netværk kan ses som udvidelser af det klassiske lineære adaptive filter rettet mod modellering af ulineære sammenhænge. Hovedvægten lægges på en neural netværks implementering af den ikke-rekursive, ulineære adaptive model med additiv støj. Formålet er at klarlægge en række faser forbundet med design af neural netværks arkitekturer med henblik på at udføre forskellige “black-box” modellerings opgaver så som: System identifikation, invers modellering og prædiktions af tidsserier.

De væsentligste bidrag omfatter:

- Formulering af en neural netværks baseret kanonisk filter repræsentation, der danner baggrund for udvikling af et arkitektur klassifikationssystem. I hovedsagen drejer det sig om en skelnen mellem globale og lokale modeller. Dette leder til at en række kendte neurale netværks arkitekturer kan klassificeres, og yderligere åbnes der mulighed for udvikling af helt nye strukturer. I denne sammenhæng findes en gennemgang af en række velkendte arkitekturer. I særdeleshed lægges der vægt på behandlingen af multi-lags perceptron neural netværket.
- Ved at kombinere det kanoniske filter med en præprocesseringsenhed fremkommer, hvad vi vil kalde, den grundlæggende ikke-lineære filter arkitektur. Arkitekturen må fortolkes som værende et heterogent tre-lags neuralt netværk. Med det formål at undgå overparametrisering foreslås forskellige præprocesserings metoder under hensyntagen til, at kvaliteten af den endelige model ikke forringes væsentligt.
- Diskussion af forskellige parameterestimationsalgoritmer, og forslag til effektive implementeringer af sædvanlige første og anden ordens optimerings algoritmer i forbindelse med lagdelte arkitekturer. Yderligere er der udviklet en algoritme til initialisering af et 2-lags neuralt netværk, således at hurtigere optimering sikres.
- Med udgangspunkt i en såkaldt modelfejlsdekomposition af middel-generalisationsfejlen klarlægges og diskuteres fundamentale begrænsninger i forbindelse med valg af optimal netværksarkitektur. Dette inkluderer bl.a. en behandling af de muligheder anvendelsen af regularisering bibringer.
- Udvikling og diskussion af en ny generalisationsfejlestimator, *GEN*, der finder anvendelse i forbindelse med ufuldstændige, ikke-lineære modeller. Muligheden for at kunne omfatte ufuldstændige modeller er især vigtig ved “black-box” modellering. Modellerne forudsættes at være estimeret ved minimering af kvadratfejlssummen og et regulariseringsled. Estimatorens baserer sig på statistiske metoder og må ses som en udvidelse af dels Akaikes klassiske *FPE*-estimator og dels af Moodys *GPE*-estimator.

- Udvikling af forskellige statistisk baserede algoritmer, der optimerer filterarkitekturen ved gradvis reduktion. Disse algoritmer generaliserer Optimal Brain Damage og Optimal Brain Surgeon procedurerne.

Potentialet af de foreslåede metoder søges demonstreret ved hjælp af beregningseksempler og numeriske simuleringer. Endelig indeholder afhandlingen en kort gennemgang af klassiske ikke-lineær filteranalysemetoder.

ABSTRACT

The subject of this Ph.D. Thesis is *design of neural network filters*. Neural network filters may be viewed as an extension of classical linear adaptive filters to deal with nonlinear modeling tasks. We focus on neural network architectures for implementation of the non-recursive, nonlinear adaptive model with additive error. The objective is to clarify a number of phases involved in the design of neural network filter architectures in connection with “black box” modeling tasks such as system identification, inverse modeling and time-series prediction.

The major contributions comprises:

- The development of an architecture taxonomy based on formulating a canonical filter representation. The substantial part of the taxonomy is the distinction between global and local models. The taxonomy leads to the classification of a number of existing neural network architectures and, in addition, suggests the potential development of novel structures. Various architectures are reviewed and interpreted. Especially we attach importance to interpretations of the multi-layer perceptron neural network.
- Formulation of a generic nonlinear filter architecture which consists of a combination of the canonical filter and a preprocessing unit. The architecture may be viewed as a heterogeneous three-layer neural network. A number of preprocessing methods are suggested with reference to bypassing the “curse of dimensionality” without reducing the performance significantly.
- Discussion of various algorithms for estimating characteristic model weights (parameters). We suggest efficient implementations of standard first and second order optimization algorithms for layered architectures. In addition, in order to speed-up convergence a weight initialization algorithm for the 2-layer perceptron neural networks is developed.
- Clarification and discussion of fundamental limitations in the search for optimal network architectures based upon a decomposition of the average generalization error, called the model error decomposition. This includes a discussion of employing regularization.
- The development and discussion of a novel generalization error estimator, *GEN*, which is valid for incomplete, nonlinear models. The ability to deal with incomplete models is particularly important when performing “black box” modeling. The models are assumed to be estimated by minimizing the least squares cost function with a regularization term. The estimator is based on a statistical framework and may be viewed as an extension of Akaike’s classical *FPE*-estimator and Moody’s *GPE*-estimator.

- Development of various statistical based pruning procedures which generalize the Optimal Brain Damage and the Optimal Brain Surgeon procedures.

The potential of the various proposals is substantiated by analytical results and numerical simulations. Furthermore, the Thesis comprises a brief review of classical nonlinear filter analysis.

CONTENTS

SYNOPSIS	i
ABSTRACT	iii
PREFACE	ix
SYMBOLS, REFERENCE INDICATIONS AND ABBREVIATIONS	xiii
1 INTRODUCTION	1
1.1 Historical Outline	2
1.2 Nonlinear Models	3
1.3 Fields of Applications	7
1.4 Designing Neural Network Filters	11
1.5 Summary	13
2 NONLINEAR FILTER ANALYSIS	15
2.1 Basic Properties of Nonlinear Filters	15
2.1.1 Superposition	16
2.1.2 Time-Invariance	16
2.1.3 Stability	18
2.1.4 Causality	18
2.2 Analysis of Nonlinear Filters	18
2.2.1 Nonlinear Filters based on Zero-Memory Nonlinear Filters	20
2.2.2 Volterra Filters and Frequency-Domain Interpretations	23
2.2.3 Concluding Remarks on Nonlinear Filter Analysis	26
2.3 Summary	27
3 NONLINEAR FILTER ARCHITECTURES	28
3.1 Taxonomy of Nonlinear Filter Architectures	28
3.1.1 Parametric and Nonparametric Architectures	29
3.1.2 Local and Global Approximation	30
3.1.3 Orthogonal Architectures	35
3.1.4 Classification of Filter Architectures	39
3.2 Global Approximation	39
3.2.1 Polynomial Filters	39
3.2.2 Multi-Layer Neural Networks	49
3.2.3 Gram-Schmidt Neural Networks	65
3.2.4 Canonical Piecewise-Linear Filters	68

3.2.5	Semilocal Units	71
3.2.6	Projection Pursuit	73
3.2.7	Neural Network with FIR/IIR Synapses	74
3.3	Local Approximation	75
3.3.1	Localized Receptive Field Networks	75
3.3.2	Tree-Structured Piecewise-Linear Filter	78
3.3.3	Gate Function Filters	80
3.4	Nonparametric Approximation	81
3.4.1	Local Filters	81
3.4.2	Parzen Window Regression	82
3.5	Discussion	83
3.6	Summary	84
4	A GENERIC NONLINEAR FILTER ARCHITECTURE BASED ON NEURAL NETWORKS	85
4.1	The Generic Neural Network Architecture	85
4.2	Preprocessing Methods	87
4.2.1	Preprocessing Algorithm	88
4.3	Principal Component Analysis	92
4.4	Derivative Preprocessor	95
4.5	Laguerre Function Preprocessor	98
4.6	Summary	99
5	ALGORITHMS FOR FILTER PARAMETER ESTIMATION	100
5.1	Parameter Estimation	100
5.1.1	Consistency	103
5.1.2	Least Squares Estimation	104
5.1.3	Maximum Likelihood Estimation	106
5.2	Performing Least Squares Estimation	107
5.3	Gradient Descent Algorithm	111
5.3.1	Convergence	113
5.3.2	Weight Initialization	114
5.3.3	The Back-Propagation Algorithm	120
5.3.4	Back-Propagation in the MFPNN	122
5.4	Stochastic Gradient Algorithm	128
5.4.1	Convergence and Step-Size Selection	129
5.4.2	The SG-algorithm and Computational Complexity	132
5.5	The Modified Gauss-Newton Algorithm	134
5.6	Recursive Gauss-Newton Algorithm	139
5.7	Recursive Gauss-Newton Algorithm with Bierman Factorization	146
5.8	Summary	150
6	FILTER ARCHITECTURE SYNTHESIS	151
6.1	System and Model	151
6.2	A Priori Knowledge	152
6.3	Generalization Ability	152
6.3.1	Alternative Definitions of Generalization Ability	153

6.3.2	Average Generalization Error	154
6.3.3	Decomposition of the Generalization Error	155
6.3.4	Model Error Decomposition	156
6.3.5	Bias/Variance Decomposition	160
6.3.6	Simple Invariance Property of the Generalization Error within MF-PNN	161
6.4	Fundamental Limitations	163
6.4.1	Complete Models	164
6.4.2	Incomplete Models	164
6.5	Generalization Error Estimates	166
6.5.1	Basic Architecture Synthesis Algorithm	166
6.5.2	The Mean Square Training Error	168
6.5.3	Cross-Validation	169
6.5.4	Leave-One-Out Cross-Validation	172
6.5.5	An Information Theoretic Criterion	174
6.5.6	The Final Prediction Error Estimator	175
6.5.7	The Generalized Prediction Error Estimator	177
6.5.8	The Generalization Error Estimator for Incomplete, Nonlinear Models	179
6.6	Statistical Methods for Improving Generalization	195
6.6.1	Linear Hypotheses	195
6.6.2	Hypothesis Testing	196
6.6.3	Asymptotic Distribution of the Weight Estimate	197
6.6.4	Irregular Hypotheses	202
6.6.5	Retraining	203
6.6.6	Similarities Between the Statistical Framework and Generalization Error Estimates	205
6.7	Procedures for Pruning the Architecture	206
6.7.1	Simple Pruning in Neural Networks	206
6.7.2	Statistically Based Pruning Procedures	207
6.7.3	Statistical Pruning Algorithms	214
6.7.4	Pruning Algorithms based on Generalization Error Estimates	215
6.7.5	Optimal Brain Damage	217
6.7.6	Optimal Brain Surgeon	218
6.8	Procedures for Expanding the Architecture	220
6.8.1	Stepwise Forward Inclusion	221
6.8.2	Cascade-Correlation	222
6.8.3	Statistical Expansion Test	223
6.8.4	Partition Function Filters	227
6.9	Reducing Generalization Error by Regularization	231
6.10	Summary	233
7	VALIDATION OF ARCHITECTURE SYNTHESIS METHODS	235
7.1	Validation of the GEN-estimate	235
7.1.1	Simulation Setup	236
7.1.2	Simulated Systems	242
7.1.3	Simulation Results	253
7.2	Validation of Statistically based Pruning Algorithms	276
7.2.1	Simulation Setup	277

7.2.2	Results	280
7.3	Summary	280
8	SIMULATION OF ARTIFICIAL SYSTEMS	282
8.1	Validating the Weight Initialization Algorithm	282
8.2	Comparison of Parameter Estimation Algorithms	283
8.3	Testing Neural Networks for Signal Processing Tasks	288
8.3.1	System Identification	288
8.3.2	Inverse Modeling	289
8.3.3	Time-Series Prediction	296
8.3.4	Modeling the Simulated Systems	299
8.4	Summary	310
9	CONCLUSION	315
A	GENERALIZATION ERROR ESTIMATES FOR XN-MODELS	321
A.1	The Basis of Estimating the Generalization Error	321
A.1.1	Systems and Models	321
A.1.2	Estimation of Model Parameters	323
A.1.3	Generalization Ability	326
A.2	Derivation of Generalization Error Estimates	327
A.2.1	LS Cost Function	328
A.2.2	LS Cost Function with Regularization Term	346
A.3	Summary	354
B	APPROXIMATION OF INVERSE STOCHASTIC MATRICES	355
B.1	Approximation of \mathbf{H}_N^{-1}	355
B.2	Approximation of $E\mathbf{H}_N^{-1}$	358
B.2.1	On the Large N Assumption	360
B.2.2	LX-models	365
B.3	Summary	367
C	EXPECTATION OF PRODUCT-SUMS OF STOCHASTIC MATRICES	368
D	EVALUATION OF GAUSSIAN INTEGRALS	370
D.1	One and Two-dimensional Gaussian Integrals	370
D.2	Generalization Error in a Simple Neural Model	371
D.2.1	The term G_1	373
D.2.2	The term G_2	374
D.2.3	The term G_3	374
D.3	Summary	375
E	MOMENTS OF GAUSSIAN STOCHASTIC VECTORS	376
E.1	The Hessian of a Polynomial Filter	376
E.2	Moment Calculations	377

F	STUDIES OF THE WEIGHT FLUCTUATION PENALTY	380
F.1	On the Changes in WFP Due to Model Complexity	380
F.2	The WFP when Dealing with Insignificant Weights	385
F.2.1	WFP of the Unrestricted Model	386
F.2.2	WFP of the Restricted Model	387
G	REDUCING GENERALIZATION ERROR BY REGULARIZATION	389
G.1	System and Model	389
G.2	Mean Square Model Error	390
G.3	Weight Fluctuation Penalty	391
G.4	Optimizing the Regularization Parameter	395
H	PAPER 1: A NEURAL ARCHITECTURE FOR ADAPTIVE FILTER-	
	ING	398
H.1	Introduction	399
H.2	Nonlinear Filter Architecture	400
H.3	Filter Design	400
H.3.1	Signal Dependence	400
H.3.2	Preprocessing Methods	401
H.3.3	Memoryless Multidimensional Nonlinearities	402
H.3.4	Weight Estimation Algorithms	403
H.4	Simulations	404
H.4.1	Simulated Systems	404
H.4.2	Numerical Results	405
H.5	Conclusion	406
H.6	Acknowledgments	406
I	PAPER 2: A GENERALIZATION ERROR ESTIMATE	408
I.1	Introduction	409
I.2	Estimate for Incomplete, Nonlinear Models	410
I.3	Numerical Experiments	412
I.3.1	Linear System	413
I.3.2	Simple Neural Network	415
I.4	Conclusion	416
I.5	Acknowledgments	417
	BIBLIOGRAPHY	418

PREFACE

The work presented in this Ph.D. Thesis was performed at the Electronics Institute Building 349, The Technical University of Denmark, DK-2800 Lyngby with Assoc. Prof. John E. Aasted Sørensen, Ph.D., Manager, Assoc. Prof. Peter Koefoed Møller, M.Sc., and Assoc. Prof. Lars Kai Hansen, Ph.D. as supervisors.

The work was granted by the Technical University of Denmark and partly by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT). I would like to thank The Technical University of Denmark, CONNECT and The Otto Mønsted Foundation for financial support in connection with conference participations. Furthermore, Peter Koefoed Møller is thanked for providing computer facilities and financial support towards the payment of travel expenses.

Lars Kai Hansen, Nils Hoffmann, Anders Krogh, Peter Koefoed Møller, Klaus Bolding Rasmussen, Lars Risbo, Claus Svarer, John E. Aasted Sørensen are gratefully acknowledged for their contribution to many profitable discussions and for providing comments and suggestions which improved the quality of the Thesis. In particular, I would like to thank Nils Hoffmann for a frictionless and valuable partnership of long standing.

The Thesis is carried out on the assumption that the reader is acquainted with the fundamentals of signal analysis, adaptive signal processing and statistics.

The starting point of the work is linear adaptive filters which have been treated intensively during the last three decades. The goal has been to extend these filters to the nonlinear domain in order to handle more complex signal processing tasks. In particular, the rapidly evolving field of neural networks has been appealing with reference to the design of nonlinear adaptive filters. The work has concentrated on explaining various aspects involved in the design of the non-recursive nonlinear filter which forms the counterpart to the linear adaptive FIR-filter.

The Thesis is composed of a number of chapters and appendices. The appendices contain mathematical material which is not necessary for an immediate comprehension. It has been a common objective that – to some extent – one should be able to read the chapters and appendices independently.

First of all list of symbols, reference indications and abbreviation is provided. However, the used symbols and abbreviations are widely repeated in the text.

Ch. 1 introduces the Thesis by providing a link to classical adaptive signal processing. The extension of linear adaptive filtering to the nonlinear case is presented and the limitations of the studied filters are given. The principal topic of this Thesis is the concept of neural network filter architectures which is introduced subsequently. The notion architecture refers to the formal construction of the nonlinear filter. Finally the introduction contains a simplified scheme for the design of neural network filters. At the same time this scheme forms an outline of various matters to be considered in the succeeding chapters.

In Ch. 2 classical theory for nonlinear filter analysis is briefly reviewed. This includes

basic properties of nonlinear filters along with methods for analyzing the functionality or mapping capabilities. In particular, the mapping capabilities of two classical nonlinear filters are discussed. This comprises the nonlinear filter based on cascading a linear filter with a zero-memory nonlinearity and the Volterra filter.

Ch. 3 presents a novel taxonomy of nonlinear filter architectures based on general properties of the filter architecture. The purpose is to convey a unified view of numerous filter architectures suggested in the literature and to guide the choice of novel filter architectures. The unification is explicitly done by formulating a canonical filter representation. Based on the taxonomy a number of existing neural network filter architectures are expounded and compared. In particular, the multi-layer feed-forward perceptron neural network (MFPNN) is attached importance.

A generic nonlinear filter architecture based on neural networks is presented in Ch. 4. The architecture consists of a preprocessor succeeded by a nonlinear filter which is formulated by the canonical filter representation. Several preprocessing methods is also described. Part of this work was carried out in co-operation with Nils Hoffmann.

The topic of Ch. 5 is estimation of the weights which specify the nonlinear filter. The weights are estimated by minimizing some cost function. Various first and second order local optimization schemes are suggested, and aiming at faster convergence we proposed an algorithm for weight initialization. Furthermore, an efficient scheme for calculation of algorithm quantities (i.e., the gradient and the Hessian) in layered architectures – like the MFPNN – is developed.

Ch. 6 deals with the synthesis of proper filter architectures. First the concept of generalization ability is discussed and suggested as a quality measure for filter architectures. The generalization ability is in this Thesis defined by the average generalization error which is given various interpretations in terms of the suggested model error decomposition. This leads to a clarification of the fundamental limitations in searching for the optimal filter architecture. Next, a basic architecture synthesis algorithm based upon an estimate of the generalization error is provided. A number of commonly generalization error estimates are reviewed and a novel estimator, the generalization error estimator for incomplete, nonlinear models (*GEN*) is presented. Furthermore, statistical procedures for filter architecture synthesis are discussed. The procedures constitute an extension of classical methods used within linear regression. Procedures for expanding and pruning is discussed and novel pruning schemes are developed (partly in co-operation with Nils Hoffmann). Finally, it is demonstrated that employing regularization may be viewed as a tool for improving the filter architecture even though the architecture is not modified after all.

Ch. 7 provides simulations which substantiate various properties of the *GEN*-estimator. In addition, various statistical based pruning procedures are validated.

Ch. 8 covers various simulations supporting the theoretical considerations in the previous chapters. This comprises:

- Evaluation of the weight initialization algorithm.
- Comparison of first and second order weight estimation algorithms.
- Test of the MFPNN for various artificial signal processing tasks.

Finally, Ch. 9 states the conclusions of the presented work.

Several appendices are also provided. App. H and App. I contain preprints of papers published by the author and deal with topics discussed in the Thesis. The remaining

appendices contain material which links to specific chapters. The coherence is given in the following table:

Chapter	Appendices
Ch. 6	App. A, App. B, App. C, App. F, App. G
Ch. 7	App. D, App. E

Jan Larsen

Lyngby, Denmark

March 1993

January 1996

Second edition Jan. 1996 contains few updates and misprints are rectified.

The manuscript was typeset with L^AT_EX 2.09 and output from a HP *LaserJet 4M* printer.

SYMBOLS, REFERENCE INDICATIONS AND ABBREVIATIONS

Ordinary Symbols

In general, lower case boldfaced letters denote column vectors,

$$\mathbf{x} = [x_1, x_2, \dots, x_m]^\top$$

while capitalized boldfaced letters denote matrices, e.g., \mathbf{X} . Furthermore, calligraphical letters, \mathcal{X} , denote – as a principal rule – sets.

\mathbf{a}	Normal vector of a separating hyperplane, \mathcal{H} . Also used to denote an arbitrary nonzero vector.
b	Transition width of the activation function.
$b(k)$	Impulse response of a FIR-filter.
$b(\mathbf{z}(k); \boldsymbol{\beta})$	Basis function parameterized by $\boldsymbol{\beta}$.
c	A constant.
$C_{m,n}$	The binomial coefficient $m!/(n!(m-n)!)$.
$C(\mathbf{w})$	The expected cost function evaluated at the weights \mathbf{w} .
$C_N(\mathbf{w})$	The cost function based on a training set of size N evaluated at the weights \mathbf{w} .
$C_\nu(\mathcal{T})$	The cross-validation estimate based on the training set, \mathcal{T} , when using ν per cent of the samples for estimating the weights.
\mathbf{C}	The $p \times L$ dimensional preprocessing matrix.
\mathcal{C}	Reject region of a statistical hypothesis.
CP	Computational complexity measured as the total number of multiplications and divisions.
$\mathbf{d}_n^{(s)}$	The s 'th weight deletion vector with dimension n .
D	Time delay. A negative value corresponds to the future.
$D(\mathbf{z}(k))$	Domain function which specify the active domain of the basis function $b(\cdot)$.
\mathbf{D}	A diagonal matrix used in matrix decompositions.
\mathcal{D}	The input domain, i.e., the possible range of $\mathbf{z}(k)$.
\mathcal{D}_j	Subset of \mathcal{D} .
dr	Dynamic range.

$e(k)$	The error signal, i.e., the difference between the desired signal and the output of the model.
E_c	Relative cross-validation error index.
$E(\mathbf{w}; \boldsymbol{\lambda})$	Extended cost function (Lagrange function) used in the Lagrange multiplier technique.
f	Normalized frequency, i.e., the frequency normalized w.r.t. the sampling frequency. When explicitly emphasizing that the frequency is normalized the symbol, f_n , is used.
f_s	Sampling frequency, i.e. $f_s = 1/T$ where T is the sampling period.
$f(\mathbf{x}(k); \mathbf{w})$	The filtering function parameterized by the weights \mathbf{w} .
$f_n(\mathbf{z}(k); \mathbf{w})$	The nonlinear filtering function parameterized by the weights \mathbf{w} .
$\mathbf{f}_p(\mathbf{x}(k))$	The preprocessing (vector) function.
\mathcal{F}	A set of parameterized nonlinear functionals which also is denoted: The filter architecture.
$g(\mathbf{x}(k))$	Function which constitutes the filtering of a nonlinear system.
$g_n(\mathbf{z}(k))$	Nonlinear filtering involved in a nonlinear system.
$G(\mathbf{w})$	Generalization error (expected LS cost function).
$h(u)$	Activation function of a nonlinear neuron.
$h(n)$	Impulse response of a linear filter.
$h(n_1, \dots, n_r)$	Time-domain kernel of a r-linear (Volterra) filter.
$H(f)$	Frequency response of a linear filter.
$H(f_1, \dots, f_r)$	The r -linear frequency response function.
\mathbf{H}	The Hessian matrix of the expected cost function $G(\mathbf{w})$.
\mathbf{H}_N	The Hessian matrix of the cost function $S_N(\mathbf{w})$.
$\tilde{\mathbf{H}}_N$	The pseudo Hessian matrix of the cost function $S_N(\mathbf{w})$.
\mathcal{H}	A hyperplane or a statistical hypothesis.
I	An integral.
\mathbf{I}	The identity matrix.
i	An integer used to index variables. Especially used to index a specific basis function, $b_i(\cdot)$.
itr	The number of iterations, i.e., the number of times the training set is replicated during training.
j	An integer used to index variables. Especially used to index the individual components of $\mathbf{z}(k)$. Further it denotes the imaginary unit.
\mathbf{J}	The Hessian matrix of the expected cost function $C(\mathbf{w})$.
\mathbf{J}_N	The Hessian matrix of the cost function $C_N(\mathbf{w})$.
$\tilde{\mathbf{J}}_N$	The pseudo Hessian matrix of the cost function $C_N(\mathbf{w})$.
k	The discrete time index.
l	Integer which denotes the number of layers in a neural network. Further it denotes polynomial order.
L	The input memory length or window length, i.e., the length of the input signal vector, $\mathbf{x}(k)$.
$L(\mathcal{T})$	The leave-one-out cross-validation estimate based on the training set, \mathcal{T} .

m	The number of parameters in the model.
m_l	The number of neurons in the l 'th layer of a layered neural network.
\mathbf{m}	\mathbf{m} specifies the number of neurons in the an MFNN, i.e., $\mathbf{m} = [m_0, m_1, \dots, m_l]$. In general we refer to a \mathbf{m} -network.
M	The dependence lag of the input vector, i.e., $\mathbf{x}(k)$ and $\mathbf{x}(k + \tau)$ are dependent as $ \tau \leq M$.
\mathcal{M}	Misadjustment.
n	Integer used for indices and to indicate the number of restrictions.
N	Size of the training set or number of training examples.
N_c	Size of the cross-validation set.
\mathbb{N}	The set of natural numbers.
\mathcal{N}_α	The α -fractile of the standard Gaussian distribution, $\mathcal{N}(0, 1)$.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	The Gaussian (normal) distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.
$\mathbf{o}(\ \boldsymbol{\xi}(\mathbf{x})\)$	The vector order function. Let $\boldsymbol{\phi} = \mathbf{o}(\ \boldsymbol{\xi}(\mathbf{x})\)$ as $\mathbf{x} \rightarrow \mathbf{x}_o$ then $\frac{\boldsymbol{\phi}}{\ \boldsymbol{\xi}(\mathbf{x})\ } \rightarrow \mathbf{0}, \quad \mathbf{x} \rightarrow \mathbf{x}_o$
p	The dimension of the preprocessed input vector signal $\mathbf{z}(k)$.
$p_{\mathbf{x}}(\mathbf{x})$	Probability density function of the vector \mathbf{x} . The subscript is often omitted.
$p_{\mathbf{x} \mathbf{y}}(\mathbf{x} \mathbf{y})$	Conditional probability density function of \mathbf{x} conditioned on \mathbf{y} .
$P_x(f)$	Power spectrum of (the discrete signal) $x(k)$.
$P_r(x)$	Polynomial of order r in the variable x .
\mathbf{P}	The inverse (pseudo) Hessian matrix.
$\overline{\mathbf{P}}$	The non-normalized inverse (pseudo) Hessian matrix.
q	The number of basis functions and also used to denote an order of expansion.
\mathbf{q}_i	The i 'th eigenvector.
Q	The number of samples of a stochastic variable. The samples constitute an ensemble. Also used elsewhere, e.g., to denote the number of WDV's.
\mathbf{Q}	The matrix of eigenvectors, i.e., $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m]$. Also used in connection with different matrix decompositions.
\mathcal{Q}	A set of weight deletion vectors.
r	Integer used for indexing variables. In particular the order of a polynomial and the layer in a neural network.
$r(k; \mathbf{w})$	The regularizing term at time instant k .
$R_N(\mathbf{w})$	The regularizing term in the cost function, $C_N(\mathbf{w})$.
\mathbf{R}	Estimated correlation matrix.
\mathbb{R}_+	The set of positive real numbers.
\mathbb{R}^m	The set of real m -dimensional vectors.
$\mathbb{R}^{m \times p}$	The set of real matrices with m rows and p columns.
s	An integer used to index variables. In particular, the order of a polynomial.

$s(\cdot)$	Nonlinear function.
$\tilde{\mathbf{s}}^{(r)}(k)$	A vector signal containing the output signals of the neurons in the r 'th layer within an multi-layered neural network. Also used in a general layered filter architecture.
$\mathbf{s}^{(r)}(k)$	The augmented vector signal:

$$\mathbf{s}^{(r)}(k) = \left[1, \left(\tilde{\mathbf{s}}_i^{(r)}(k) \right)^\top \right]^\top$$

$S_N(\mathbf{w})$	The LS cost function based on a training set of size N evaluated at the weights \mathbf{w} .
\mathcal{S}	The stack consisting of stack elements \mathcal{E} .
$\text{sgn}(x)$	The signum function,

$$\text{sgn}(x) = \begin{cases} 1 & , x > 0 \\ 0 & , x = 0 \\ -1 & , x < 0 \end{cases}$$

t	Threshold which defines the offset of a hyperplane, \mathcal{H} . Furthermore, used to denote the continuous time.
T	Sampling period and Test statistic.
\mathcal{T}	Denotes the training set, i.e., samples of input and desired signals.
$u_i^{(r)}(k)$	The linear output of the i 'th neuron in the r 'th layer of an multi-layered neural network.
\mathbf{U}	An upper triangular matrix, i.e.,

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1m} \\ 0 & u_{22} & u_{23} & \cdots & u_{2m} \\ 0 & 0 & u_{33} & \cdots & u_{3m} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{mm} \end{bmatrix}$$

$v_i(k)$	The signal corresponding to the response of the i 'th basis function.
\mathbf{v}	The vector: $\mathbf{v} = [1, v_1(k), v_2(k), \dots, v_q(k)]^\top$.
\mathbf{V}	The weight covariance matrix equal to $E_{\mathcal{T}}\{\Delta\mathbf{w}\Delta\mathbf{w}^\top\}$.
\mathbf{w}	Weight or parameter vector with dimension m .
$w_{i,j}^{(r)}$	The j 'th component of $\mathbf{w}_i^{(r)}$.
$\mathbf{w}_i^{(r)}$	The weights associated with the i 'th neuron within the r 'th layer of an multi-layered neural network.
$\hat{\mathbf{w}}$	Denotes the estimated weights.
$\hat{\mathbf{w}}_N$	Denotes the estimated weights based on N training data.
\mathbf{w}^*	Denotes the optimal weights.
\mathbf{w}°	Denotes the true weights.
$\mathbf{W}^{(r)}$	The $m_{r+1} \times (m_r + 1)$ weight matrix of the r 'th layer in a multi-layer neural network.
\mathcal{W}	The set of parameter vectors $\hat{\mathbf{w}}$ which minimize the cost function.
\mathcal{W}^*	The set of optimal parameter vectors \mathbf{w}^* .

$x(k)$	The model input signal.
$x_s(k)$	The system input signal.
$\mathbf{x}(k)$	The (model) input signal vector: $[x(k), x(k-1), \dots, x(k-L+1)]^\top$.
\mathbf{X}_i	The i 'th matrix in the sequence $\mathbf{X}_1, \mathbf{X}_2, \dots$.
\mathbf{X}^q	The q 'th power of \mathbf{X} where q is an integer, thus

$$\mathbf{X}^q = \underbrace{\mathbf{X} \dots \mathbf{X}}_{q \text{ terms}}$$

and

$$\mathbf{X}^{-q} = \underbrace{\mathbf{X}^{-1} \dots \mathbf{X}^{-1}}_{q \text{ terms}}$$

$y(k)$	Output signal of the (nonlinear) model.
$y_s(k)$	Output signal of the system.
$\hat{y}(k)$	Predicted output of the (nonlinear) model, also denoted the output of the corresponding nonlinear filter.
z	The complex variable used in the z -transform.
$\mathbf{z}(k)$	The preprocessed input vector signal, i.e., $\mathbf{z}(k) = \mathbf{f}_p(\mathbf{x}(k))$.
$\mathbf{0}$	The zero vector or matrix.

Greek Symbols

α	An auxiliary parameter. Also used to denote a significance level.
α_j	The weight with which the j 'th basis function contribute to the formation of the filter output.
β	An auxiliary parameter.
$\beta(n, k)$	Weighting function.
$\boldsymbol{\beta}$	Parameters of a basis function.

γ	The initialization parameter of the Hessian in connection with recursive Gauss-Newton algorithms. Also used to denote an auxiliary parameter.
$\gamma_{xy}(\tau)$	The crosscovariance function of $x(k)$ and $y(k)$ at correlation lag τ , i.e.,

$$\gamma_{xy}(\tau) = E \{ (x(k) - E\{x(k)\}) (y(k + \tau) - E\{y(k)\}) \}$$

If $x(k) \equiv y(k)$ the crosscovariance function reduces to the autocovariance function $\gamma_x(\tau)$.

Γ	The average generalization error w.r.t. all training set of size N .
$\hat{\Gamma}$	An estimator of Γ . $\hat{\Gamma}_G$ is the estimator of Γ based on calculation of the generalization error, G .
δ	Used to denote an auxiliary parameter.
$\delta(k)$	The Kronecker delta function (k is an integer): $\delta(k) = 1$ when $k = 0$ and zero otherwise.
$\delta(f)$	The Dirac delta function (f is a real variable), i.e.,

$$\int \phi(f)\delta(f)df = 1$$

where $\phi(f)$ is an arbitrary function.

$\delta^{(r)}$	A vector used in order to calculate the gradient vector in a layered filter architecture.
$\delta\mathbf{w}$	The fluctuation of the weights around a fixed weight vector, e.g. the estimated weight vector. That means, $\delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}$.
$\Delta\mathbf{w}$	The fluctuation of the estimated weights around the optimal weights, i.e., $\Delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$.
$\Delta\mathbf{w}_{(i)}$	The weight update vector, i.e. $\Delta\mathbf{w}_{(i)} = \mathbf{w}_{(i+1)} - \mathbf{w}_{(i)}$. For recursive algorithms we use the notation: $\Delta\mathbf{w}(k)$.
$\epsilon(N)$	The epsilon vector function, i.e., $\epsilon(N) \rightarrow \mathbf{0}$, $N \rightarrow \infty$.
$\varepsilon(k)$	Inherent noise in a nonlinear system.
η	A scale parameter.
$\boldsymbol{\theta}(k)$	The perturbation matrix at time k or a perturbation vector.
Θ	Perturbation matrix.
κ	Regularization parameter, e.g., the weight decay parameter.
λ	Eigenvalue or exponential forgetting factor.
$\lambda(n)$	Instantaneous forgetting factor.
$\mathbf{\Lambda}$	Eigenvalue matrix, i.e., $\mathbf{\Lambda} = \text{diag}\{[\lambda_1, \lambda_1, \dots, \lambda_m]\}$.
μ	Step-size of a stochastic parameter estimation algorithm.
$\mu(\cdot)$	Denotes a measure.
$\mu(n)$	The step function and the time-varying step-size.
ν	Position parameter. Furthermore, it denotes the percentage of the data in the training set used for estimating the weights when using cross-validation.
$\xi(k)$	A random i.i.d. sequence.
$\xi(\cdot)$	Restriction function.

Ξ	Denotes a hypersphere.
Ξ	Restriction matrix.
π	The irrational number 3.141592...
Π	Denotes the probability of proximity.
ρ_{xy}	The correlation coefficient

$$\rho_{xy} = \frac{E\{(x - \mu_x)(y - \mu_y)\}}{\sqrt{V\{x\}V\{y\}}}$$

where μ_x, μ_y are the mean values of x and y , respectively. The subscript xy is possibly omitted.

ϱ	The saliency. $\tilde{\varrho}$ is the modified saliency.
σ_x^2	The variance of x .
ς	The size of the stack \mathcal{S} .
Σ	Covariance matrix.
τ	Correlation lag. Also used to denote a threshold.
$\phi_{xy}(\tau)$	The crosscorrelation function of $x(k)$ and $y(k)$ at correlation lag τ , i.e.,

$$\phi_{xy}(\tau) = E\{x(k)y(k + \tau)\}$$

If $x(k) \equiv y(k)$ the crosscorrelation function reduces to the autocorrelation function $\phi_x(\tau)$. Furthermore if the τ dependence is omitted we tacitly take τ to be equal to zero.

$\varphi(\cdot)$	An arbitrary auxiliary function. Possibly a vector function indicated by boldface.
Φ	Correlation matrix.
χ	Eigenvalue spread, i.e., $\lambda_{\max}/\lambda_{\min}$.
$\chi^2(n)$	The chi-square distribution with n degrees of freedom.
$\chi_{1-\alpha}^2(n)$	The $1 - \alpha$ fractile of the chi-square distribution with n degrees of freedom.
$\psi(k; \mathbf{w})$	The partial derivative vector of the output of a nonlinear system w.r.t. the weights:

$$\frac{\partial f(\mathbf{x}(k); \mathbf{w})}{\partial \mathbf{w}}$$

$\Psi(k; \mathbf{w})$	The partial derivative matrix:
-----------------------	--------------------------------

$$\frac{\partial \psi(k; \mathbf{w})}{\partial \mathbf{w}^\top}$$

ω	The normalized angular frequency.
$\boldsymbol{\omega}$	The linearly transformed weight vector, $\boldsymbol{\omega} = \mathbf{Q}^\top \mathbf{w}$, where \mathbf{Q} is a transformation matrix.
Ω, Ω^*	Bounded sets in \mathbb{R}^m .

Operators

\approx	Denotes any type of approximation.
\lesssim	Approximately less than.

$\arg \min_x y(x)$	Denotes the argument(s) x which minimizes the function $y(x)$.
\leftarrow	The assign operator. For instance, $x \leftarrow y$ means that x is assigned the value of y .
$\langle x \rangle$	The (time) average of x .
$B\{\hat{x}\}$	Bias of an estimator \hat{x} of x , defined by: $B\{\hat{x}\} = E\{\hat{x}\} - x$, i.e the expected value minus the true value.
\square	Denotes the end of an example.
$*$	The convolution operator.
$D^i[x(k)]$	The discrete i 'th derivative operator, i.e., $D^i[x(k)] \approx x'(t)$.
$\det(\mathbf{X})$	The determinant of the square matrix \mathbf{X} .
$\text{diag}\{\mathbf{x}\}$	The diagonal matrix:

$$\begin{bmatrix} x_{11} & 0 & \cdots & 0 \\ 0 & x_{22} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & x_{m,m} \end{bmatrix}$$

$\dim(\mathbf{X})$	Dimension of the matrix (vector) \mathbf{X} .
\emptyset	The empty set.
\leftrightarrow	Denotes two objects which are equivalent in some respect.
\hat{x}	Denotes an estimator of x .
$E\{x\}$	The expectation of x .
$E\{x y\}$	Conditional expectation of x w.r.t. y .
$\text{Fou}\{\cdot\}$	The Fourier transform operator.
$\nabla_N(\mathbf{w})$	The gradient of the cost function $C_N(\mathbf{w})$.
\Leftrightarrow	Identical equal to.
\mathbf{X}^{-1}	The inverse of \mathbf{X} .
\Rightarrow	Implication operator.
$\langle x, y \rangle$	The inner product of two arbitrary objects, e.g., functions or vectors.
$[a; b]$	The real closed interval from a to b . If one of the brackets (or both) is reversed - e.g., if the left bracket $[$ is replaced by $]$ - then the interval is open.
\otimes	The Kronecker tensor product.
$\mathcal{A} \mapsto \mathcal{B}$	Denotes a mapping of the space \mathcal{A} into the space \mathcal{B} .
$[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m]$	A matrix composed of the m column vectors, \mathbf{x}_i .
$x \times y$	Used to indicate a matrix with x rows and y columns.
$\ \cdot\ $	A vector or matrix norm. A present subindex defines a specific norm, e.g., $\ \cdot\ _2$; the 2-norm. If taken on a vector we usually consider the 2-norm.
$\#$	The number symbol.
$ x $	Numerical value of x . If x is a complex number the numerical value corresponds to the modulus.
$ \mathbf{x} $	The Euclidean length of the vector, i.e., $ \mathbf{x} = \mathbf{x}^\top \mathbf{x}$.
$\partial x / \partial \mathbf{y}$	Partial derivative of the scalar x w.r.t. the (column) vector \mathbf{y} . That

is,

$$\frac{\partial \mathbf{x}}{\partial \mathbf{y}} = \left[\frac{\partial x}{\partial y_1}, \dots, \frac{\partial x}{\partial y_m} \right]^\top$$

$\partial \mathbf{x} / \partial \mathbf{y}^\top$

Partial derivative matrix of the (column) vector \mathbf{x} ($\dim(\mathbf{x}) = p$) w.r.t. the (row) vector \mathbf{y}^\top ($\dim(\mathbf{y}) = m$). That is:

$$\frac{\partial \mathbf{x}}{\partial \mathbf{y}^\top} = \begin{bmatrix} \frac{\partial x_1}{\partial y_1} & \dots & \frac{\partial x_1}{\partial y_m} \\ \vdots & & \vdots \\ \frac{\partial x_p}{\partial y_1} & \dots & \frac{\partial x_p}{\partial y_m} \end{bmatrix}$$

$\text{Prob}\{A\}$	Probability of the event A .
\odot	The element by element product of matrices (vectors), i.e., $\mathbf{A} \odot \mathbf{B} = \{a_{ij}b_{ij}\}$, where A and B are two matrices with equal dimensions.
\triangleq	Per definition.
\propto	The proportional operator.
$\lceil \cdot \rceil$	Denotes rounding to the nearest integer towards infinity.
$\lfloor \cdot \rfloor$	Denotes rounding to the nearest integer towards minus infinity.
$\mathbf{X}^{1/2}$	The square root matrix, i.e., $\mathbf{X} = \mathbf{X}^{1/2} \mathbf{X}^{\top/2}$
\circ	Denotes a tensor product.
$\text{tr}[\cdot]$	The trace operator, i.e., let $\mathbf{X} = \{x_{ij}\}$ be a $m \times m$ matrix then $\text{tr}[\mathbf{X}] = \sum_{i=1}^m x_{ii}$.
\mathbf{x}^\top	The transposed of \mathbf{x} .
$\mathbf{X}^{-\top}$	The transposed inverse matrix, i.e., $\mathbf{X}^{-\top} = (\mathbf{X}^{-1})^\top$.
$V\{x\}$	The variance of x .
$[1, 2, \dots, n]$	Denotes a row vector with n elements.
x'	The derivative of x .
$\{x_{ij}\}$	The matrix \mathbf{X} consisting of the elements x_{ij} where i and j are the row and column indices, respectively.

Reference Indications

App. x	Reference to Appendix x .
As. $x.y$	Reference to Assumption y in Chapter (Appendix) x .
$[x, y]$	Bibliography reference: x is the author name(s) and publication year, see the Bibliography chapter for further details. y (which may be omitted) represents some detailed information concerning the reference, e.g., chapter or page numbers.
Ch. x	Reference to Chapter x .
Co. $x.y$	Reference to Corollary y in Chapter (Appendix) x .
Def. $x.y$	Reference to Definition y in Chapter (Appendix) x .
Eq. $(x.y)$	Reference to Equation y in Chapter (Appendix) x .
Ex. $x.y$	Reference to Example y in Chapter (Appendix) x .
Fig. $x.y$	Reference to Figure y in Chapter (Appendix) x .

x	Indicates the reference to footnote no. x .
p. x , pp. x - y	Reference to page x or the pages x - y .
Sec. $x.y_1.y_2.y_3$	Reference to Section $y_1.y_2.y_3$ in Chapter (Appendix) x .
Table $x.y$	Reference to Table y in Chapter (Appendix) x .
Th. $x.y$	Reference to Theorem y in Chapter (Appendix) x .

Abbreviations

ASA	A rchitecture S ynthesis A lgorithm.
AIC	A n I nformation T heoretic C riterion.
BIBO	B ounded- I nput- B ounded- O utput.
BP	B ack- P ropagation.
cf.	Confer (according to).
DPP	D erivative P reprocessor.
e.g.	Exempli gratia (for instance).
FIR	Finite impulse response.
FPE	F inal P rediction E rror.
GD	G radient D escent.
GEN	G eneralization E rror Estimator for Incomplete, Nonlinear Models.
GEPA	G eneralization E rror based P runing A lgorithm.
GFF	G ate F unction F ilter.
GNFA	G eneric N onlinear F ilter A rchitecture.
GPE	G eneralized P rediction E rror.
i.e.	Id est (that is).
i.i.d.	Independent identically distributed stochastic – possibly multi-dimensional – sequence. That is, let $\mathbf{x}(k)$, $k = 1, 2, \dots$ be the sequence then $\mathbf{x}(k_1)$ is independent of $\mathbf{x}(k_2)$, $\forall k_1 \neq k_2$.
IIR	Infinite impulse response.
LL-model	A parameterized model which is linear both in the parameters and the input. The nomenclature is also used for filters
LN-model	A parameterized model which is linear in the parameters and nonlinear in the input. The nomenclature is also used for filters.
LX-model	A parameterized model which is linear in the parameters and may be a linear as well as nonlinear in the input. The nomenclature is also used for filters.
LMS	L east M ean S quares.
LS	L east S quares.
MFNN	M ulti-layer F eed-forward N eural N etwork.
MFPNN	M ulti-layer F eed-forward P erceptron N eural N etwork.
MGN	M odified G auss- N ewton algorithm.
MIL	M atrix I nversion L emma.
ML	M aximum L ikelihood.
MSME	M ean S quare M odel E rror.
MSE	M ean S quare E rror.

NN-model	A parameterized model which is nonlinear both in the parameters and the input. The nomenclature is also used for filters.
<i>NB</i>	N ormalized B ias.
NSG	N ormalized S tochastic G radient.
OBD	O ptimal B rain D amage.
OBS	O ptimal B rain S urgeon.
PCA	P rincipal C omponent A nalysis.
p.d.f.	Probability density function.
PFF	P artition F unction F ilter.
PPA	P reprocessing A lgorithm.
PWLF	P iecewise- L inear F ilter.
q.e.	Quod est (which means).
Q.E.D.	Quod erat demonstrandum (which was to be proved).
RGN	R ecursive G auss- N ewton.
RGNB	R ecursive G auss- N ewton with B ierman factorization.
RLS	R ecursive L east S quares.
SFI	S teppwise F orward I nclusion.
SPA	S tatistical P runing A lgorithm.
SG	S tochastic G radient.
<i>SNR</i>	S ignal-to- N oise R atio.
WDV	W eight D eletion V ector.
viz.	Videlicet (namely).
w.r.t.	With respect to.
<i>WFP</i>	W eight F luctuation P enalty.
XN-model	A parameterized model which is nonlinear in the the input and may be linear as well as nonlinear in the parameters. The nomenclature is also used for filters

CHAPTER 1

INTRODUCTION

In recent years much attention was directed to *adaptive models* as devices for design of flexible signal processing systems. Adaptive models may display the following advantageous properties:

- The ability to learn a signal processing task from acquired examples of how the task should be resolved. A general task is to model a relation between two signals. In this case the learning examples simply are related samples of these signals. The learning (also referred to as supervised learning) is often done by adjusting some parameters (weights) such that some cost function is minimized. This property may be valuable in situations where it is difficult or impossible to exactly explain the physical mechanisms involved in the task.
- The possibility of continuously tracking changes in the environments (i.e., handling of nonstationary signals).
- The ability of generalizing to cases which were not explicitly specified by the learning examples. For instance, the ability to estimate the relation between two signals which were not used for training the filter.

The effort has up to now mainly been on *linear adaptive models* which have been successfully applied to various signal processing tasks including: System identification, adaptive control systems, equalization of communication channels, noise reduction, speech coding, prediction of time series [Haykin 91], [Widrow & Stearns 85]. Often it appears that linear models are approximations of systems which fundamentally are *nonlinear* by nature. The approximation may be suitable in many cases; however, a large potential consists in extending the models to be nonlinear, e.g., [Bendat 90], [DARPA 88], [Priestley 88], [Schetzen 89], [Seber & Wild 89]. An illustrative example is that many physical systems may display very complex behavior such as chaos, limit cycles, and bifurcations (e.g., [Thomson & Stewart 86]); consequently, they are nonlinear. However, dealing with nonlinear models involve some immediate disadvantages:

- The class of nonlinear models contains, in principle, all models which are not linear. The problem, which arises, is how to delimit subclasses of nonlinear models so that they are applicable within a wide range of signal processing tasks.
- The computational complexity faced with nonlinear models will often be huge compared to linear models.

- Theoretical analysis becomes very difficult.

In this Thesis we will focus on *neural networks as devices for designing nonlinear, adaptive filters* for the purpose of modeling *discrete*¹, nonlinear, systems.

1.1 Historical Outline

Research within artificial neural networks is inspired by neurobiological studies of the processes in the human brain. The human brain consists of a huge number of neurons which are connected in a network. This enables the neurons to mutually exchange information in order to collaborate on the control of the human body. By artificial neural networks we consider computational models of the real neural networks². The research was initiated by McCulloch & Pitts [McCulloch & Pitts 43] in 1943 who proposed a simple parametric nonlinear computational model of a real neuron. Rosenblatt [Rosenblatt 58], [Rosenblatt 62] proposed around 1960 a layered neural network consisting of *perceptrons* and an algorithm for adjusting the parameters of single layer perceptron network so that the network was able to implement a desired task. At the same time Widrow and Hoff [Widrow & Hoff 60] proposed the MADALINE neural network which resembles the perceptron network. Widrow pioneered the use of neural networks within signal processing and in [Widrow & Lehr 90] a review of this work can be found. However, in 1969 Minsky and Papert interrupted the development by showing that the one-layer perceptron network was not capable of implementing simple tasks (as e.g., the XOR problem) and algorithms for adjusting the weights of multi layered perceptron networks were not invented. Recently a new edition was published [Minsky & Papert 88] which relaxes some of the firm statements done in the 1969 edition.

In the 1950's and 1960's effort was also put into other types of nonlinear phenomena, e.g., the study of the Wiener model, Hammerstein model, the Gate function model and the study of chaotic motion [Haddad 75], [Schetzen 89], [Thomson & Stewart 86].

Until the 1980's the interest on nonlinear systems and neural networks became sparse. However, the extensively increased power of the computers in the 1980's enabled to study more complex phenomena and a lot of progress was made within the study of chaos. Furthermore, around 1985 [McClelland & Rumelhart 86] an algorithm for adjusting the parameters (learning) of a multi-layered neural network – known as the *back-propagation algorithm* – was rediscovered. This turned on an enormous interest for neural networks. The reader is referred to [Haykin 94], [Hertz et al. 91], [Hush & Horne 93], [White 89a] for recent reviews on the progress within neural networks.

In the DARPA study (1988) [DARPA 88] a number of prominent neural network scientists devised the directions of the future neural network studies. They concluded that neural networks may provide a very useful tool within a broad range of applications. Recently the Danish Research Councils (SNF, STVF) supported the establishment of The Computational Neural Network Center (CONNECT) which studies the theory, implementation and application of neural computation.

¹In the signal processing literature the term “digital” is often used.

²In the rest of this thesis we for simplicity omit the term “artificial”.

1.2 Nonlinear Models

Consider modeling related input-output data which are generated by a – in general – nonlinear *system*. In the most general form a discrete parametric *model* is described by the following equation ³:

$$\mathbf{F}(\mathbf{X}(k), \mathbf{Y}(k), \mathbf{E}(k); \mathbf{w}(k)) = \mathbf{0} \quad (1.1)$$

where

- k is the discrete time index, i.e., the model corresponds to an analog model sampled at the time instants: k/f_s , where f_s is the sampling frequency.
- \mathbf{F} is a vector function, which maps a vector space into a vector space.
- $\mathbf{X}(k) = [\mathbf{x}_1(k), \mathbf{x}_2(k), \dots, \mathbf{x}_I(k)]$ is the multivariate input (I is the number of inputs) at time k with $\mathbf{x}_i(k) = [x_i(k), x_i(k-1), \dots]^\top$ (Here $^\top$ denotes the transpose operator).
- $\mathbf{Y}(k)$ is the multivariate output at time k .
- $\mathbf{E}(k)$ is the multivariate (non-observable) error which represents the uncertainty on the output when the input is known.
- $\mathbf{w}(k)$ is the parameter (weight) vector which specifies the actual model within the class of models given by \mathbf{F} . In order to track changes in the nonlinear system $\mathbf{w}(k)$ must be time-varying. Unless something else is mentioned explicitly, we assume that the model contains m parameters.

However, in this Thesis the general model is restricted to comply with the *non-recursive model with additive error*⁴:

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \quad (1.2)$$

where

- $y(k)$ is a one dimensional output signal.
- $\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-L+1)]^\top$ is the input vector signal and L the memory length (window length).
- The *filtering function* $f(\cdot)$ maps the input vector into the output and is parameterized by the m -dimensional parameter vector \mathbf{w} . The mapping belongs to a set of functionals, \mathcal{F} , which will be referred to as the *filter architecture*. The term architecture is used to cover the construction of these functionals. The filtering function, $f(\cdot) \in \mathcal{F}$, thus represents a specific member of the architecture \mathcal{F} .
- $e(k; \mathbf{w})$ is the error. Often the dependence on the parameters is omitted for simplicity.

³Further elaborations on general models may be found in e.g., [Leontaritis & Billings 85], [Priestley 88].

⁴This model may also be seen as a special version of the **Nonlinear autoregressive model with exogenous input** (NARX) [Leontaritis & Billings 85].

When applying the model to simulate an underlying system we use the filter⁵:

$$\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}), \quad (1.3)$$

$\hat{y}(k)$ is denoted the *filter output* or the *prediction*. The last designation stems from the fact that $\hat{y}(k)$ predicts the output $y(k)$. This is the best we can do since the error, per definition, is non-observable. The model with additive error is shown in Fig. 1.1.

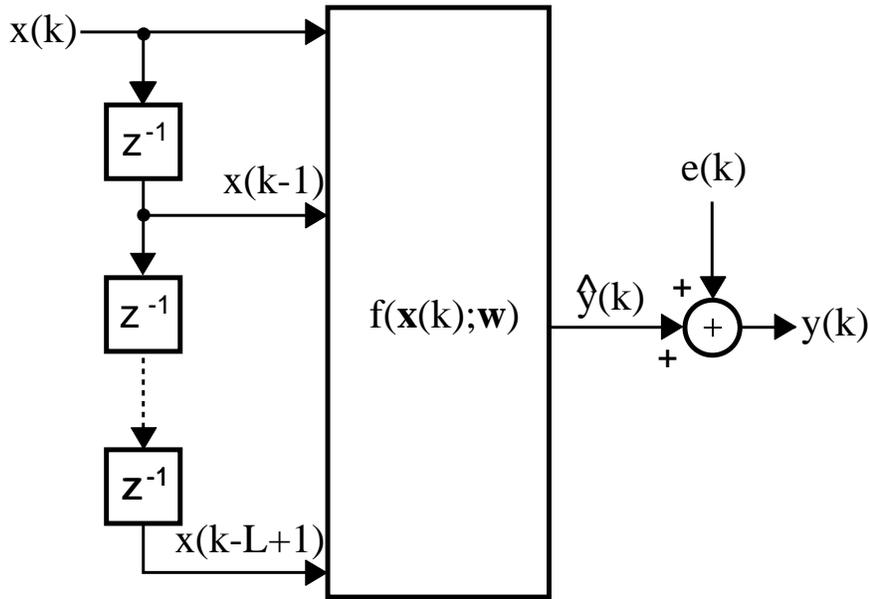


Figure 1.1: The general non-recursive model with additive error. z^{-1} denotes the unit delay operator.

In connection with the model in int:model the following limitations are introduced:

- We consider single-input-single-output models only (SISO).
- The model is non-recursive and thus with finite memory as L is finite and no delayed output samples $y(k-n)$, $n > 0$ enters the model. In the literature several authors indeed has suggested to incorporate the recursive part, e.g., [Chen & Billings 89], [Narendra & Parthasarathy 90], [Priestley 88] and [Tong & Lim 80]. However, the complex behavior of recursive models e.g., chaos and limit cycles have not been addressed especially and may cause serious problems with respect to the design of robust models.
- The parameter vector \mathbf{w} is assumed to be independent of time and the filtering function $f(\cdot)$ does not depend on time explicitly. That is, the tracking abilities of the model by allowing \mathbf{w} to be time varying is not under consideration.

⁵Note that this filter may be viewed as an extension of the classical adaptive, linear FIR-filter [Haykin 91], [Widrow & Lehr 90]: $\hat{y}(k) = \mathbf{w}^T \mathbf{x}(k)$.

- The error, $e(k)$, is assumed to be additive.
- The input and the output signals are considered to be stationary stochastic processes and furthermore real valued; that is, we deal with regression or functional approximation type of problems. In particular, classification and pattern recognition problems are not under consideration. Consequently, we focus on neural network paradigms which enable us to perform nonlinear regression tasks. In addition, quantization effects will not be treated.

Four main structures of the filter architecture appear as, $f(\cdot)$, can be a linear or nonlinear function of both the input and the parameter vector. Table 1.1 shows the four structures along with the nomenclature used in the following. $\varphi(\cdot)$ denotes an arbitrary vector function. The nomenclature consists of a 2-tuple: $A_{\text{par}}A_{\text{in}}$, where $A_{\text{par}} \in \{L, N, X\}$

Input Parameters	Linear	Nonlinear
Linear	LL-model $f(\mathbf{x}(k); \mathbf{w}) = \mathbf{w}^T \mathbf{x}(k)$	LN-model $f(\mathbf{x}(k); \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}(k))$
Nonlinear	NL-model $f(\mathbf{x}(k); \mathbf{w}) = \phi^T(\mathbf{w}) \mathbf{x}(k)$	NN-model $f(\mathbf{x}(k); \mathbf{w})$

Table 1.1: Main structures of the filter architecture \mathcal{F} and the corresponding nomenclatures. L refers to linear, N to nonlinear, and X to either linear or nonlinear. If the filter architecture is not parameterized the first element of the 2-tuple is set equal to X.

denotes whether $f(\cdot)$ is linear (L), nonlinear (N), either linear or nonlinear (X) dependent on the parameters, \mathbf{w} . If the filter architecture is not parameterized we set $A_{\text{par}} = X$. Similarly, A_{in} denotes the functional dependence on the input vector, $\mathbf{x}(k)$. In this Thesis we mainly deal with XN-models and filters. It should be emphasized that the NL-model is included for the sake of completeness only as it normally is irrelevant. This is due to the fact that it is possible to make the following redefinition: $\mathbf{w} \leftarrow \varphi(\mathbf{w})$.

This Thesis concentrates mainly on neural network architectures for the implementation of the filter in int:nonfilt. In this context we make the following definition (see also e.g., [Hect-Nielsen 89]):

DEFINITION 1.1 A neural network filter architecture consists of a number of processing elements, called neurons, which are connected in a network. The neurons collaborate in order to produce the mapping from an input vector to an output vector. Each neuron has a single output and receives inputs from other neurons (possibly its own output), perhaps from the input signal vector. In general, there are no restrictions concerning the processing within the neurons. The neurons which produce the output are denoted output neurons and any neuron which is not an output neuron is called a hidden neuron.

In Fig. 1.2 an example of a general single-output neural network filter is shown⁶. In the

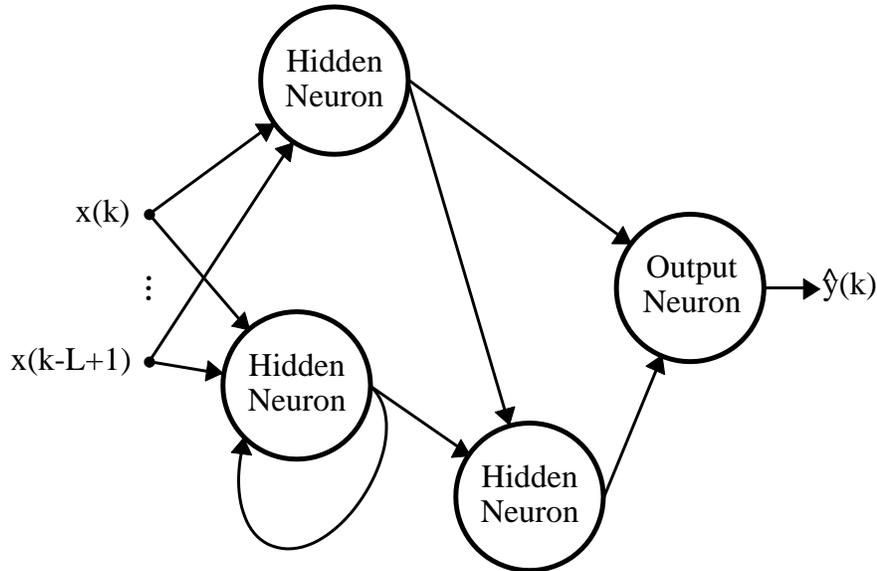


Figure 1.2: Principal chart of a neural network filter. The network consists of a number of processing elements (neurons) which collaborate in order to produce a nonlinear mapping of the input vector signal.

following chapters we will differentiate between two types of neurons: *Nonlinear neurons* and *linear neurons*. Let the m_{r-1} inputs to an arbitrary neuron (numbered i) be given by $\tilde{\mathbf{s}}^{(r-1)} = [s_1^{(r-1)}, s_2^{(r-1)}, \dots, s_{m_{r-1}}^{(r-1)}]^\top$ and let $s_i^{(r+1)}$ be the (single) output. If the neuron is nonlinear then the processing yields:

$$s_i^{(r)} = \varphi(\tilde{\mathbf{s}}^{(r-1)}; \mathbf{w}_i^{(r)}) \quad (1.4)$$

where $\varphi(\cdot)$ is a nonlinear mapping and $\mathbf{w}_i^{(r)} = \{w_{ij}^{(r)}\}$, $j = 0, 1, \dots$, is a parameter vector associated with neuron numbered i . In Ch. 3 special cases of $\varphi(\cdot)$ is considered. However, if the neuron is linear then:

$$s_i^{(r)} = w_{i0}^{(r)} + \sum_{j=1}^{m_{r-1}} s_j^{(r-1)} w_{ij}^{(r)} = [1, (\tilde{\mathbf{s}}^{(r-1)})^\top] \mathbf{w}_i^{(r)} = (\mathbf{s}^{(r-1)})^\top \mathbf{w}_i^{(r)} \quad (1.5)$$

⁶Notice that only one output neuron is relevant in connection with the filter in int:nonfilt as the output is a scalar.

where $\mathbf{s}^{(r)}$ is the *augmented* input vector. The linear and nonlinear neurons along with their pictographs are shown in Fig. 1.3.

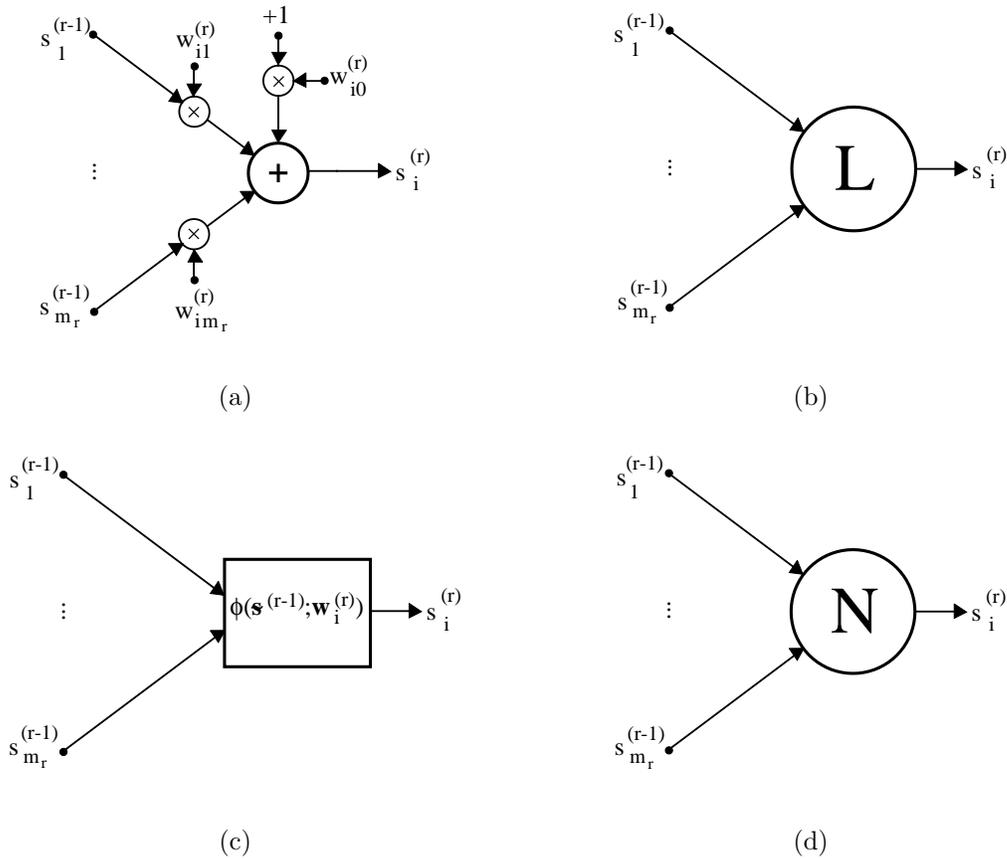


Figure 1.3: The processings involved in the linear and nonlinear neuron are shown on the left images. The right images show the corresponding pictographs which often will be used in the following chapters.

1.3 Fields of Applications

The model in int:model may be used for various general adaptive signal processing tasks [Widrow & Stearns 85] such as:

- System identification,
- Inverse modeling,
- Prediction and time series identification, and
- Adaptive noise cancellation.

which briefly are discussed below.

System Identification In this case we consider a system with input $x_s(k)$ and output $y_s(k)$ ⁷ given as:

$$y_s(k) = g(\mathbf{x}_s(k)) + \varepsilon(k) \quad (1.6)$$

where $g(\cdot)$ constitutes a (nonlinear) mapping, $\mathbf{x}_s(k)$ is the input vector signal, and $\varepsilon(k)$ is an inherent (non-observable) stochastic noise which explains lack of information about $y_s(k)$ contained in $\mathbf{x}_s(k)$, e.g., measurement noise.

The system is identified by using the model,

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \quad (1.7)$$

where $y(k) = y_s(k)$ and $x(k) = x_s(k)$, and adjusting the parameters \mathbf{w} (in Fig. 1.4 this is depicted by the arrow passing through the filter) so that the error, $e(k)$ is eliminated. The filter is then said to be in the *adaptation* or *estimation mode*. For instance, the parameters may be estimated so that the mean squared error is minimized (the LS criterion), see further Ch. 5. If the structure of $g(\cdot)$ resembles the structure of $f(\cdot)$ for a given set of parameters the error may become close to the inherent noise. When the model has been estimated (adapted), novel input samples can be filtered in order to obtain estimates of $y(k)$. This mode is denoted the *filtering mode*. Fig. 1.4 shows the adaptation and filtering modes when applying the model in int:model for the system identification task.

Using neural networks for this task – e.g., in connection with control of dynamical systems – have been proposed by [Levin et al. 91], [Narendra & Parthasarathy 90], and [Nguyen & Widrow 89].

Inverse Modeling The inverse modeling problem is to “deconvolve” or equalize a system as shown in Fig. 1.5. In this context we consider a recursive, causal system with input $x_s(k)$ and output $y_s(k)$ which is given by:

$$y_s(k) = g(y_s(k-1), y_s(k-2), \dots, y_s(k-P+1)) + x_s(k) + \varepsilon(k). \quad (1.8)$$

Let the output of the system be equal to the input of the model, i.e., $y_s(k) = x(k)$. Further let the output of the model be equal to the delayed input of the system, i.e., $y(k) = x_s(k-D)$, where $D \geq 0$ is the delay. The delay is inserted due to the fact that the system is assumed to be causal which results in that the output of the system responds to changes at the input with a certain delay. That is, it may be better to predict $x_s(k)$ on the output samples: $y(k+D), y(k+D-1), \dots, y(k+D-L+1)$. The reason for considering a recursive system is easily seen by substituting the equalities: $y_s(k) = x(k)$ and $y(k) = x_s(k-D)$ into int:invmodsys. That is:

$$x(k) = g(x(k-1), x(k-2), \dots, x(k-P+1)) + y(k+D) + \varepsilon(k). \quad (1.9)$$

Rewriting yields:

$$y(k) = x(k-D) - g(x(k-D-1), x(k-D-2), \dots, x(k-D-P+1)) - \varepsilon(k). \quad (1.10)$$

This equation is equivalent to the structure of the non-recursive model in int:model. However, if the system includes delayed input signals, i.e., $x_s(k-i)$, $i > 0$, the non-recursive filter may still be applied; hence, we expect that the memory length, L , has to be relatively large.

⁷Subscript $_s$ is used to emphasize that the signals are associated with the system rather than the model. However, if no confusion is possible the subscripts are omitted.

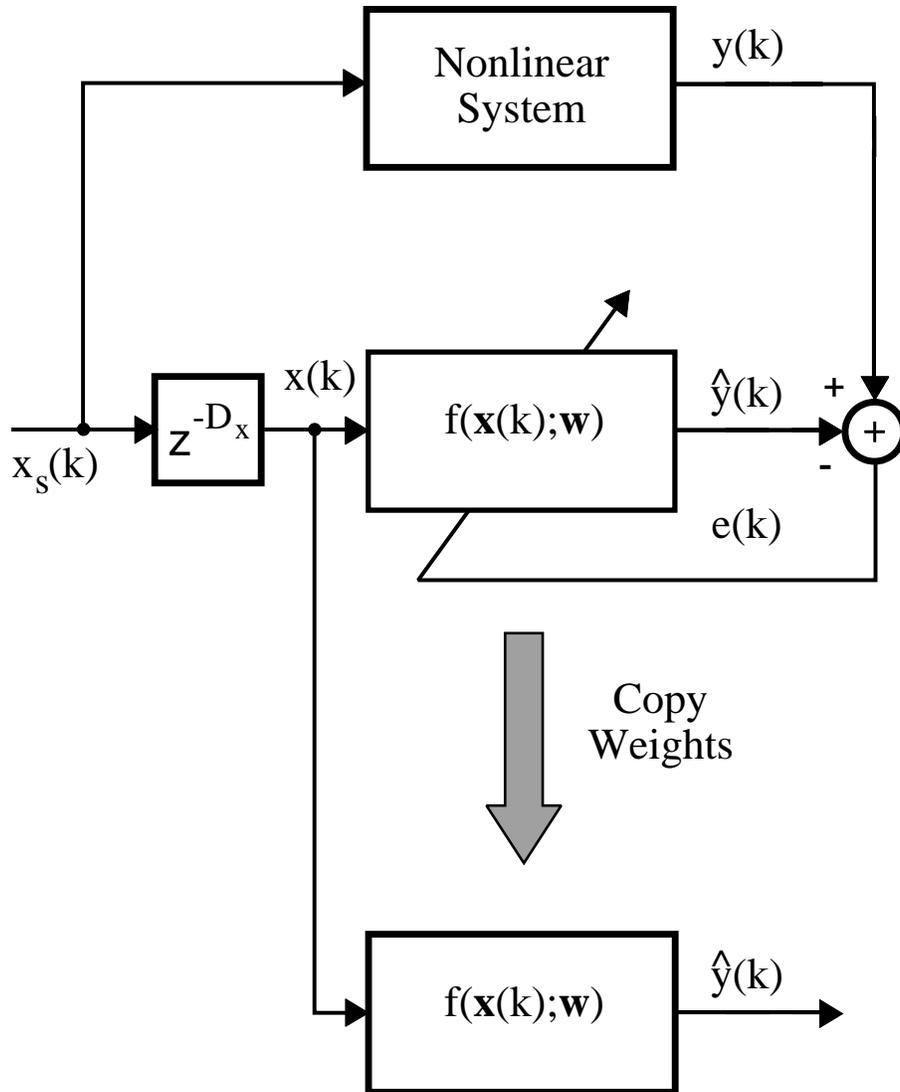


Figure 1.4: Adaptation (top) and filtering (bottom) modes in the system identification case.

In [Chen et al. 90c] a neural network filter was successfully used for equalization of a communication channel. However, also other types of nonlinear filters have shown to be useful within channel equalization, e.g., [Falconer 78].

Prediction and Time Series Identification Using the filter as a predictor (forecasting of time series) is shown in Fig. 1.6. The model is estimated by letting the input of the model be equal to a delayed sample of the time series, $y_s(k)$, which we want to predict. That is, afterwards the series is predicted by feeding the filter with $x(k) = y_s(k)$. The

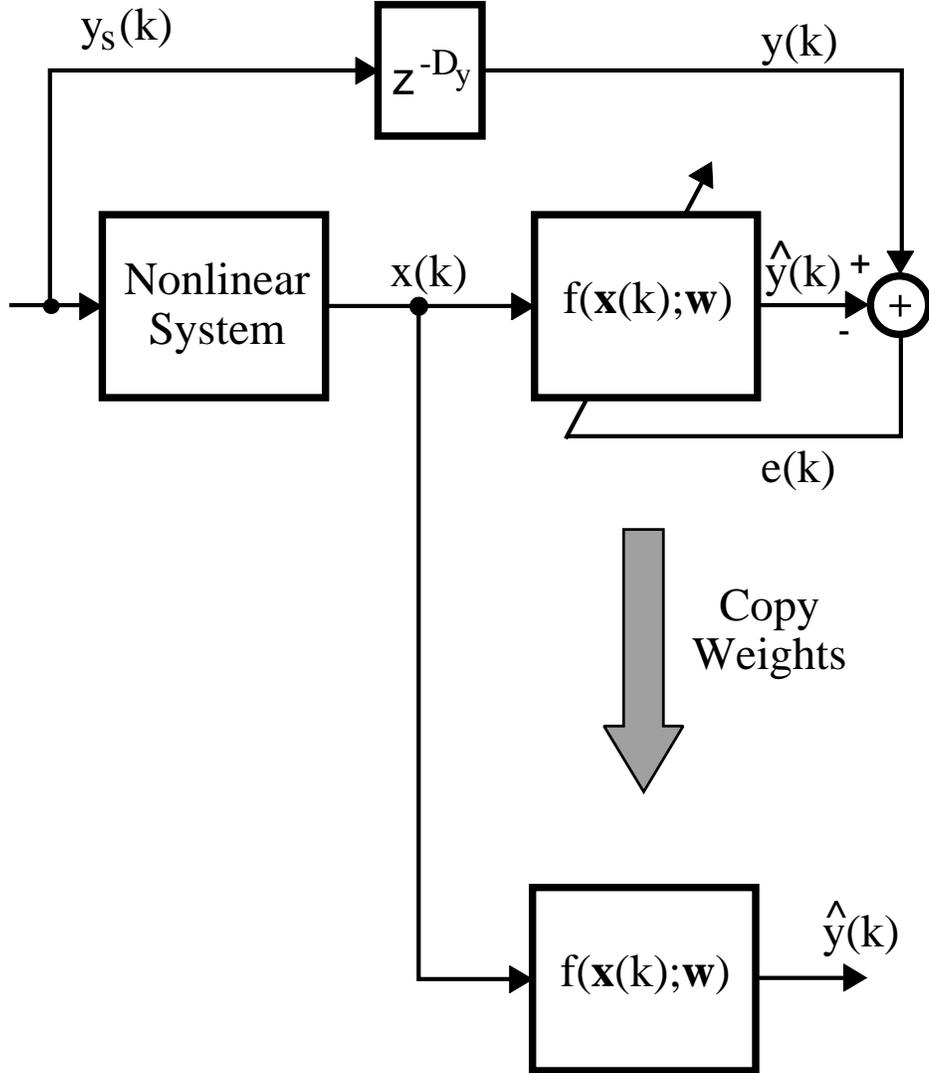


Figure 1.5: Adaptation (top) and filtering (bottom) modes within inverse modeling.

prediction of the times series D_x samples ahead then becomes:

$$\hat{y}_s(k + D_x) = f(\mathbf{y}_s(k); \hat{\mathbf{w}}) \quad (1.11)$$

where $\hat{\mathbf{w}}$ are the estimated parameters.

Another application of the present configuration is the identification of time series. Let a time series, $y_s(k)$, be generated by the recursive system (i.e., $y_s(k)$ is a nonlinear autoregressive process (NAR) [Leontaritis & Billings 85]):

$$y_s(k) = g(y_s(k-1), y_s(k-2), \dots, y_s(k-P+1)) + \varepsilon(k) \quad (1.12)$$

where $\varepsilon(k)$ is a non-observable random uncorrelated (white) sequence. By estimating a non-recursive model with output $y(k) = y_s(k)$, input $x(k) = y_s(k-1)$, and $L = P$ we

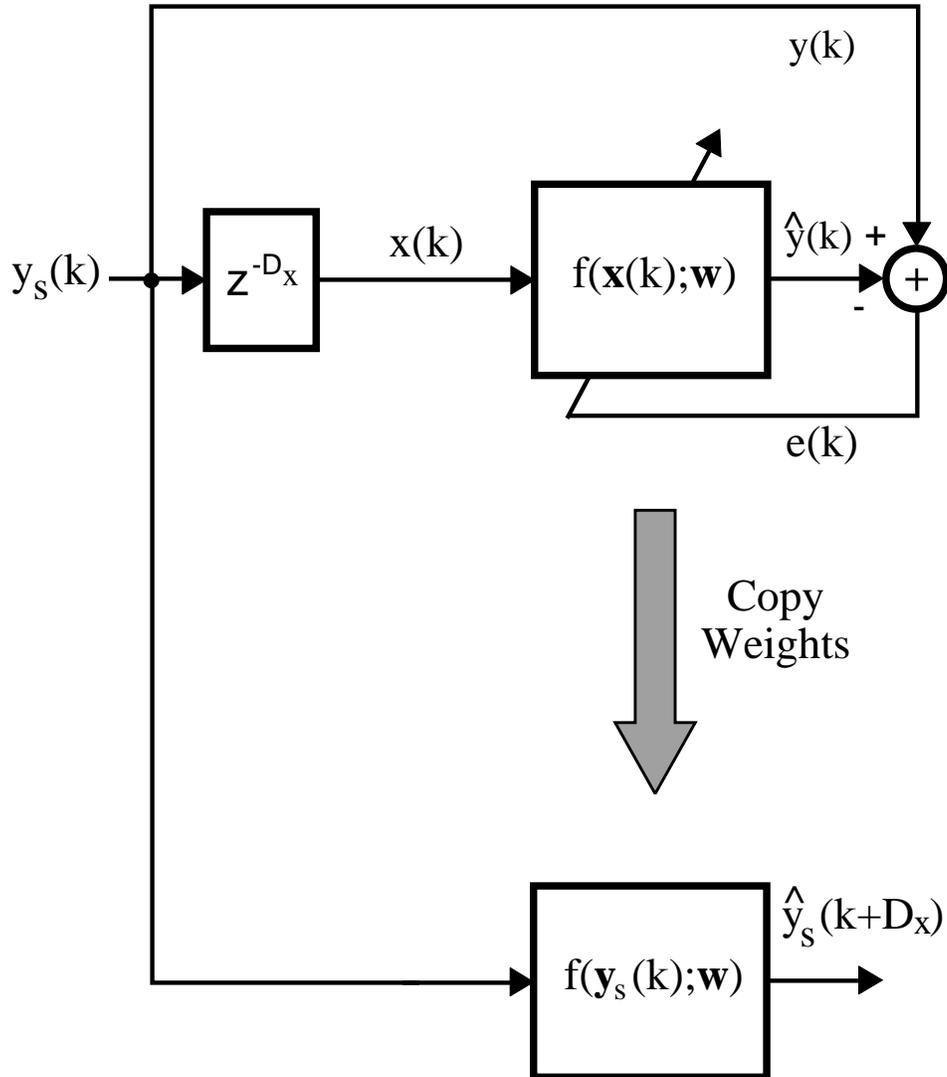


Figure 1.6: Adaptation (top) and filtering (bottom) modes within prediction and time series identification.

notice that the process is identified provided that $f(\cdot)$ suitably models $g(\cdot)$ and that a convenient algorithm is used for estimating the model parameters (see Ch. 5).

In the literature several attempts based on neural networks have been applied for the purpose of predicting time series, e.g., [Lapedes & Farber 87], [Poddar & Unnikrishnan 91], [Weigend et al. 90], [White 88].

Adaptive Noise Cancellation The adaptive noise cancellation configuration [Widrow & Stearns 85] shown in Fig. 1.7 resembles the structure of the inverse modeling configuration. The task is to cancel noise which interferes with a desired signal. The

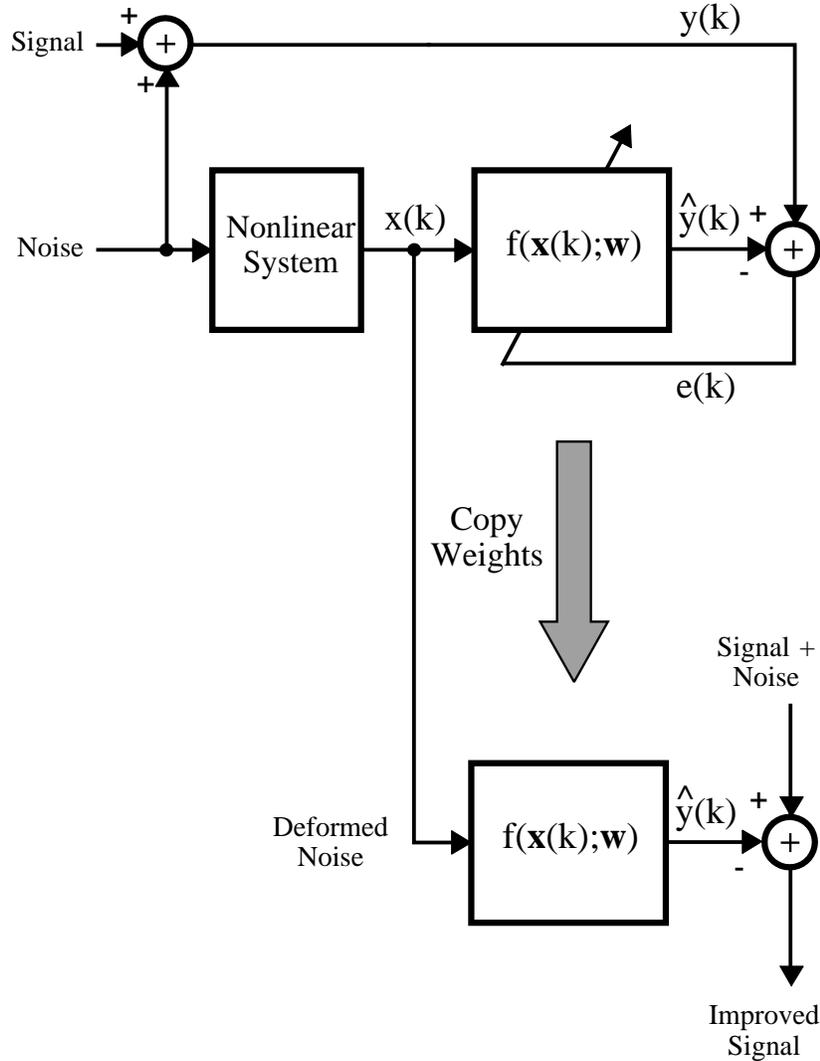


Figure 1.7: Adaptation (top) and filtering (bottom) modes within the adaptive noise canceler.

corrupted signal is denoted $y(k)$. The basic idea is that a deformed version of the noise, $x(k)$, is available. If it is possible to equalize the deformed noise by an XN-filter, $\hat{y}(k)$ becomes equal to the noise and the error then equals the original signal. An essential assumption is, however, that the noise is independent⁸ on the desired signal. A recent paper on this configuration is [Giannakis & Dandawate 90].

⁸Subsidiarily, appropriate n 'th order correlations between the noise and the desired signal are equal to zero.

1.4 Designing Neural Network Filters

The objective of this Thesis is to study the design of adaptive XN-filters based on neural networks. The design involves a number of phases which are depicted in Fig. 1.8. The first phase involved is the collection of data which describe the task to be mod-

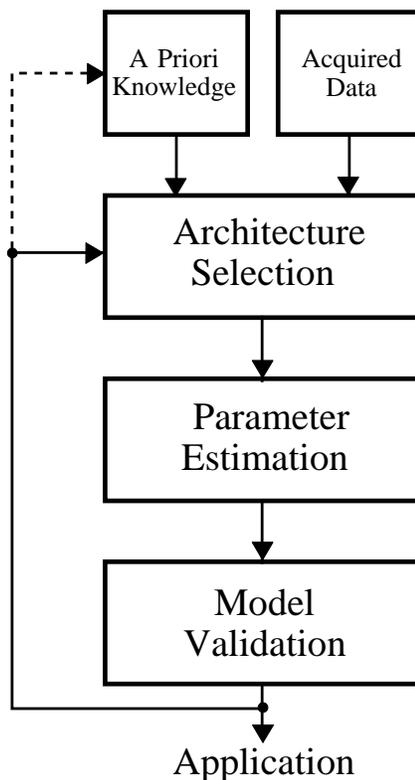


Figure 1.8: Principle chart of the design procedure.

eled. General notes concerning data acquisition are omitted; the reader is referred to e.g., [Bendat & Piersol 86, Ch. 10]. A serious problem is that in order to estimate a complex model a huge number of data must be available. The lack of data may consequently often result in a suboptimal model.

The second phase is the a priori knowledge concerning the task under consideration. A priori knowledge may be due to basic physical analysis or knowledge gained by estimating relevant features such as Fourier spectra, plots of related signals, etc.. This phase is of course important and should be incorporated in the chosen architecture as much as possible. However, in the general case it is difficult to state how this should be done. In Ch. 6 this topic is further elaborated.

The third phase is the selection of a proper model architecture based on the a priori knowledge and the type and the amount of data available. Usually a catalog of alternative

architecture exists. In Ch. 3 a variety of architectures is described and Ch. 4 presents a generic neural network architecture.

When an architecture has been selected the characteristic model parameters are estimated on the available data. Algorithms for model parameter estimation are further treated in Ch. 5.

Finally, the estimated model is validated, that is, it is tested whether the model satisfies the requirements specified by the actual task. A specific validation criterion is the generalization error which states how the model will perform on future data, i.e., data which were not used in the estimation of model parameters. In Ch. 6 a general framework for estimation of the generalization error is presented. Based on the result of the validation phase we may accept the current model or otherwise, return to the architecture selection phase; subsidiary, make more enquiries about the physical conditions or collect more data. Procedures for synthesizing filter architectures are further described in Ch. 6.

1.5 Summary

This Thesis focus on neural networks as devices for designing adaptive XN-models (i.e., models in which the output is a nonlinear function of the input). The models under consideration are nonrecursive and the error is assumed additive.

The benefits and perspectives of using adaptive XN-filters within various signal processing applications such as: System identification, inverse modeling, prediction and time series identification, and adaptive noise cancellation, are mentioned.

Finally, a principle chart of designing XN-filters based on neural networks is presented. The chart forms the basis for the topics under consideration in the subsequent chapters.

CHAPTER 2

NONLINEAR FILTER ANALYSIS

In this chapter a brief review of the classical theory for characterization of nonlinear filters (i.e., XN-filters) is presented. Even though this theory provides insight in the functionality of various special filter architectures it is emphasized that the theory does not appear to be profitable when analyzing general adaptive XN-filters. In consequence one may resort to a partial analysis which is highly dependent on the specific architecture and an estimate of how well the filter performs.

2.1 Basic Properties of Nonlinear Filters

Consider an XN-model (see Section 1.2)

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k), \quad (2.1)$$

and the corresponding XN-filter

$$\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}) \quad (2.2)$$

where

- $y(k)$ is the model output and $\hat{y}(k)$ is the filter output (also referred to as the prediction). Recall that the task of the filter is to predict $y(k)$ so that a suitable norm of the error, $e(k)$, is as small as possible.
- $\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-L+1)]^\top$ is the input vector and L is the memory length (window length). $L = 0$ is denoted a *zero-memory* filter, $L < \infty$ a *finite-memory* filter, and $L \rightarrow \infty$ an *infinite-memory* filter¹.
- \mathbf{w} is the m -dimensional parameter vector which determines the specific filtering function $f(\cdot) \in \mathcal{F}$, where \mathcal{F} denotes the actual (filter) architecture.

¹An infinite-memory filter is a mathematical abstraction. The actual output, $\hat{y}(k)$, of a recursive filter depends on all input samples $x(0), x(1), \dots, x(k)$, where $k = 0$ denotes the start of the filtering. Consequently, the memory grows at each time step and reaches infinity as $k \rightarrow \infty$. A recursive filter is therefore said to possess infinite memory. If the filter is linear the finite-memory and the infinite-memory filter correspond to the FIR and the IIR filter, respectively.

In this section the filtering function $f(\cdot)$ is often substituted by the nonlinear operator $F[\cdot]$, i.e., $F[x(k)] \equiv f(\mathbf{x}(k); \mathbf{w})$.

XN-filters can be characterized by some basic properties which are a direct extension of those known from the theory of linear filters.

2.1.1 Superposition

A fundamental property of XN-filters is that the superposition theorem is invalid, i.e.,

$$F[c_1x_1(k) + c_2x_2(k)] \neq c_1F[x_1(k)] + c_2F[x_2(k)] \quad (2.3)$$

where c_1, c_2 are arbitrary constants and $x_1(k), x_2(k)$ are two different input signals. This lack reduces the possibilities of stating nice analytical results contrary to dealing with linear filters. One of the consequences is that frequency-domain interpretations such as the transfer function (\mathbf{z} -transform) and the Fourier spectrum in general² are precluded.

Three immediate consequences are:

1. A change in mean value of the input signal can cause the filter to respond totally different.
2. The filter may be highly sensitive to a scaling (amplification) of the input signal.
3. It is not possible to perform a decomposition of the input signals. When dealing with linear filters the input signal is usually decomposed into a sum of deterministic and stochastic signals³ allowing for an individual treatment of these signal types. Hence, all signals are treated as stochastic unless they are purely deterministic.

2.1.2 Time-Invariance

If $\hat{y}(k)$ is the filter response due to the input $x(k)$, i.e., $\hat{y}(k) = F[x(k)]$, then the filter is said to be *time-invariant* if

$$\hat{y}(k + \tau) = F[x(k + \tau)], \quad \forall \tau. \quad (2.4)$$

In order to track alterations in a time-varying system the filter has to be time-varying; however, in this Thesis we consider time-invariant filters only.

In this context we emphasize the existence of an interesting pseudo duality between time-varying linear filters and time-invariant XN-filters. This is illustrated by the following example: Let the signal $y(k)$ be generated by the time-invariant (unknown) nonlinear system

$$y(k) = g(\mathbf{x}(k)) \cdot x(k) \quad (2.5)$$

where $x(k)$ is a stationary stochastic process and $g(\cdot)$ a nonlinear function which does not depend on time explicitly. Consequently, $y(k)$ is stationary. It is assumed that the variation of the signal $g(\mathbf{x}(k))$ is “slow” (i.e., the spectrum is dominated by low frequencies) compared to the variations in the input signal, $x(k)$. Due to the fact that $y(k)$ is observed in limited time one may not be able to recognize that $g(\mathbf{x}(k))$ in fact is a stationary process. $y(k)$ may therefore be viewed as a quasi-stationary (subsidiary piecewise-stationary) signal

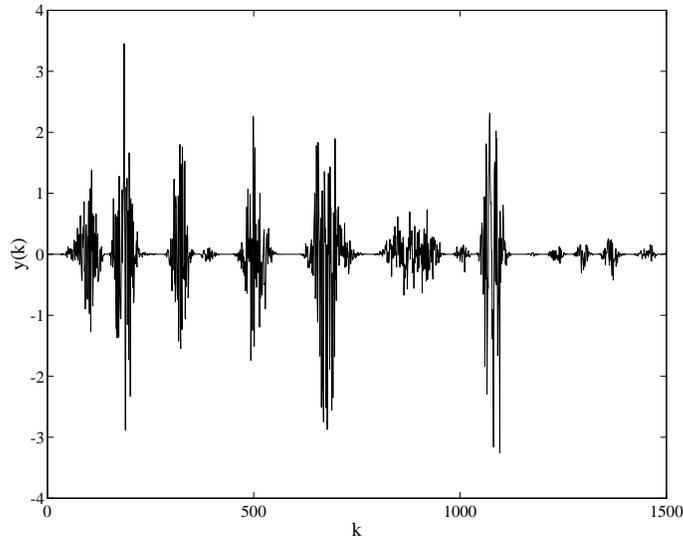


Figure 2.1: An example of $y(k)$ generated according to `fa:nonsys`. We used $g(\mathbf{x}(k)) = 75(h(n) * x(n))^3$ where $*$ denotes convolution. $x(n) \in \mathcal{N}(0, 1)$, i.e., a standard Gaussian distribution and $h(n)$ is a FIR-filter with 500 taps designed to implement a low-pass filter with normalized cutoff frequency 0.02 and DC gain equal to one.

which depends linearly on the input⁴. In Fig. 2.1 an example of $y(k)$ is shown. A model of the system described above thus becomes

$$\hat{y}(k) = w(k)x(k) \tag{2.6}$$

where $w(k) = w_i$ as $k \in [n_i; n_{i+1}]$ and $n_i, i = 1, 2, \dots$ defines a time segmentation so that the model becomes linear within each time segment $[n_i; n_{i+1}]$. Thus the model is a piecewise-stationary linear model. However, it may eventually be more efficient to use a XN-model.

Further support of the claimed pseudo duality can be found in the following theorem [Bendat 90, Ch. 3.3], [Schetzen 89]:

THEOREM 2.1 (SCHETZEN) *Any time-varying linear system can be synthesized by a time-invariant bilinear system and the bilinear system is causal if and only if the time-varying linear system is causal.*

The definition of a bilinear system (model) is given in the subsection below. The theorem is; however, of minor practical interest since the time-invariant bilinear system depends on the actual input signal $x(k)$.

²However, dealing with e.g., Volterra filters allows for a frequency-domain approach. This is discussed below.

³A deterministic signal (or process) is defined as a signal which can be perfectly predicted from its own past. An obvious example is a periodic signal.

⁴For a rigorous approach on the differentiation between stationary and nonstationary processes the reader is referred to e.g., [Bendat & Piersol 86, Ch. 12], [Priestley 88, Ch. 6]

2.1.3 Stability

An XN-filter is BIBO (bounded-input-bounded-output) stable⁵ if $|\hat{y}(k)| < K_1$ provided that the input is bounded, i.e., $|x(k)| < K_2$. Clearly if the filter is a finite-memory filter ($L < \infty$) it is BIBO stable if $|f(\cdot, \mathbf{w})| < \infty, \forall \mathbf{w}$. This is normally met as $\|\mathbf{w}\| < \infty$ where $\|\cdot\|$ denotes any vector norm.

2.1.4 Causality

The filter is causal (physically realizable) if

$$x(k) \equiv 0, k < k_0 \Rightarrow \hat{y}(k) \equiv 0, k < k_0 \quad (2.7)$$

where k_0 is any fixed time index. The nonlinear system involved in the filtering task is normally assumed to be causal; however in connection with inverse modeling (see Ch. 1) we mentioned the necessity of noncausal filtering, i.e., the model output $\hat{y}(k)$ has to be a function of the input vector $\mathbf{x}(k+D) = [x(k+D), x(k+D-1), \dots, x(k-D-L+1)]^\top$, $D \geq 0$. Noncausal filtering is simply done by redefining the time index ($k \leftarrow k-D$ where \leftarrow is the assign operator) so that the filter output, $\hat{y}(k)$, due to the input $\mathbf{x}(k)$ is a prediction of $\hat{y}(k-D)$.

2.2 Analysis of Nonlinear Filters

The theoretical analysis of the functionality of XN-filters may show useful when designing the filter. This analysis may be regarded as a source of *a priori knowledge* (see the figure on page 11). That is, it may facilitate the choice of a proper filter architecture. On the other hand, the analysis of general XN-filter architectures is not possible and furthermore the analysis of specific architectures is often very difficult to carry out.

The principal paradigms for analyzing the functionality of a XN-filter are:

- Expansion of the nonlinear filtering function⁶, $f(\mathbf{x}(k); \mathbf{w})$, in a set of (fixed) simple basis functions, i.e.,

$$f(\mathbf{x}(k); \mathbf{w}) = \sum_i b_i(\mathbf{x}(k)). \quad (2.8)$$

For instance, $b_i(\mathbf{x}(k))$ could represent a Taylor series expansion which, in this context, is denoted a Volterra series expansion. This is further elaborated below. The response of each basis function is now examined individually.

- Linearization of the nonlinear function around a fixed point in the input space, say \mathbf{x}_0 . This technique yields:

$$\begin{aligned} f(\mathbf{x}(k); \mathbf{w}) &\approx f(\mathbf{x}_0; \mathbf{w}) + \frac{\partial f(\mathbf{x}_0; \mathbf{w})}{\partial \mathbf{x}^\top} (\mathbf{x}(k) - \mathbf{x}_0) \\ &\triangleq f_0 + \boldsymbol{\alpha}^\top \mathbf{x}(k) \end{aligned} \quad (2.9)$$

⁵In the literature a variety of different definitions of stability exists; however, this will not be further elaborated.

⁶Recall that $f(\cdot)$ is a multidimensional function as it depends on the entire L -dimensional input vector $\mathbf{x}(k)$. Here the weights, \mathbf{w} , are considered to be fixed.

where

$$f_0 \triangleq f(\mathbf{x}_0; \mathbf{w}) - \frac{\partial f(\mathbf{x}_0; \mathbf{w})}{\partial \mathbf{x}^\top} \mathbf{x}_0, \quad (2.10)$$

$$\boldsymbol{\alpha} \triangleq \frac{\partial f(\mathbf{x}_0; \mathbf{w})}{\partial \mathbf{x}}. \quad (2.11)$$

In the vicinity of \mathbf{x}_0 the output of the XN-filter is approximated by the sum of a linear filter with impulse response $h(k) = a_k$, $k = 1, 2, \dots, L^7$ and a DC-value, f_0 . Of course it is possible to include higher order derivatives of $f(\cdot)$. However, this implies that the simplicity of the linear filter approach is sacrificed. The point of expansion \mathbf{x}_0 ought to vary with time in order to ensure the linear approximation to hold. Hence, the XN-filter is approximated by a time-varying linear filter:

$$f(\mathbf{x}(k); \mathbf{w}) \approx f_0(k) + \boldsymbol{\alpha}^\top(k) \mathbf{x}(k). \quad (2.12)$$

This approach implies that all common analytical tools within linear filtering (e.g., Fourier transform) are applicable in each domain where the coefficients, $\boldsymbol{\alpha}(k)$ and $f_0(k)$, are constant. A crucial drawback of this paradigm is that the distance (w.r.t. some distance measure) between successive input vectors, e.g., $\mathbf{x}(k)$ and $\mathbf{x}(k+1)$, may be large. That is, vectors close in time may be spatially remote. In consequence, the coefficients will be strongly time varying.

In addition, the basis of State-Dependent Models [Priestley 88, Ch. 5] is indeed the present paradigm. The technique is to perform successive linearizations around fixed points in the state space. However, State-Dependent Models will not be treated further in this Thesis.

- Transformation of the input and output in order to ease the interpretation of the model. Contemplate for instance the following XN-filter

$$\hat{y}(k) = f(\mathbf{x}; \mathbf{w}) = \exp(\mathbf{w}^\top \mathbf{x}(k)). \quad (2.13)$$

Taking the natural logarithm on each side of this equation yields:

$$\ln(\hat{y}(k)) = \mathbf{w}^\top \mathbf{x}(k). \quad (2.14)$$

That is, the logarithm of the output is obtained by a linear filtering of the input. The issue of suggesting a proper transformation of the input/output is a serious limitation of this approach. For further examples of transformation to linearity see [Seber & Wild 89, Ch. 1 & Ch. 2.8].

If none of the above listed paradigms seem applicable one may restore to an *ad hoc* analysis. Examples of *ad hoc* analyses are given below.

In the rest of this section a brief review of the classical methods for analyzing XN-filters is given. The review is mainly based on Bendat [Bendat 90] and Schetzen [Schetzen 89]. The main focus is on the methods which convey comprehension of the functionality of XN-filters.

The scope of Bendat [Bendat 90] is identification of nonlinear systems from random data. The XN-filters under consideration are simple zero-memory filters and general third order Volterra filters. The analysis and identification is based on a frequency-domain approach. In Schetzen [Schetzen 89] the Volterra and Wiener theories of nonlinear systems are presented. It is suggested that these theories may guide the analysis of “black box” models of XN-systems.

⁷Note that the dot product, $\boldsymbol{\alpha}^\top \mathbf{x}(k)$, corresponds to the convolution, $h(k) * x(k)$.

2.2.1 Nonlinear Filters based on Zero-Memory Nonlinear Filters

Consider the zero-memory XN-filter

$$\hat{y}(k) = s(x(k)) \quad (2.15)$$

where $s(\cdot)$ is a one-dimensional nonlinear function: $\mathbb{R} \mapsto \mathbb{R}$. Candidates of $s(\cdot)$ are given in [Bendat 90, Ch. 2.2.2]. For instance, $s(\cdot)$ could be a limiter reflecting a saturation effect which typically exists in a physical system. Two typical limiters are the hyperbolic tangent (smooth limiter), $\tanh(\cdot)$ and the signum function (hard limiter)

$$\text{sgn}(x) = \begin{cases} 1 & , x > 0 \\ 0 & , x = 0 \\ -1 & , x < 0 \end{cases} . \quad (2.16)$$

which are shown in Fig. 2.2. The general form of a XN-filter, restricted so that the only

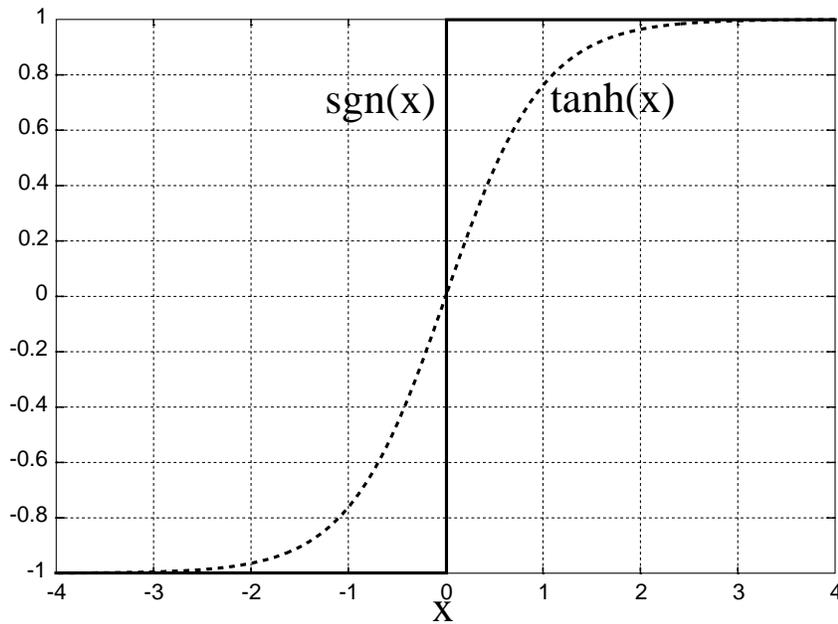


Figure 2.2: Limiting functions as examples of a zero-memory XN-filter. The solid line is the signum function, $\text{sgn}(x)$ and the dashed line is the hyperbolic tangent, $\tanh(x)$.

nonlinearity inherent is a zero-memory XN-filter block, is given by the equation:

$$\hat{y}(k) = h_{\text{po}}(k) * s(h_{\text{pr}}(k) * x(k)) \quad (2.17)$$

where $*$ denotes convolution. $h_{\text{pr}}(k)$, $h_{\text{po}}(k)$ are impulse responses of linear pre- and post-filters, respectively. If no pre-filter is present the filter is known as a Hammerstein filter. In Fig. 2.3 the XN-filter is shown. In this context the following definitions were made: $z(k) \triangleq h_{\text{pr}} * x(k)$, $v(k) \triangleq s(z(k))$. Two facts should be noticed:

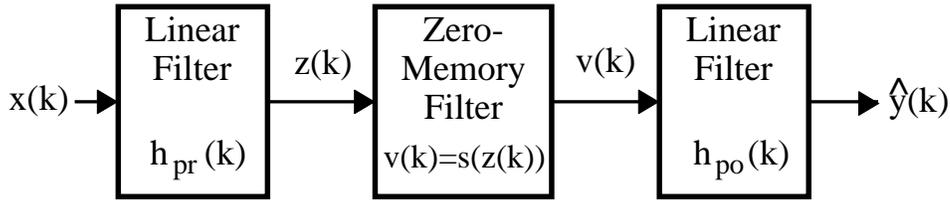


Figure 2.3: The XN-filter based on a zero-memory XN-filter block.

- It is not possible to implement an arbitrary XN-filter with this configuration. A simple example is the filter,

$$\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}) = f_1(x(k)) + f_2(x(k-1)) \quad (2.18)$$

where $L = 2$ and $f_1(\cdot)$, $f_2(\cdot)$ are arbitrary functions⁸ which is not realizable.

- If the linear filters, $h_{pr}(k)$ and $h_{po}(k)$, are stable recursive filters the stability of the entire XN-filter is ensured. The reason is that no *nonlinear* feedback is present. Furthermore, the XN-filter can not display strange behavior (chaos, limit cycles etc.) which is the case when dealing with a general recursive XN-filter as mentioned in Ch. 1.

A frequency-domain interpretation of this filter configuration is often possible due to the simplicity of the nonlinearity. If the input signal is deterministic the aim is to evaluate the Fourier spectrum of the filter output provided that the input spectrum and the details (i.e., h_{pr} , h_{po} and $s(\cdot)$) of the filter is known. The spectrum of the output, $\hat{Y}(f)$, is defined as (e.g., [Oppenheim & Schaffer 75])

$$\hat{Y}(f) = \text{Fou} \{ \hat{y}(k) \} = \sum_{k=-\infty}^{\infty} \hat{y}(k) e^{-j2\pi f_n k} \quad (2.19)$$

where $f_n \in [-1/2; 1/2]$ is the normalized frequency f/f_s , f_s denotes the sampling frequency, and $\text{Fou}\{\cdot\}$ is the Fourier transform operator. Throughout the rest of this Thesis we will omit the subindex n for simplicity, thus consider f as the normalized frequency. When the input signal is stochastic instead the aim is to evaluate the power spectrum of the output, $P_{\hat{y}}(f)$, which is given by

$$P_{\hat{y}}(f) = \sum_{\tau=-\infty}^{\infty} \phi_{\hat{y}}(\tau) e^{-j2\pi f \tau} \quad (2.20)$$

⁸It is assumed that the functions are neither zero nor proportional.

where $\phi_{\hat{y}}(k)$ is the autocorrelation function⁹

$$\phi_{\hat{y}}(\tau) = E\{\hat{y}(k)\hat{y}(k + \tau)\}, \quad \tau = 0, \pm 1, \pm 2, \dots \quad (2.21)$$

Using the results of linear spectral analysis (see e.g., [Oppenheim & Schaffer 75]) the frequency-domain representations of the XN-filter in `fa:zmfilt` yields in the deterministic case:

$$\hat{Y}(f) = H_{po}(f) \cdot s_1(H_{pr}(f)X(f)), \quad (2.22)$$

and in the stochastic case:

$$P_{\hat{y}}(f) = |H_{po}(f)|^2 \cdot s_2(|H_{pr}|^2 P_x(f)) \quad (2.23)$$

where $H_{pr}(f) = \text{Fou}\{h_{pr}(k)\}$, $H_{po}(f) = \text{Fou}\{h_{po}(k)\}$ and $s_1(\cdot)$, $s_2(\cdot)$ denote certain nonlinear functions which depend on the chosen zero-memory nonlinear function $s(\cdot)$. The frequency-domain representation is shown in Fig. 2.4. The crucial part in the frequency-

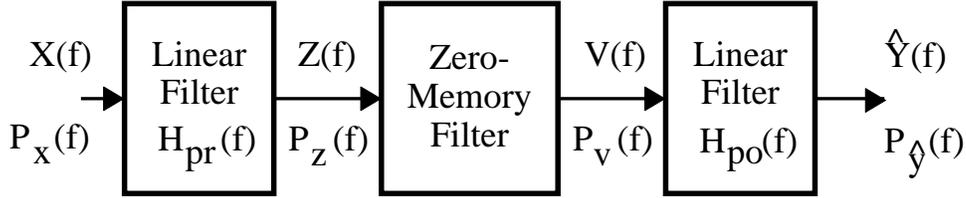


Figure 2.4: Frequency-domain representations of the XN-filter based on a zero-memory XN-filter block.

domain interpretation is the feasibility of calculating the functions $s_1(\cdot)$ and $s_2(\cdot)$. Here we merely treat two special examples of calculating $s_2(\cdot)$, or more precisely, the relationship between the power spectra $P_z(f)$ and $P_v(f)$ (see Fig. 2.4).

The examples are based on the following theorem [Haddad 75, pp. 59–63], [Bendat 90, Ch. 2.4]:

THEOREM 2.2 (PRICE) *Let $z(k)$ be a sum of a number of independent stationary stochastic processes, i.e.,*

$$z(k) = \sum_{j=1}^p z_j(k), \quad (2.24)$$

and $z_1(k)$ a Gaussian process with autocovariance function,

$$\gamma_{z_1}(\tau) = E\{(z_1(k) - E\{z_1(k)\})(z_1(k + \tau) - E\{z_1(k)\})\}. \quad (2.25)$$

⁹Here $E\{\cdot\}$ denotes expectation. Sometimes we use the notation: $E_x\{\cdot\}$ in order to emphasize that the expectation is w.r.t. the stochastic variable x .

Further let $v(k) = s(z(k))$ where $s(\cdot)$ is a zero-memory, differentiable nonlinear function. The autocorrelation function of $v(k)$, $\phi_v(\tau) = E\{v(k)v(k+\tau)\}$ then complies with the following differential equation:

$$\frac{\partial \phi_v(\tau)}{\partial \gamma_{z_1}(\tau)} = E\{s'(z(k))s'(z(k+\tau))\}. \quad (2.26)$$

Here $s'(z)$ is the derivative, $ds(z)/dz$.

Assume for simplicity that $x(k)$ is a zero-mean Gaussian process¹⁰ and the filter $h_{\text{pr}}(k)$ is designed so that the variance of $z(k)$ equals one, i.e., $z(k) \in \mathcal{N}(0, 1)$. If $s(\cdot)$ implements the signum function then the above theorem gives (see [Haddad 75, p. 61]):

$$\phi_v(\tau) = \frac{2}{\pi} \arcsin(\phi_z(\tau)). \quad (2.27)$$

When applying the smooth limiter¹¹

$$s(z) = \frac{1}{\sqrt{2\pi}\sigma} \int_0^z \exp\left(-\frac{t^2}{2\sigma^2}\right) dt. \quad (2.28)$$

a similar result can be obtained [Haddad 75, p. 62]

$$\phi_v(\tau) = \frac{1}{2\pi} \arcsin\left(\frac{\phi_z(\tau)}{1 + \sigma^2}\right). \quad (2.29)$$

According to fa:stofreq the power spectrum of the output in smooth limiting case is calculated via the steps:

$$P_z(f) = |H_{\text{pr}}(f)|^2 P_x(f), \quad (2.30)$$

$$\phi_v(\tau) = \frac{1}{2\pi} \arcsin\left(\frac{\phi_z(\tau)}{1 + \sigma^2}\right), \quad (2.31)$$

$$P_v(f) = \text{Fou}\{\phi_v(\tau)\}, \quad (2.32)$$

$$P_{\hat{y}}(f) = |H_{\text{po}}(f)|^2 P_v(f). \quad (2.33)$$

2.2.2 Volterra Filters and Frequency-Domain Interpretations

The Volterra filter is an attempt to approximate the nonlinear system, which has to be modeled, by a multidimensional Taylor series expansion. The l 'th order Volterra filter is given by:

$$\begin{aligned} \hat{y}(k) = & h_0 + \sum_{n_1=-\infty}^{\infty} h_1(n_1)x(k-n_1) \\ & + \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} h_2(n_1, n_2)x(k-n_1)x(k-n_2) + \dots \\ & + \sum_{n_1=-\infty}^{\infty} \dots \sum_{n_l=-\infty}^{\infty} h_l(n_1, n_2, \dots, n_l)x(k-n_1)x(k-n_2)\dots x(k-n_l) \end{aligned} \quad (2.34)$$

¹⁰Note that $z(k)$ is Gaussian whenever $x(k)$ is Gaussian.

¹¹Note that this limiting function is equal to the distribution function of a $\mathcal{N}(0, \sigma^2)$ stochastic variable.

where $h_r(n_1, n_2, \dots, n_r)$ is the r 'th order (time-domain) kernel which, without loss of generality, can be assumed to be symmetric w.r.t. any permutation of the indices n_1, n_2, \dots, n_r . The Volterra filter is completely characterized by these kernels. Normally only filters of relatively low order are under consideration due to fact that the involved computations increase tremendously with increasing order (see further below).

$\hat{y}_r(k)$ represents the output of a r -linear (linear, bilinear, trilinear, etc.) filter and is given by the r 'th Volterra operator in `fa:volfilt`, i.e.,

$$\hat{y}_r(k) = \sum_{n_1=-\infty}^{\infty} \cdots \sum_{n_r=-\infty}^{\infty} h_r(n_1, n_2, \dots, n_r) x(k - n_1) x(k - n_2) \cdots x(k - n_r). \quad (2.35)$$

Notice that this equation represents an r 'th order convolution.

An example of an r -linear filter is the simple bilinear filter:

$$\hat{y}_2(k) = (x(k) * h(k))^2. \quad (2.36)$$

According to the last subsection this bilinear filter may be viewed as a general filter based on a zero-memory filter block with pre-filter, $h(k)$, no post-filter (i.e., $h_{po}(k) = \delta(k)$), and zero-memory nonlinearity, $s(z) = z^2$. In [Bendat 90, pp. 94–95] it is shown that the second-order kernel of this filter is given by:

$$h_2(n_1, n_2) = h(n_1)h(n_2). \quad (2.37)$$

According to the definition of the r -linear filter, the output of the Volterra filter can be expressed as:

$$\hat{y}(k) = y_0 + \sum_{r=1}^l \hat{y}_r(k) \quad (2.38)$$

where $y_0 \triangleq h_0$. In Fig. 2.5 the general Volterra filter is depicted. The l 'th order Volterra filter is causal if all r -linear filters, $i = 1, 2, \dots, l$ are causal which is ensured if and only if $\forall r \in [1; l]$:

$$h_r(n_1, n_2, \dots, n_r) = 0, \quad \text{for } n_i < 0, \quad i \in [1; r]. \quad (2.39)$$

Furthermore the BIBO-stability of the filter is ensured if $\forall r \in [1; l]$:

$$\sum_{n_1=-\infty}^{\infty} \cdots \sum_{n_r=-\infty}^{\infty} |h_r(n_1, n_2, \dots, n_r)| < \infty. \quad (2.40)$$

An advantage of using the Volterra filter representation is the possibility of a frequency interpretation via higher order spectra. The r 'th order Fourier transform of the kernel $h_r(n_1, n_2, \dots, n_r)$ is defined as:

$$\begin{aligned} H_r(f_1, f_2, \dots, f_r) &= \text{Fou}\{h_r(n_1, n_2, \dots, n_r)\} \\ &= \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} \cdots \sum_{n_r=-\infty}^{\infty} h_r(n_1, n_2, \dots, n_r) e^{-j2\pi(f_1 n_1 + f_2 n_2 + \dots + f_r n_r)} \end{aligned} \quad (2.41)$$

where f_i , $i = 1, 2, \dots, r$ are normalized frequencies (w.r.t. the sampling frequency, f_s). $H_r(f_1, f_2, \dots, f_r)$ is often called the r -linear frequency response function or the r 'th order frequency-domain kernel.

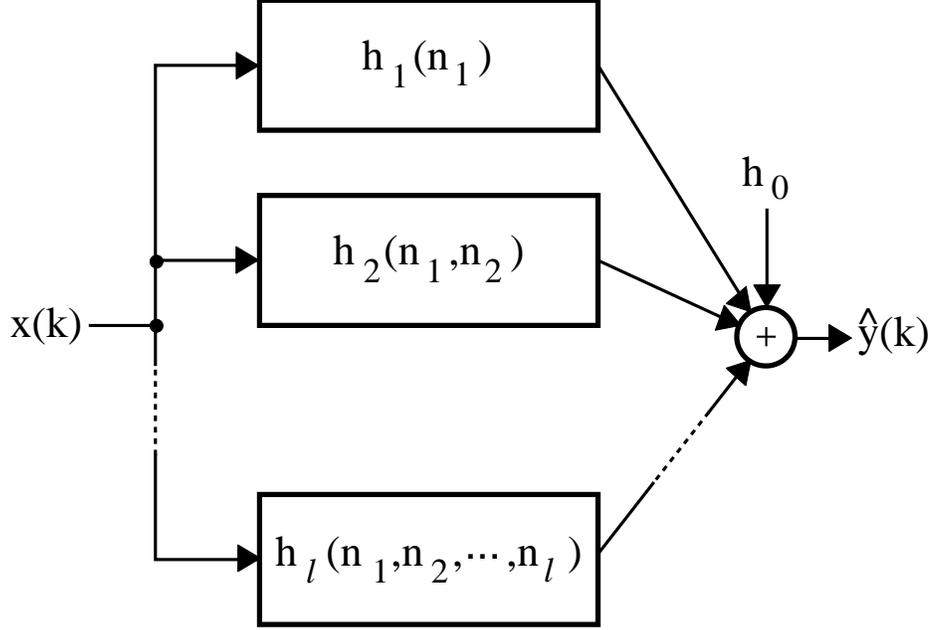


Figure 2.5: The general l 'th order Volterra filter specified by the kernels: h_0 , $h_r(n_1, n_2, \dots, n_r)$, $r = 1, 2, \dots, l$.

When $x(k)$ is a deterministic process the spectrum of the output of the r -linear filter, $\hat{Y}_r(f)$ is given as [Schetzen 89, Ch. 6.3]:

$$\hat{Y}_r(f) = \int_{-1/2}^{1/2} \cdots \int_{-1/2}^{1/2} H_r(f - \alpha_1, \alpha_1 - \alpha_2, \dots, \alpha_{r-2} - \alpha_{r-1}, \alpha_{r-1}) X(f) X(f - \alpha_1) \cdots X(\alpha_{r-1}) d\alpha_1 d\alpha_2 \cdots d\alpha_{r-1}. \quad (2.42)$$

Note that this equation reduces to the usual relation, $\hat{Y}_1(f) = H_1(f)X(f)$, for linear filters (i.e., $r = 1$). The output spectrum of the Volterra filter fa:volfilt then yields:

$$\hat{Y}(f) = h_0\delta(f) + \sum_{r=1}^l \hat{Y}_r(f) \quad (2.43)$$

where $\delta(f)$ is the Dirac delta function.

When $x(k)$ is a stochastic process the relevant feature is the power spectrum of the output. The power spectrum is – due to the uniqueness of the Fourier transform – equivalent with the autocorrelation function of the output which formally is given by:

$$\begin{aligned} \phi_{\hat{y}}(\tau) &= E \{ \hat{y}(k) \hat{y}(k + \tau) \} \\ &= E \left\{ \left(\sum_{r=0}^l \hat{y}_r(k) \right) \left(\sum_{r=0}^l \hat{y}_r(k + \tau) \right) \right\} \\ &= \sum_{r_1=0}^l \sum_{r_2=0}^l E \{ \hat{y}_{r_1}(k) \hat{y}_{r_2}(k) \}. \end{aligned} \quad (2.44)$$

In general it is difficult to calculate this autocorrelation. This difficulty arises from the following observations:

- The output of r_1 -linear and the r_2 -linear filter is usually correlated. Consequently, fa:autocor contains up to $(l + 1)^2$ terms.
- A non-zero mean value of the input induces a lot of extra terms which have to be calculated. Consider for instance the simple example bilinear filter: $\hat{y}_2(k) = x^2(k)$. Let the mean value $E\{x(k)\} \triangleq \bar{x}$ and define the zero-mean variable $x_z(k) \triangleq x(k) - \bar{x}$. Now, the autocorrelation function, $\phi_{\hat{y}_2}(k)$, of $\hat{y}_2(k)$ becomes (certain algebraic manipulations are omitted):

$$\begin{aligned} \phi_{\hat{y}_2}(\tau) &= E \left\{ x^2(k)x^2(k + \tau) \right\} \\ &= E \left\{ x_z^2(k)x_z^2(k + \tau) \right\} + 2\bar{x}E \left\{ x_z(k)x_z^2(k + \tau) \right\} + 2\bar{x}E \left\{ x_z^2(k)x_z(k + \tau) \right\} \\ &\quad + 2\bar{x}^2E \left\{ x_z^2(k) \right\} + 4\bar{x}^2E \left\{ x_z(k)x_z(k + \tau) \right\} + \bar{x}^4. \end{aligned} \quad (2.45)$$

However, without loss of generality, one may assume that the mean value of the input is zero¹². A possible mean value at the output, $\hat{y}(k)$, is compensated by adjusting h_0 properly.

- Input correlations of form

$$E \left\{ \prod_{i=1}^r x(k - \tau_i) \right\}, \quad r = 1, 2, \dots, l \quad (2.46)$$

are involved in the calculation of the autocorrelation function of the output according to fa:autocor. If no assumptions concerning the distribution of the input are available one has to estimate these correlations. This may be very time consuming and requires a large amount of data in order to ensure reliable estimates. However, for a Gaussian distributed variable it is known that all higher order moments are expressed as a function of the first and second moments only (see e.g., [Bendat & Piersol 86, Ch. 3]). Consequently, assuming that the input is Gaussian distributed implies a reduction in the number of necessary quantities which have to be estimated.

In [Bendat 90, Ch. 4] the power spectrum of a general 3rd-order Volterra filter with zero-mean Gaussian input is calculated. The details of the calculation is due to the extend not recapitulated. In order to alleviate the issue of correlation among the outputs of the r -linear filters – as mentioned in the first item above – the r -linear filters are modified so that the outputs become uncorrelated and obtain zero means¹³. The result is:

$$P_{\hat{y}}(f) = (h_0^2 + \bar{y}_2^2)\delta(f) + P_{\hat{y}_a}(f) + P_{\hat{y}_b}(f) + P_{\hat{y}_c}(f) \quad (2.47)$$

where \bar{y}_2 is the mean value of the bilinear filter, $\hat{y}_a(k)$, $\hat{y}_b(k)$, and $\hat{y}_c(k)$ are the outputs of the modified linear, bilinear, and trilinear filters, respectively. Further,

$$\bar{y}_2 = \int_{-1/2}^{1/2} H_2(f, -f)P_x(f) df, \quad (2.48)$$

¹²If the input contains a mean value a zero-mean input variable is constructed simply by subtracting an estimate of the mean value.

¹³Note that the output of an r -linear filter where r is even has a non-zero mean value even though the mean value of the input is zero.

$$C(f) = 3 \int_{-1/2}^{1/2} H_3(\alpha, -\alpha, f) P_x(f) d\alpha, \quad (2.49)$$

$$P_{y_a}^{\wedge}(f) = |H_1(f) + C(f)|^2 P_x(f), \quad (2.50)$$

$$P_{y_b}^{\wedge}(f) = 2 \int_{-1/2}^{1/2} |H_2(f - \alpha, \alpha)|^2 P_x(f - \alpha) P_x(\alpha), \quad (2.51)$$

$$P_{y_c}^{\wedge}(f) = 6 \int_{-1/2}^{1/2} \int_{-1/2}^{1/2} |H_3(f - \alpha_1, \alpha_1 - \alpha_2, \alpha_2)|^2 P_x(f - \alpha_1) \cdot P_x(\alpha_1 - \alpha_2) P_x(\alpha_2) d\alpha_1 d\alpha_2. \quad (2.52)$$

2.2.3 Concluding Remarks on Nonlinear Filter Analysis

The presented paradigms (see pp. 18–19) for XN-filter analysis may provide as a catalogue of alternatives when facing the analysis of a specific filter architecture.

The mentioned frequency-domain methods for analyzing XN-filters may provide a useful tool in some applications; however, we emphasize some restrictions:

- The analysis of filter based zero-memory filter blocks is highly dependent on the shape of the zero-memory function $s(\cdot)$ and the assumptions concerning the distribution of the input.
- The frequency interpretation of a Volterra filter becomes difficult when the order is greater than three. Furthermore, the analysis may be very difficult when the Gaussian input assumption is circumvented.
- The fact that the Volterra frequency kernels in general depend on more than two frequency parameters obstructs a visual survey compared to the classical frequency response function.
- The spectrum or power spectrum of the output of a r -linear filter is connected with the frequency kernels and the input power spectrum in a very complex way (see e.g., fa:outspec). Consequently, the effect of changing a specific kernel is not easy to clarify.

2.3 Summary

First a number of basic properties of XN-filters was given. This includes: Superposition, time-invariance, stability and causality. Next three fundamental paradigms for nonlinear filter analysis were presented. The first paradigm consists of expanding the nonlinear filtering function in a set of simple basis functions. The second paradigm deals with successive linearizations of the filtering function around samples in the input space. This results in a time-varying linear filter. Finally, the third paradigm treats the possibility of mapping the input and output variables into new variables in which the filter becomes more simple.

The rest of the chapter was devoted to a review of the classical filter analysis. This covers the XN-filters based on zero-memory filter blocks and the Volterra filters.

CHAPTER 3

NONLINEAR FILTER ARCHITECTURES

In this chapter a novel taxonomy of XN-filter architectures based on general properties of the filter architecture is presented. The purpose is to convey a unified view of the numerous architectures proposed in the literature. This is explicitly done by formulating a *canonical filter representation*. In the literature contributions on comparing nonlinear filter architectures have appeared, e.g., [Cherkassky 92], [Farmer & Sidorowich 88], [Friedman 91]; however, an obvious benefit of the proposed taxonomy is that it deals with the properties of the filter architecture only. Hence, the issues concerning algorithms for estimating filter parameters, and algorithms for selecting a proper architecture for a specific task, have no influence on the discussion. On the other hand, the comprehension of the properties of a specific architecture may guide the choice of these algorithms. It should be emphasized that the proposed taxonomy is based on a few architectural properties only. Further, it is developed in preparation for implementing the SISO, non-recursive XN-model with additive error. That is, future research may provide several refinements of the taxonomy.

The presented taxonomy permits the classification of a variety of existing filter architectures, and furthermore, it opens up the prospect of suggesting a variety of novel architectures for future research. We mainly concentrate on architectures which possess a neural network like structure.

Based on the taxonomy a number of existing neural network like filter architectures are expounded, interpreted and compared aiming at implementing the XN-model. In particular, we attach importance to the multi-layer feed-forward perceptron neural network (MFPNN).

3.1 Taxonomy of Nonlinear Filter Architectures

In this section we present a taxonomy of XN-filter architectures based on general architectural properties.

Recapitulate the nonrecursive XN-model with additive error (cf. nf:model):

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k), \quad (3.1)$$

and the corresponding XN-filter

$$\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}), \quad (3.2)$$

aiming at modeling the relationship between the input signal¹, $x(k)$, and the output signal, $y(k)$. $f(\cdot)$ is the filtering function which represents a specific member of the filter architecture \mathcal{F} and may be parameterized by the m -dimensional weight (parameter) vector \mathbf{w} . However, some filter architectures – the so-called nonparametric architectures – do not contain any parameters at all. In most cases there is a need for preprocessing the input signal which is further treated in Ch. 4. In consequence the input vector signal, $\mathbf{x}(k)$, is mapped into a preprocessed vector $\mathbf{z}(k) = [z_1(k), z_2(k), \dots, z_p(k)]^\top$ which – in contrast to $\mathbf{x}(k)$ – does not contain a time shift structure. In the rest of this chapter we will replace $\mathbf{x}(k)$ by the corresponding preprocessed vector $\mathbf{z}(k)$. In consequence, the filtering function, $f(\cdot)$, in `nf:nonfilt` is decomposed into a preprocessing vector function, $\mathbf{f}_p(\cdot)$, and a nonlinear filtering function, $f_n(\cdot)$. According to `nf:nonfilt` we get:

$$\mathbf{z}(k) = \mathbf{f}_p(\mathbf{x}(k)) \quad (3.3)$$

and

$$\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}) \triangleq f_n(\mathbf{z}(k); \mathbf{w}). \quad (3.4)$$

The explicit knowledge of the input-output relationship is obtained by collecting a set, \mathcal{T} , of connected input-output samples. This set is denoted the *training set*:

$$\mathcal{T} = \{\mathbf{x}(k); y(k)\}, \quad k = 1, 2, \dots, N \quad (3.5)$$

where N is the size of the training set.

Provided that a filter architecture has been chosen the task is to exploit the knowledge contained in the training set so that future samples of the predicted output, $\hat{y}(k)$, $k = N + 1, N + 2, \dots$, become as close as possible to the output, $y(k)$, w.r.t. some metric, e.g., in the least squares (LS) sense, see further Ch. 5, 6. That is, the filter is adjusted so that some *cost function* based on the training data is minimized.

In Fig. 3.1 an example of a nonlinear relationship², $y(k) = g_n(\mathbf{z}(k))$, is depicted in order to accentuate a geometrical interpretation. The task of the XN-filter is then to approximate the $p + 1$ -dimensional input-output surface spanned by all points satisfying the nonlinear relationship $g(\cdot)$. In particular the training points

$$[z_1(k), z_2(k), \dots, z_p(k), y(k)], \quad k = 1, 2, \dots, N$$

lie on this surface.

3.1.1 Parametric and Nonparametric Architectures

The first property relevant to the taxonomy is the distinction between *parametric* and *nonparametric* architectures.

DEFINITION 3.1 *A filter architecture is said to be nonparametric if the prediction, $\hat{y}(k)$, is determined from $\mathbf{z}(k)$ and the entire training set, \mathcal{T} , solely. That is,*

$$\hat{y}(k) = f_n(\mathbf{z}(k); \mathcal{T}). \quad (3.6)$$

Otherwise, the architecture is parametric and the prediction is given by

$$\hat{y}(k) = f_n(\mathbf{z}(k); \hat{\mathbf{w}}) \quad (3.7)$$

where $\hat{\mathbf{w}}$ are parameters estimated on the training set w.r.t. some performance criterion or cost function (see Ch. 5).

¹Recall that the input vector signal is defined by: $\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-L+1)]^\top$.

²A decomposition of the nonlinear system, $g(\cdot)$, similar to that of $f(\cdot)$ is provided in this example.

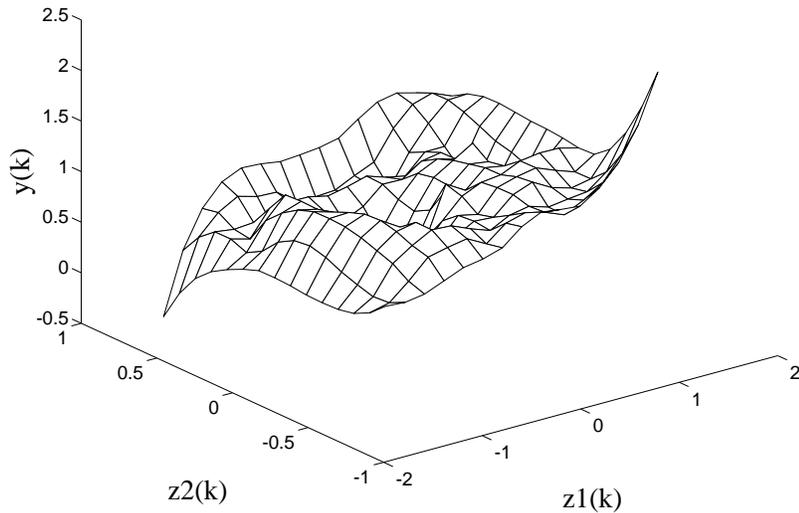


Figure 3.1: An example of a nonlinear relationship $y(k) = g_n(\mathbf{z}(k))$. The example originates from predicting the chaotic Mackey-Glass time series which is explained in detail in Ch. 8. The dimension of $\mathbf{z}(k)$ is in fact equal to 4; however, only the first two components are considered in this figure. Clearly, the relationship is nonlinear; otherwise, the surface would have been a plane.

When using a nonparametric architecture one has to store the entire training set whereas only the estimated parameters, $\hat{\mathbf{w}}$, have to be stored when dealing with a parametric architecture. The estimated parameters may thus be viewed as an encoding of the training set. The geometrical point of view given above may guide a possible interpretation of the nonparametric architecture. First, notice that the training set – regarded as a set of points in the $p + 1$ -dimensional input-output space – obviously lies on the input-output surface. The predicted output of a novel sample of $\mathbf{z}(k)$ can now be interpreted as an inter- or extrapolation of the training points performed by the function $f_n(\cdot)$. On the other hand, when dealing with a parametric architecture the input-output surface is considered as a family of surfaces which arise by varying the parameter vector, \mathbf{w} . The training thus results in selecting one of these surfaces.

3.1.2 Local and Global Approximation

The second property of the architecture deals with the differentiation between *local* and *global* approximation.

DEFINITION 3.2 Let $\mathbf{z}(k) \in \mathcal{D}$ where $\mathcal{D} \subseteq \mathbb{R}^p$ is the set (input domain) in which the nonlinear relationship under consideration has to be modeled. Within a local approximation

approach \mathcal{D} is divided into q subsets, \mathcal{D}_i , $i = 1, 2, \dots, q$, which cover \mathcal{D} but may have non-empty intersections. In each subset a local filter is formulated, and the final output results by combining the outputs of the local filters.

A global approximation approach is characterized by the fact that \mathcal{D} is not divided at all.

3.1.2.1 Canonical Filter Representation

In order to detail various global/local approximation approaches (when implementing the parametric XN-filter `nf:nonfilt`) it is possible – without loss of generality – to rewrite `nf:nonfilt` to comply with the following *canonical filter representation*³ which is shown in Fig. 3.2:

$$\begin{aligned}
 \hat{y}(k) &= f_n(\mathbf{z}(k); \mathbf{w}) \\
 &= \alpha_0 + \sum_{i=1}^q \alpha_i \cdot b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) \cdot D_i(\mathbf{z}(k)) \\
 &= \alpha_0 + \sum_{i=1}^q \alpha_i v_i(k) \\
 &= \boldsymbol{\alpha}^\top \mathbf{v}(k)
 \end{aligned} \tag{3.8}$$

where

- $b_i(\mathbf{z}(k); \boldsymbol{\beta}_i)$ is the i 'th *basis function* parameterized by $\boldsymbol{\beta}_i$. The basis functions define a set of functions in which $f_n(\cdot)$ is expanded. Note that a basis function equal to unity indirectly is introduced by the parameter α_0 . Normally $\alpha_0 \neq 0$ in order to control the mean value of the filter output.
- α_i is the weight with which the i 'th basis function contributes to the generation of the filter output⁴. The weights are assembled in the $q + 1$ -dimensional vector: $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \dots, \alpha_q]^\top$.
- $D_i(\mathbf{z}(k))$ is a *domain function* which specifies the input domain related to the i 'th basis function. We assume $\forall \mathbf{z}(k), \forall i$:

$$0 \leq D_i(\mathbf{z}(k)) \leq 1. \tag{3.9}$$

- The signals, $v_i(k)$, $i = 1, 2, \dots, q$, are defined as:

$$v_i(k) = b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) \cdot D_i(\mathbf{z}(k)), \tag{3.10}$$

and ⁵

$$\mathbf{v}(k) = [1, v_1(k), \dots, v_q(k)]^\top. \tag{3.11}$$

Note – according to Sec. 3.2.2 – that the canonical filter representation can be regarded as a two-layer feed-forward neural network consisting of q hidden nonlinear neurons. The i 'th neuron possesses the processing: $b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) \cdot D_i(\mathbf{z}(k))$.

In the next paragraphs a number of specialized forms of $D_i(\cdot)$ will be treated.

³Note that the preprocessing is excluded.

⁴Strictly speaking this is only true when $D_i(\mathbf{z}) = 1$.

⁵Later on we shall denote a vector like \mathbf{v} an *augmented* vector as it contains a one and thereby an augmentation of the vector: $\tilde{\mathbf{v}} = [v_1, v_2, \dots, v_q]^\top$.

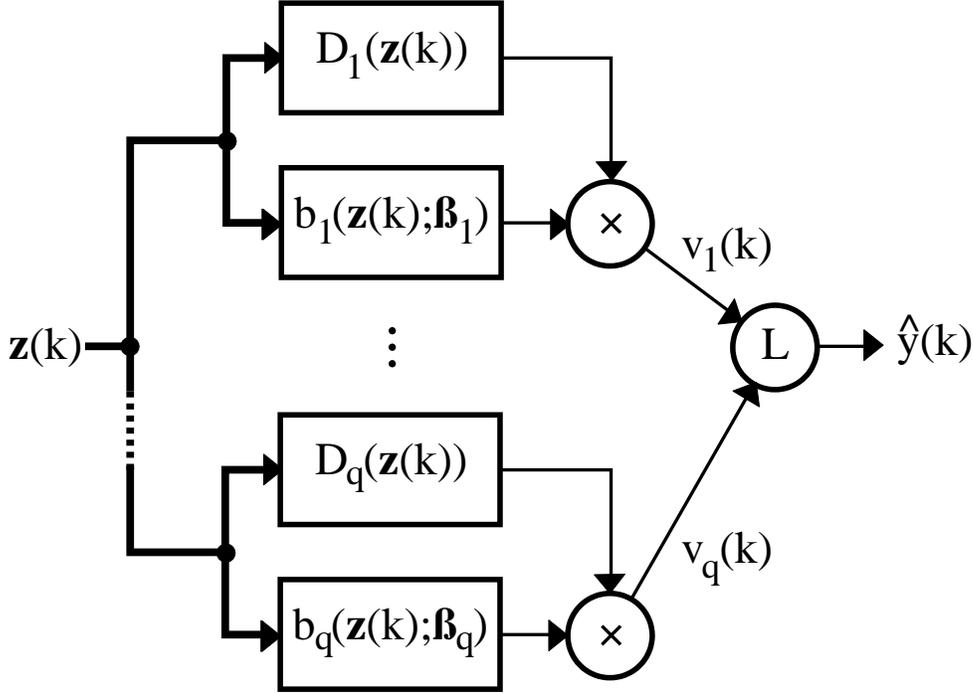


Figure 3.2: The canonical filter representation. $b_i(\cdot)$ are the basis functions, $D_i(\cdot)$ are the domain functions, and α_i are the weights of the linear neuron. The bold lines represent p connections.

Global Approximation Filters Dealing with a global approximation approach then $\forall z(k), \forall i \in [1; q]$:

$$D_i(z(k)) \equiv 1. \quad (3.12)$$

That is, all basis functions contribute over the entire input domain, \mathcal{D} . Consequently, the domain function plays no significant role. As a result, the canonical representation of nfnfcan becomes:

$$\hat{y}(k) = \alpha_0 + \sum_{i=1}^q \alpha_i \cdot b_i(z(k); \beta_i). \quad (3.13)$$

Localized Covering Functions Consider a local approximation approach where the subsets, \mathcal{D}_i , cover the input domain, \mathcal{D} , but possess non-empty intersections. $D_i(\cdot)$ is in this context denoted a *localized covering function* with the following properties:

1. Assume the existence of a compact set of input vectors, $\mathcal{P}_i \subseteq \mathcal{D}_i$, so that

$$D_i(z_i^o) = 1, \quad \forall z_i^o \in \mathcal{P}_i. \quad (3.14)$$

That is, D_i is equal to unity⁶ in the compact set, \mathcal{P}_i , within \mathcal{D}_i . It should be noted that if $\mathcal{P}_i \equiv \mathcal{D}_i$ then we assume that $\mathcal{D}_i \subset \mathcal{D}$.

⁶Note that setting the domain functions equal to unity at the points, z^o , is not significant; on the contrary just a convenient choice.

2. Let $\|\cdot\|$ denote a vector norm. If $\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{D}_i \setminus \mathcal{P}_i$, $\mathcal{D}_i \setminus \mathcal{P}_i \neq \emptyset$, and

$$\|\mathbf{z}_1 - \mathbf{z}_i^o\| < \|\mathbf{z}_2 - \mathbf{z}_i^o\|, \quad \mathbf{z}_i^o \in \mathcal{P}_i, \quad (3.15)$$

then it is required that:

$$D_i(\mathbf{z}_2) < D_i(\mathbf{z}_1). \quad (3.16)$$

This requirement ensures that $D_i(\cdot)$ decreases exterior to \mathcal{P}_i .

In Fig. 3.3 is shown an example of Gaussian bell-shaped localized covering functions. That is,

$$D_i(\mathbf{z}(k)) = \exp\left(-\frac{|\mathbf{z}(k) - \mathbf{z}_i^o|^2}{\sigma_i^2}\right) \quad (3.17)$$

where σ_i^2 is a scaling parameter which defines the effective width of the bell.

Partition Functions Let the subsets, \mathcal{D}_i , be disjoint, i.e., $\mathcal{D}_{i_1} \cap \mathcal{D}_{i_2} = \emptyset$, $\forall i_1, i_2$. Consequently, $\mathcal{D} = \bigcup_{i=1}^q \mathcal{D}_i$. In this case the subsets \mathcal{D}_i are denoted *partitions* and the domain functions, $D_i(\mathbf{z}(k))$, are denoted *partition functions*. It is assumed that:

$$D_i(\mathbf{z}(k)) = \begin{cases} 1 & , \mathbf{z}(k) \in \mathcal{D}_i \\ 0 & , \text{otherwise} \end{cases} . \quad (3.18)$$

The shape of the partitions \mathcal{D}_i can, in principle, be of any kind; however, normally we prefer the different regions to be separated by a number of hyperplanes. A specific hyperplane, \mathcal{H} , is given by:

$$\mathcal{H} : \mathbf{a}^\top \mathbf{z}(k) - t = 0 \quad (3.19)$$

where \mathbf{a} defines the p -dimensional normal vector of the hyperplane and t is a threshold which specifies the offset of the hyperplane relative to the origin of the input domain, \mathcal{D} . \mathcal{H}^+ , \mathcal{H}^- are the positive and negative regions, respectively, produced by the hyperplane, \mathcal{H} , and obey the following definition:

$$\mathcal{H}^+ = \{ \mathbf{z}(k) : \mathbf{a}^\top \mathbf{z}(k) \geq t \}, \quad (3.20)$$

$$\mathcal{H}^- = \{ \mathbf{z}(k) : \mathbf{a}^\top \mathbf{z}(k) < t \}. \quad (3.21)$$

The issue of determining which subset a specific vector $\mathbf{z}(k)$ belongs to can be resolved using a binary tree structure, see e.g., [Friedman 91], [Gelfand et al. 91], [Hoffmann 92a]. In this context we define a *node*, \mathcal{O}_{rs} , as the tuple consisting of a hyperplane normal vector and a threshold, i.e.,

$$\mathcal{O}_{rs} = \{ \mathbf{a}_{rs}, t_{rs} \} \quad (3.22)$$

where $r = 0, 1, \dots$ is the depth and $s \in [0; 2^r - 1]$ specifies the node number at the actual depth. In Fig. 3.4 the binary tree and the corresponding partitions⁷, D_{rs} , specified by the hyperplanes, \mathcal{H}_{rs} , are shown.

In order to ascertain which domain a given input vector $\mathbf{z}(k)$ belongs to, the tree is traversed according to the following algorithm:

Tree Traversing Algorithm Start at node \mathcal{O}_{00} and initialize: $(r, s) = (0, 0)$. If $\mathbf{z}(k) \in \mathcal{H}_{rs}^-$, i.e., $\mathbf{a}_{rs}^\top \mathbf{z}(k) < t_{rs}$ then continue to node

⁷Note that the notation is slightly changed. However, there exists a one-to-one correspondence between D_i and D_{rs} for each $i = 1, 2, \dots, q$.

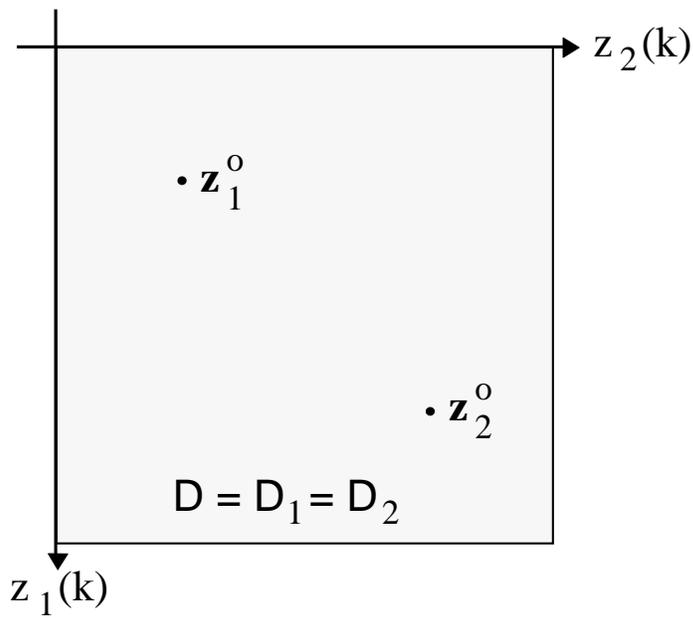
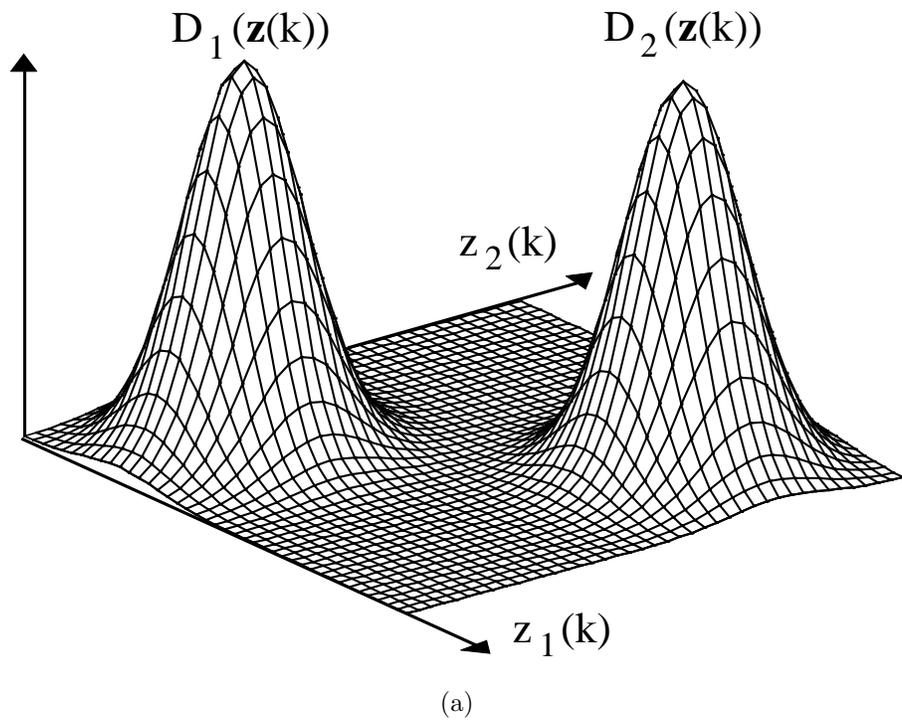


Figure 3.3: Example of Gaussian bell-shaped localizing covering functions cf. `nf:gausbell`. Note that the subsets $\mathcal{D}_1, \mathcal{D}_2$ in this example are equal to \mathcal{D} as the localizing functions have infinite support. In practice; however, the scaling parameter, σ_i^2 (cf. `nf:gausbell`), normally is chosen so that the overlap between different localizing functions is small.

$\mathcal{O}_{r+1,2s}$ and update $(r, s) \leftarrow (r + 1, 2s)$. Otherwise, continue to node $\mathcal{O}_{r+1,2s+1}$ and update $(r, s) \leftarrow (r + 1, 2s + 1)$. If the actual node is a terminal node then stop and realize that $\mathbf{z}(k) \in \mathcal{D}_{rs}$; otherwise, go to *Step 2*.

Note that the terminal nodes, \mathcal{O}_{10} , \mathcal{O}_{22} , and \mathcal{O}_{23} do not contain a normal vector or a threshold unless further extension of the tree is needed. Furthermore, it is observed that the tree is not necessarily complete, i.e., some nodes at a given depth may not be present.

Gate Functions The domain function, called the *gate function*, appears as a special case of the partition function, and the corresponding subsets \mathcal{D}_i are denoted *gates*. The restriction consists in the imposition that the hyperplanes are parallel to the axes, $z_j(k)$, $j = 1, 2, \dots, p$. Consequently, the normal vectors comply with:

$$a_j = \begin{cases} 1 & , j = j_{rs} \\ 0 & , \text{otherwise} \end{cases} \quad (3.23)$$

where j_{rs} defines the direction of the hyperplane, \mathcal{H}_{rs} .

The definition of a node, cf. `nf:node`, is changed to:

$$\mathcal{O}_{rs} = \{j_{rs}, t_{rs}\}. \quad (3.24)$$

Fig. 3.5 shows an example of a binary tree and the corresponding partitions when dealing with gate functions.

In the Venn-diagram Fig. 3.6 possible special cases of the domain function is summarized.

3.1.3 Orthogonal Architectures

The third property which characterizes a parametric filter architecture is about whether the $v_i(k)$ signals are *orthogonal* or not. A necessary prerequisite, regarding the formulation of orthogonality, is the definition of a suitable *inner product* for functions. Consider the two signals

$$v_{i_1}(k) = b_{i_1}(\mathbf{z}(k); \beta_{i_1}) \cdot D_{i_1}(\mathbf{z}(k)), \quad (3.25)$$

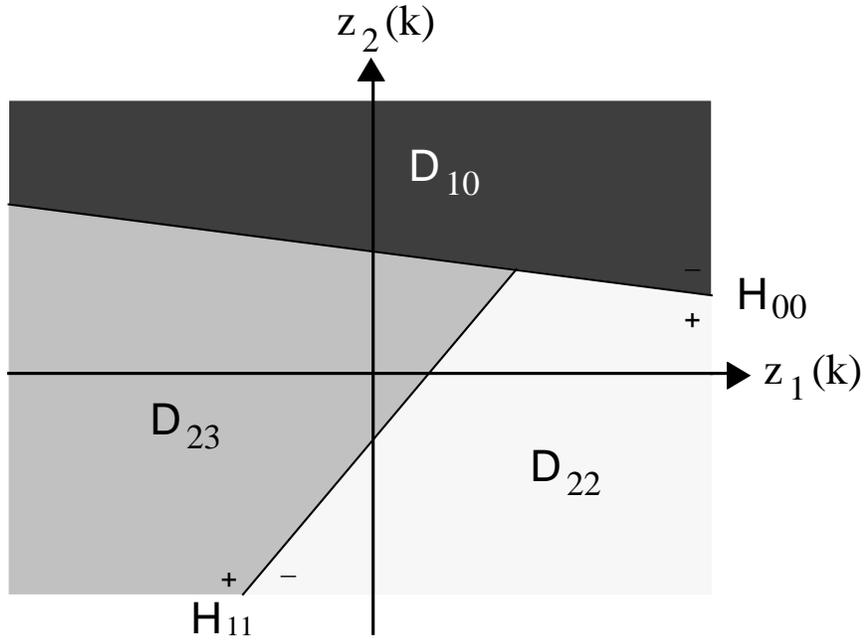
$$v_{i_2}(k) = b_{i_2}(\mathbf{z}(k); \beta_{i_2}) \cdot D_{i_2}(\mathbf{z}(k)) \quad (3.26)$$

and the associated inner product: $\langle v_{i_1}(k), v_{i_2}(k) \rangle$. If $\mathbf{z}(k)$ is a deterministic vector signal then one may chose the following inner product definition:

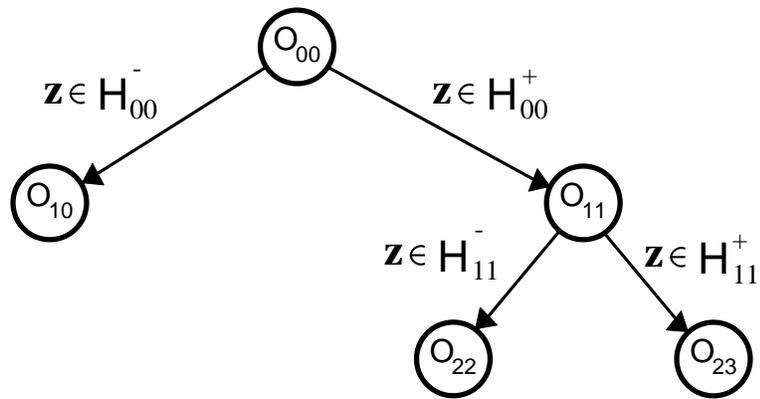
$$\langle v_{i_1}(k), v_{i_2}(k) \rangle = \frac{1}{T} \sum_{k=1}^T v_{i_1}(k)v_{i_2}(k) \quad (3.27)$$

where T is the length of the signal (which may be infinite). However, when dealing with a stochastic input signal a common choice is:

$$\begin{aligned} \langle v_{i_1}(k), v_{i_2}(k) \rangle &= E_{\mathbf{z}(k)} \{v_{i_1}(k)v_{i_2}(k)\} \\ &= \int_{\mathcal{D}} v_{i_1}(k)v_{i_2}(k)p(\mathbf{z}(k)) d\mathbf{z}(k) \end{aligned} \quad (3.28)$$



(a)



(b)

Step 3

Figure 3.4: Example of a binary tree and corresponding partitions, \mathcal{D}_{rs} , produced by the separating hyperplanes, \mathcal{H}_{rs} . In this case $q = 3$ and $\mathcal{D}_1 \equiv \mathcal{D}_{10}$, $\mathcal{D}_2 \equiv \mathcal{D}_{22}$, and $\mathcal{D}_3 \equiv \mathcal{D}_{23}$.

where $p(\mathbf{z}(k))$ is the probability density function (p.d.f.) of $\mathbf{z}(k)$. If the stochastic signal is ergodic nf:inprod can be replaced by:

$$\langle v_{i_1}(k), v_{i_2}(k) \rangle = \lim_{T \rightarrow \infty} \left\{ \frac{1}{T} \sum_{k=1}^T v_{i_1}(k) v_{i_2}(k) \right\}. \quad (3.29)$$

In the remaining part the Thesis we will use the following definition:

DEFINITION 3.3 *A parametric architecture is said to be orthogonal if*

$$E \{v_{i_1}(k) v_{i_2}(k)\} = 0, \quad \forall i_1 \neq i_2 \quad (3.30)$$

where $i_1, i_2 \in [1; q]$. Whenever $\alpha_0 \neq 0$ (cf. nf:nfcan) it is furthermore required that:

$$E \{v_i(k)\} = 0, \quad \forall i \in [1; q]. \quad (3.31)$$

The major benefit of using an orthogonal architecture lies in the fact that the terms, $v_i(k)$, do not interact linearly. That is, the issue of deciding whether a certain term should enter the architecture or not depends solely on the characteristics of that term, provided we are content with no linear interaction. The subject is further treated in Ch. 6 concerning the determination of suitable filter architectures. Further it should be noted that the above definition of orthogonality is closely related with using a least squares (LS) criterion for estimating the parameters of the filter. This relation will be further elaborated in the discussion of orthogonal polynomial filters, such as the Wiener model based LN-filter, and in Ch. 5.

An immediate attribute of the localized partition function architecture⁸ is that it is an orthogonal architecture. To see this we evaluate the correlation between $v_{i_1}(k)$ and $v_{i_2}(k)$, $i_1 \neq i_2$, by using nf:vsub, (3.28) and the definition of the partition function nf:partfun. Provided that the inner products exist then:

$$\begin{aligned} & E_{\mathbf{z}(k)} \{v_{i_1}(k) v_{i_2}(k)\} \\ &= \int_{\mathcal{D}} v_{i_1}(k) v_{i_2}(k) \cdot p(\mathbf{z}(k)) \, d\mathbf{z}(k) \\ &= \int_{\mathcal{D}} b_{i_1}(\mathbf{z}(k); \beta_{i_1}) \cdot D_{i_1}(\mathbf{z}(k)) \cdot b_{i_2}(\mathbf{z}(k); \beta_{i_2}) \cdot D_{i_2}(\mathbf{z}(k)) \cdot p(\mathbf{z}(k)) \, d\mathbf{z}(k) \\ &= \int_{\mathcal{D}_{i_1}} b_{i_1}(\mathbf{z}(k); \beta_{i_1}) \cdot D_{i_1}(\mathbf{z}(k)) \cdot b_{i_2}(\mathbf{z}(k); \beta_{i_2}) \cdot D_{i_2}(\mathbf{z}(k)) \cdot p(\mathbf{z}(k)) \, d\mathbf{z}(k) \\ &\quad + \int_{\mathcal{D}_{i_2}} b_{i_1}(\mathbf{z}(k); \beta_{i_1}) \cdot D_{i_1}(\mathbf{z}(k)) \cdot b_{i_2}(\mathbf{z}(k); \beta_{i_2}) \cdot D_{i_2}(\mathbf{z}(k)) \cdot p(\mathbf{z}(k)) \, d\mathbf{z}(k) \\ &\quad + \int_{\mathcal{D} \setminus (\mathcal{D}_{i_1} \cup \mathcal{D}_{i_2})} b_{i_1}(\mathbf{z}(k); \beta_{i_1}) \cdot D_{i_1}(\mathbf{z}(k)) \cdot b_{i_2}(\mathbf{z}(k); \beta_{i_2}) \cdot D_{i_2}(\mathbf{z}(k)) \cdot p(\mathbf{z}(k)) \, d\mathbf{z}(k) \\ &= 0. \end{aligned} \quad (3.32)$$

This is due to the fact that $D_{i_2} = 0$ when integrating w.r.t. \mathcal{D}_{i_1} and vice versa. Furthermore, both partition functions equal zero when integrating w.r.t. $\mathcal{D} \setminus (\mathcal{D}_{i_1} \cup \mathcal{D}_{i_2})$. In addition, $\alpha_0 = 0$ is assumed. This proves the stated orthogonality.

Note that $v_{i_1}(k)$, $v_{i_2}(k)$ in the present case in fact are independent since for arbitrary r_1, r_2 : $E_{\mathbf{z}(k)} \{v_{i_1}^{r_1}(k) v_{i_2}^{r_2}(k)\} = 0$.

⁸This includes consequently also the gate function architectures according to the Venn-diagram Fig. 3.6.

3.1.4 Classification of Filter Architectures

Based on the main features:

- Parametric versus nonparametric architectures,
- Global versus local approximation,

it is possible to classify a number of filter architectures provided in the literature. The classification is presented in Table 3.1.

In the following sections we convey detailed discussion and comparison of selected architectures listed in the table.

3.2 Global Approximation

3.2.1 Polynomial Filters

In this subsection we discuss three closely related global approximation filter architectures:

- The Volterra filter,
- The Wiener Model based LN-filter,
- The Chebyshev Filter.

They are all based on an expansion of the nonlinear function $f_n(\cdot)$ in multidimensional polynomials. Provided that the time-domain kernels which define an l 'th order Volterra filter (cf. fa:volfilt) are finite and replacing products of x by adequate polynomials in z the general polynomial filter (see also [Hoffmann 92a, Ch. 2 & Ch. 4]) yields:

$$\begin{aligned}
 \hat{y}(k) &= f_n(\mathbf{z}(k); \mathbf{w}) \\
 &= h_0 + \sum_{n_1=1}^p h_1(n_1)P_1(z_{n_1}(k)) + \sum_{n_1=1}^p \sum_{\substack{n_2=1 \\ n_2 \neq n_1}}^p h_2(n_1, n_2)P_1(z_{n_1}(k))P_1(z_{n_2}(k)) + \\
 &\quad + \sum_{n_1=1}^p h_2(n_1, n_1)P_2(z_{n_1}(k)) + \cdots + \sum_{n_1=1}^p h_l(n_1, \dots, n_l)P_l(z_{n_1}(k)) \quad (3.33)
 \end{aligned}$$

where

- $P_r(z)$ is a polynomial in z of degree r .
- $h_r(n_1, n_2, \dots, n_r)$ is the r 'th order finite time-domain kernel. By nf:polfilgen \mathbf{w} is defined as:

$$\mathbf{w} = [h_0, h_1(1), h_1(2), \dots, h_2(1, 2), \dots, h_l(p, p, \dots, p)]^\top. \quad (3.34)$$

The dimension of \mathbf{w} , m , equals $C_{l+p,l}$ cf. [Schetzen 89] where

$$C_{n,k} = \frac{n!}{(k!(n-k)!)}$$

is the binomial coefficient.

	Local Approximation	Global Approximation
Parametric	Localized Receptive Fields [Moody & Darken 88], [Moody & Darken 89], [Niranjan & Kar-dirkamanathan 91], [Platt 91], [Sanger 91], [Stokbro <i>et al.</i> 90] Gate Function Filters [Hoffmann 92a], [Schetzen 89] Tree-Structured Piecewise-Linear Filter [Gelfand <i>et al.</i> 91] Threshold Autoregressive Models [Tong & Lim 80] Multivariate Adaptive Regression Splines [Friedman 91] Local Experts [Jacobs <i>et al.</i> 91] Quantization based Piecewise-Linear Filter Networks [Sørensen 92]	Polynomial Filter [Hoffmann 92a] Wiener Model [Schetzen 89] Volterra Filter [Schetzen 89] Multi-Layer Neural Network [Haykin 94], [Hertz <i>et al.</i> 91], [Rosenblatt 62], [Widrow & Hoff 60] Gram-Schmidt Neural Networks [Orfanidis 90] Canonical Piecewise-Linear Filters [Kang & Chua 78], [Lin & Unbehauen 90] Semilocal Units [Hartmann & Keeler 91] Neural Networks with FIR/IIR Synapses [Back & Tsoi 91]
Nonparametric	Local Filters [Farmer & Sidoro-wich 88] Nearest-Neighbor Regression [Duda & Hart 73], [Geman <i>et al.</i> 92] Parzen Window Regression [Spect 91]	

Table 3.1: Classification of nonlinear filter architectures. Note that a nonparametric global approximation architecture does not exist since the processing within a nonparametric architecture can be interpreted as an interpolation (or extrapolation) of the training data. Consequently, all data can not contribute with the same weight; remote data (w.r.t. the actual input) have to contribute with less weight.

It is possible to rewrite nf:polfilgen into the canonical representation which yields:

$$\hat{y}(k) = \alpha_0 + \sum_{i=1}^q \alpha_i b_i(\mathbf{z}(k)) = \boldsymbol{\alpha}^\top \mathbf{v} \quad (3.35)$$

where

- $q = (m - 1) = C_{l+p,l} - 1$.
- $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \dots, \alpha_q]^\top = \mathbf{w}$.

•

$$b_i(\mathbf{z}(k)) = P_{r_1(i)}(z_1(k))P_{r_2(i)}(z_2(k)) \cdots P_{r_p(i)}(z_p(k)), \quad 1 \leq i \leq q, \quad (3.36)$$

and the vector $\mathbf{r}(i) = [r_1(i), r_2(i), \dots, r_p(i)]$ is obtained by `nf:polfilgen`. For instance: $\mathbf{r}(1) = [1, 0, 0, \dots, 0]$ and $\mathbf{r}(q) = [0, 0, 0, \dots, l]$. Here $P_0(\cdot)$ per definition is set equal to one.

In Fig. 3.7 the general polynomial filter is depicted.

Certain facts concerning the general polynomial filter should be noted:

- The number of weights (i.e., the number of polynomial terms), $q + 1 = C_{l+p,p}$, grows exponentially fast⁹ with both $p = \dim(\mathbf{z}(k))$ and with the order of the filter, l . Even for moderate values of p and l the number of weights which have to be estimated becomes large. This fact is known as the *curse of dimensionality*¹⁰. In the table below typical values of q are given. In consequence, the number of training data also

p	l	q
5	5	251
5	10	3002
10	3	285
10	5	3002
20	2	230

Table 3.2: Typical number of weights in a polynomial filter.

becomes large (since it is required that $N \geq q$, and typically N/q lies in the range 2–10, see Ch. 6). This fact reduces the feasibilities of the filter since:

- The number of training data available in some application may be sparse.
- The computational complexity involved in estimation of the weights is large (see Ch. 5 and [Hoffmann 92a, Ch. 6.3]).
- The basis functions do not contain any parameters (i.e., β_i does not exist). That is, it is not possible to adjust the shape of the basis function to comply with the present modeling task. On the other hand, the filter output becomes a linear function of the weights, α_i , which is desirable with regard to an estimation point of view. This is further treated in Ch. 5.

⁹Applying Stirling's formula [Abramowitz & Stegun 72, p. 257] to approximate the factorial function, we get for large l, p :

$$q \approx \frac{1}{\sqrt{2\pi}} \exp\left(\left(l + p + \frac{1}{2}\right) \ln(l + p) - \left(l + \frac{1}{2}\right) \ln(l) - \left(p + \frac{1}{2}\right) \ln(p)\right)$$

where $\ln(\cdot)$ is the natural logarithm.

¹⁰The idea was first introduced in R.E. BELLMAN: *Adaptive Control Processes*, Princeton University Press, 1961.

3.2.1.1 Volterra Filter

The Volterra filter results by setting:

$$P_r(z) = z^r. \quad (3.37)$$

In general no assumption concerning the nature of the nonlinearity, $g_n(\mathbf{z}(k))$, of the system being modeled, is made. Hence, we require that the specific filter architecture is capable of implementing a wide class of nonlinear functions. Concerning the Volterra filter Fréchet [Schetzen 89, App. B] showed that any *continuous function* of $\mathbf{z}(k) \in \mathcal{I}$, where $\mathcal{I} \subseteq \mathcal{D}$ is a compact subset, can be approximated arbitrarily close. $\forall \mathbf{z}(k) \in \mathcal{I}$, (point-wise convergence) by a Volterra filter. That is,

$$f_n(\mathbf{z}(k)) \rightarrow g_n(\mathbf{z}(k)) \text{ as } l \rightarrow \infty. \quad (3.38)$$

In the neural network literature this property often is referred to as the capability of *universal approximation* (e.g., in [Hertz et al. 91]). Anticipating some of the material in Ch. 6, the property of universal approximation is per se no guarantee for a proper filter design. The ultimate goal is to exploit the knowledge contained in the training set so that the filter performs the best possible w.r.t. some performance measure which will be the quintessence of generalization ability (see Ch. 6).

The Volterra filter is afflicted with some drawbacks. First, recall that a Volterra filter can be conceived as a multidimensional Taylor series expansion. This implies that convergence is ensured in the subset, \mathcal{I} , only. However, the subset may merely constitute a small fraction of the entire input space of interest, i.e., \mathcal{D} . Secondly, the output of the basis functions, i.e., the $v_i(k)$ signals, will be highly correlated¹¹. The consequence is that certain algorithms (e.g., the LMS-algorithm) for estimating the weights, α_i , will converge slowly. However, the performance of the filter (i.e., generalization ability) does not depend on the correlation among the $v_i(k)$ signals (see further Sec. 6.3.6).

3.2.1.2 Wiener Model based LN-Filter

In order to circumvent some of the obstacles associated with the Volterra filter other types of polynomials may be introduced. The idea is to make the polynomials, i.e., the basis functions, orthogonal cf. Sec. 3.1.3. Assume that $\mathbf{z}(k)$ is stochastic and furthermore that the basis functions (polynomials) are orthogonal w.r.t. to the inner product definition `mf:inprod`, i.e.,

$$\begin{aligned} E_{\mathbf{z}(k)} \{b_{i_1}(\mathbf{z}(k))b_{i_2}(\mathbf{z}(k))\} &= E \{v_{i_1}(k)v_{i_2}(k)\} \\ &= \begin{cases} E \{v_{i_1}^2\} & , i_1 = i_2 \\ 0 & , i_1 \neq i_2 \end{cases}, \end{aligned} \quad (3.39)$$

and

$$E_{\mathbf{z}(k)} \{b_i(\mathbf{z}(k))\} = E \{v_i(k)\} = 0, \quad \forall i \in [1; q]. \quad (3.40)$$

In general we define the *adjustment error*:

$$e_a(k; \mathbf{w}) = g_n(\mathbf{z}(k)) - f_n(\mathbf{z}(k); \mathbf{w}) \quad (3.41)$$

¹¹In App. B a numerical example of this correlation is given. Fig. B.2 shows the condition number of the Hessian matrix, \mathbf{H} , which – with the present notation – is defined as: $\mathbf{H} = E\{\mathbf{v}(k)\mathbf{v}^T(k)\}$.

which measures the error done when using the architecture $f_n(\cdot)$ parameterized by \mathbf{w} in order to approximate the nonlinear system $g_n(\mathbf{z}(k))$. In the present case the adjustment error when using q polynomial terms becomes:

$$\begin{aligned} e_a(k; \boldsymbol{\alpha}) &= g_n(\mathbf{z}(k)) - \alpha_0 - \sum_{i=1}^q \alpha_i b_i(\mathbf{z}(k)) \\ &= g_n(\mathbf{z}(k)) - \boldsymbol{\alpha}^\top \mathbf{v}(k) \end{aligned} \quad (3.42)$$

where the notation of `nf:nfcan` is adapted.

The approximation capability of the filter (with the present definition of inner product) is measured by mean squared adjustment error, $G(\boldsymbol{\alpha})$, i.e.,

$$G(\boldsymbol{\alpha}) = E\{e_a^2(k; \boldsymbol{\alpha})\} = E\left\{\left[g_n(\mathbf{z}(k)) - \boldsymbol{\alpha}^\top \mathbf{v}(k)\right]^2\right\}. \quad (3.43)$$

In the context we assume that the function $g_n(\cdot)$ obeys:

$$E\left\{g_n^2(\mathbf{z}(k))\right\} < \infty \quad (3.44)$$

For any fixed q the optimal weight vector, $\boldsymbol{\alpha}^*$, which minimizes $G(\boldsymbol{\alpha})$ is given by:

$$\boldsymbol{\alpha}^* = \mathbf{H}^{-1} \boldsymbol{\phi}_{g_n} \mathbf{v} \quad (3.45)$$

where

$$\mathbf{H} = \text{diag}\left\{\left[1, E\{v_1^2(k)\}, E\{v_2^2(k)\}, \dots, E\{v_q^2(k)\}\right]\right\}, \quad (3.46)$$

$$\boldsymbol{\phi}_{g_n} \mathbf{v} = E\left\{g_n(\mathbf{z}(k)) \mathbf{v}(k)\right\}. \quad (3.47)$$

This is due to the fact that the nonlinear filter is linear in the parameters (i.e., an LN-filter) so that the minimization of $G(\boldsymbol{\alpha})$ is equivalent to solving a linear least squares (LS) problem e.g., [Haykin 91], [Seber & Wild 89, Ch. 2.1] (see further Ch. 5). The mean squared adjustment error at the optimal weights yields cf. `nf:galf` and `nf:aloft`–(3.47):

$$\begin{aligned} G(\boldsymbol{\alpha}^*) &= E\left\{g_n^2(\mathbf{z}(k))\right\} - 2E\left\{g_n(\mathbf{z}(k)) (\boldsymbol{\alpha}^*)^\top \mathbf{v}(k)\right\} + E\left\{(\boldsymbol{\alpha}^*)^\top \mathbf{v}(k) \mathbf{v}^\top(k) \boldsymbol{\alpha}^*\right\} \\ &= E\left\{g_n^2(\mathbf{z}(k))\right\} - 2(\boldsymbol{\alpha}^*)^\top \boldsymbol{\phi}_{g_n} \mathbf{v} + (\boldsymbol{\alpha}^*)^\top \mathbf{H} \boldsymbol{\alpha}^* \\ &= E\left\{g_n^2(\mathbf{z}(k))\right\} - 2(\boldsymbol{\alpha}^*)^\top \mathbf{H} \boldsymbol{\alpha}^* + (\boldsymbol{\alpha}^*)^\top \mathbf{H} \boldsymbol{\alpha}^* \\ &= E\left\{g_n^2(\mathbf{z}(k))\right\} - (\alpha_0^*)^2 - \sum_{i=1}^q (\alpha_i^*)^2 E\left\{v_i^2(k)\right\}. \end{aligned} \quad (3.48)$$

Since per definition $G(\boldsymbol{\alpha}) \geq 0, \forall \boldsymbol{\alpha}$ and the terms in `nf:galf` involving α_i^* all are positive the following inequality results:

$$(\alpha_0^*)^2 + \sum_{i=1}^q (\alpha_i^*)^2 E\left\{v_i^2(k)\right\} \leq E\left\{g_n^2(\mathbf{z}(k))\right\}. \quad (3.49)$$

If the basis functions constitute a complete orthogonal functional set (w.r.t. the inner product definition) and $g_n(\cdot)$ is a piecewise continuous function then Parseval's equation gives [Schetzen 89]:

$$(\alpha_0^*)^2 + \lim_{q \rightarrow \infty} \sum_{i=1}^q (\alpha_i^*)^2 E\left\{v_i^2(k)\right\} = E\left\{g_n^2(\mathbf{z}(k))\right\}. \quad (3.50)$$

That is, the infinite set of orthogonal basis functions approximates the unknown function $g_n(\cdot)$ with zero mean square adjustment error. This is also known as: Convergence in mean [Schetzen 89].

In contrast to the Volterra filter a polynomial filter based on an orthogonal expansion is convenient as one is able to ensure universal approximation within the entire input space, \mathcal{D} .

Within the Wiener model it is assumed that $\mathbf{z}(k)$ is Gaussian distributed with zero mean vector and covariance matrix equal to the identity matrix, i.e., $\mathbf{z}(k) \in \mathcal{N}(\mathbf{0}, \mathbf{I})$. The individual signals, $z_j(k)$, $j \in [1; p]$ are thus mutually independent¹². Construction of a signal, $z_j(k)$, which obeys the above properties depends on the p.d.f. of the input signal, $x(k)$, and the demands on the memory length, L . Here we merely mention the assumptions made in the Wiener model. First, the input is assumed to be a zero mean Gaussian sequence, i.e., $x(k) \in \mathcal{N}(0, \sigma_x^2)$. Secondly, infinite memory length ($L \rightarrow \infty$) is considered, i.e the input signal vector, $\mathbf{x}(k) = [x(k), x(k-1), x(k-2), \dots]$ is infinite. In order to make $z_j(k)$ Gaussian only linear filtering of $x(k)$ is allowed. Consequently, $z_j(k)$ can be expressed as the convolution:

$$z_j(k) = \sum_{n=0}^{\infty} h_j(n)x(n-k), \quad j \in [1; p] \quad (3.51)$$

where $h_j(n)$ is the impulse response of the j 'th filter. Since the $z_j(k)$ is required to be mutually independent (uncorrelated), i.e., $E\{z_{j_1}(k)z_{j_2}(k)\} = 0$, $j_1 \neq j_2$. Let us evaluate this correlation. According to nf:zj:

$$\begin{aligned} E\{z_{j_1}(k)z_{j_2}(k)\} &= E\left\{\sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} h_{j_1}(n_1)h_{j_2}(n_2)x(n_1-k)x(n_2-k)\right\} \\ &= \sigma_x^2 \sum_{n=0}^{\infty} h_{j_1}(n)h_{j_2}(n). \end{aligned} \quad (3.52)$$

That is, the impulse responses then have to obey:

$$\sum_{n=0}^{\infty} h_{j_1}(n)h_{j_2}(n) = 0, \quad j_1 \neq j_2. \quad (3.53)$$

A set of functions which comply with this orthogonality condition is the discrete Laguerre functions (see e.g., [Masnadi-Shirazi & Ahmed 91], [Hoffmann 92a]). The \mathbf{z} -transform of the Laguerre functions is given by:

$$H_1(\mathbf{z}) = \frac{\sqrt{1-\theta^2}}{1-\theta\mathbf{z}^{-1}}, \quad (3.54)$$

$$H_j(\mathbf{z}) = \frac{-\theta + \mathbf{z}^{-1}}{1-\theta\mathbf{z}^{-1}}H_{j-1}(\mathbf{z}), \quad 2 \leq j \leq q, \quad (3.55)$$

and $0 < \theta < 1$. The choice of θ is discussed in Ch. 4. $H_1(\mathbf{z})$ is a first order recursive low-pass filter. The succeeding filters are obtained by constantly multiplying with the all-pass filter: $-\theta + \mathbf{z}^{-1}/1 - \theta\mathbf{z}^{-1}$. The time-domain expression of $z_j(k)$ then becomes:

$$z_1(k) = \theta z_1(k) + \sqrt{1-\theta^2}x(k), \quad (3.56)$$

$$z_j(k) = \theta z_j(k) - \theta z_{j-1}(k) + z_{j-1}(k-1), \quad 2 \leq j \leq q. \quad (3.57)$$

In Fig. 3.8 the construction of the $z_j(k)$ signals are shown. It should be noted that the Laguerre functions furthermore are orthonormal, i.e.,

$$\sum_{n=0}^{\infty} h_{j_1}(n)h_{j_2}(n) = \begin{cases} 1 & , j_1 = j_2 \\ 0 & , j_1 \neq j_2 \end{cases} . \quad (3.58)$$

This implies cf. nf:corzj that: $\sigma_{z_j}^2 = \sigma_x^2$, $j \in [1;p]$. Since we require that the variances of the $z_j(k)$ signals to be unity the input signal has to be scaled with an estimate of the standard deviation, σ_x .

The above construction of the $z_j(k)$ signals enables the use of the Hermite polynomials in order to make the $v_i(k)$ signals uncorrelated. The Hermite polynomials are generated so that they are orthogonal w.r.t. the inner product definition nf:inprod. Recall that the $z_j(k) \in \mathcal{N}(0,1)$, i.e., the p.d.f. becomes:

$$p(z_j(k)) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_j^2(k)}{2}\right). \quad (3.59)$$

Further let $\text{He}_r(z)$ be the Hermite polynomial of order r of the scalar variable z (writing z for shorthand of $z_j(k)$). It can be shown [Schetzen 89, Ch. 19], [Abramowitz & Stegun 72, p. 775] that the Hermite polynomials meet the following orthogonality relation: $\forall r_1, r_2 \in [0; l]$

$$E_z \{\text{He}_{r_1}(z)\text{He}_{r_2}(z)\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left(-\frac{z^2}{2}\right) \text{He}_{r_1}(z)\text{He}_{r_2}(z) dz = \delta(r_1 - r_2) \cdot r_1! \quad (3.60)$$

where $\delta(\cdot)$ is the Kronecker delta function and l is the order of the polynomial filter. This orthogonality relation along with the fact that the $z_j(k)$'s are mutually independent ensures that the $v_i(k)$'s are mutually uncorrelated. To establish this result we note that:

$$v_i(k) = \text{He}_{r_1(i)}(z_1(k))\text{He}_{r_2(i)}(z_2(k)) \cdots \text{He}_{r_p(i)}(z_p(k)), \quad 1 \leq i \leq q, \quad (3.61)$$

cf. nf:polbas. The correlation between $v_{i_1}(k)$ and $v_{i_2}(k)$ then becomes:

$$\begin{aligned} E \{v_{i_1}(k)v_{i_2}(k)\} &= E \left\{ \text{He}_{r_1(i_1)}(z_1(k))\text{He}_{r_1(i_2)}(z_1(k))\text{He}_{r_2(i_1)}(z_2(k))\text{He}_{r_2(i_2)}(z_2(k)) \right. \\ &\quad \left. \cdots \text{He}_{r_p(i_1)}(z_p(k))\text{He}_{r_p(i_2)}(z_p(k)) \right\} \\ &= \prod_{j=1}^p E \left\{ \text{He}_{r_j(i_1)}(z_j(k))\text{He}_{r_j(i_2)}(z_j(k)) \right\} \\ &= 0, \quad \exists j : r_j(i_1) \neq r_j(i_2). \end{aligned} \quad (3.62)$$

The Hermite polynomials can be generated by the simple recursive formula:

$$\text{He}_{r+1}(z) = z\text{He}_r(z) - r\text{He}_{r-1}(z), \quad r \geq 1 \quad (3.63)$$

with $\text{He}_0(z) = 1$, $\text{He}_1(z) = z$.

The Wiener model based LN-filter is depicted in Fig. 3.9 In general where no assumptions neither on the input distribution nor on the autocorrelation function of the input are available we face two alternatives:

¹²Gaussian variables which are uncorrelated are furthermore independent.

1. Choosing a set of polynomials which under certain conditions are orthogonal and besides are believed to be less correlated than the Volterra polynomials. The Chebyshev filter mentioned below is an example of this strategy.
2. Forming a set of orthogonal polynomial terms by using an orthogonal transformation of the Volterra basis functions. In [Korenberg & Paarmann 91] a Gram-Schmidt orthogonalizing procedure is used. Let $\mathbf{b}(k)$ be the (augmented) vector of Volterra basis functions:

$$\mathbf{b}(k) = [1, b_1(\mathbf{z}(k)), b_2(\mathbf{z}(k)), \dots, b_q(\mathbf{z}(k))]^\top. \quad (3.64)$$

The $\mathbf{v}(k)$ vector signal then appears from:

$$\mathbf{v}(k) = \mathbf{U}^\top \mathbf{b}(k). \quad (3.65)$$

where \mathbf{U}^\top is the $(q+1) \times (q+1)$ unit lower triangular matrix:

$$\mathbf{U}^\top = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ u_{10} & 1 & 0 & \cdots & 0 & 0 \\ u_{20} & u_{21} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{q0} & u_{q1} & u_{q2} & \cdots & u_{q,q-1} & 1 \end{bmatrix} \quad (3.66)$$

which forms the Gram-Schmidt orthogonalizing by adjusting¹³ u_{i_1, i_2} , $i_1, i_2 \in [0; q]$ so that $E\{v_{i_1}(k)v_{i_2}(k)\} = 0$, $\forall i_1 \neq i_2$.

3.2.1.3 Chebyshev Filter

The Chebyshev filter [Hoffmann 92a, Ch. 4] is an attempt to accomplish the wish for constructing the $v_i(k)$ signals such that they are relatively weakly correlated for most ordinary input signals (e.g., Gaussian, uniform, sinusoids). The Chebyshev filter results by using the Chebyshev polynomials $T_r(z)$ in `mf:polfilgen`. These are given by the following recursive formula [Abramowitz & Stegun 72, p. 782]:

$$T_{r+1}(z) = 2zT_r(z) - T_{r-1}(z), \quad r \geq 1 \quad (3.67)$$

with $T_0(z) = 1$ and $T_1(z) = z$. It turns out that both the roots and extrema are located in the interval $z \in [-1; 1]$. Consequently, the graphs of the Chebyshev polynomials of different orders, r , are very different in this interval. This provides a hope for the $v_i(k)$ signals to be weakly correlated. It should be emphasized that the $z_i(k)$ signals then have to be restricted to the interval $[-1; 1]$. This can be done by proper scaling and limitation, e.g., using a smooth limiter like the hyperbolic tangent (see page 20), as proposed in [Hoffmann 92a, Ch. 4].

Let the input be a purely deterministic, aperiodic signal composed of sinusoids with frequencies which do not divide, i.e.,

$$x(k) = \sum_{j=1}^p \sin(\omega_j k + \phi_j) \quad (3.68)$$

where $\omega_j \geq 0$, $\phi_j \in [0; 2\pi]$ is the angular frequencies and phase of the j 'th sinusoid, respectively. Further, it is assumed that $w_{j_1} \neq n \cdot w_{j_2}$, $\forall j_1, j_2 \in [1; p]$ where n is a positive

¹³This subject is further mentioned on page 65.

integer. In consequence, the expression nf:sinus is not able to constitute a Fourier series. Furthermore, let

$$z_j(k) = h_j(k) * x(k), \quad j = 1, 2, \dots, p \quad (3.69)$$

where $h_j(k)$, $j \in [1; p]$ is a set of band-pass filters such that

$$z_j(k) = \sin(\omega_j k + \phi_j). \quad (3.70)$$

The $v_i(k)$ can then be shown to be orthogonal (w.r.t. the definition nf:ortinf) [Hoffmann 92a, Ch. 4].

Also in the case where the $z_j(k)$ are stochastic signals orthogonality can be ensured (see [Hoffmann 92a, Ch. 4], [Abramowitz & Stegun 72, p. 774]. This requires that $z_j(k)$, $j \in [1; p]$, are mutually independent and further that the marginal p.d.f. takes the form:

$$p(z_j(k)) = \begin{cases} \frac{1}{\pi \sqrt{1 - z_j^2(k)}} & , |z_j(k)| < 1 \\ 0 & , |z_j(k)| \geq 1 \end{cases}. \quad (3.71)$$

A p.d.f. of this form turns up when $z_j(k)$ is a sinusoid with stochastic phase, ϕ_j , uniformly distributed over the interval $[0; 2\pi]$, see [Bendat & Piersol 86, p. 53].

3.2.2 Multi-Layer Neural Networks

The neural networks under consideration is a restriction of the general neural network defined in Def. 1.1. The restrictions consist in:

- No feedback connections are present, i.e., a neuron never receive an input signal which is a function of its own output. Consequently, all processing involved is feed-forward.
- The neurons are organized in layers such that the outputs of the neurons in a specific layer act as the inputs for the neurons in the next layer. Consequently, all processing involved is feed-forward. It is possible to relax the inter-connectivity¹⁴ of the neurons and still maintaining the feed-forward structure. This is done by allowing the outputs of the neurons in a specific layer to feed not only the next layer but all succeeding layers.
- The used nonlinear neurons are *perceptrons* [Hertz et al. 91], [Rosenblatt 58], [Widrow & Hoff 60].
- The output neuron is linear (cf. `int:linneu`).

A neural network which complies with the restrictions above (the last excepted) is denoted a *multi-layer feed-forward perceptron neural network* (MFPNN). If the network does not consists of perceptrons (but other types of neurons) we use the designation: MFNN. In Fig. 3.10 a multi-layer feed-forward neural network implementation of the XN-filter is shown. The depicted neural network is a 3-layer network since it consists of two hidden layers and an output layer with one linear neuron. In this terminology the inputs are not considered as an independent layer¹⁵. In general we consider a l layer network with

¹⁴The concept inter-connectivity is related to how the layers are connected. If no connection between two neurons exists then the associated weight equals zero.

¹⁵However, some authors counts the inputs as an independent layer.

m_r neurons in the r 'th layer, $r = 1, \dots, l$. m_0 is the number of inputs, i.e., $m_0 = p$. Furthermore, $m_l = 1$ as only one output neuron is present. In general we refer to a \mathbf{m} -network where $\mathbf{m} = [m_0, m_1, \dots, m_l]$. The output signals of the neurons in the r 'th layer are denoted, $s_i^{(r)}(k)$, $i = 1, 2, \dots, m_r$ and assembled in the state vector:

$$\tilde{\mathbf{s}}^{(r)}(k) = \left[s_1^{(r)}(k), s_2^{(r)}(k), \dots, s_{m_r}^{(r)}(k) \right]^\top. \quad (3.72)$$

Further we define the augmented state vector:

$$\mathbf{s}^{(r)}(k) = \left[\mathbf{1}, \left(\tilde{\mathbf{s}}^{(r)}(k) \right)^\top \right]^\top. \quad (3.73)$$

Per definition: $\mathbf{s}^{(0)}(k) = [\mathbf{1}, \mathbf{z}(k)^\top]^\top$ where $\mathbf{z}(k) = [z_1(k), z_2(k), \dots, z_p(k)]^\top$, see Fig. 3.10.

Nonlinear Perceptron The signal processing within the i 'th (nonlinear) perceptron in the r 'th layer is shown in Fig. 3.11 and obeys the following equations:

$$\begin{aligned} u_i^{(r)}(k) &= w_{i,0}^{(r)} + \sum_{j=1}^{m_{r-1}} s_j^{(r-1)}(k) w_{i,j}^{(r)} \\ &= \left(\mathbf{w}_i^{(r)} \right)^\top \mathbf{s}^{(r-1)}(k) \\ s_i^{(r)}(k) &= h \left(u_i^{(r)}(k) \right) \end{aligned} \quad (3.74)$$

where

- $\mathbf{w}_i^{(r)} = [w_{i,0}^{(r)}, w_{i,1}^{(r)}, \dots, w_{i,m_{r-1}}^{(r)}]^\top$ is the $m_{r-1} + 1$ weights (parameters) associated with the i 'th perceptron.
- $u_i^{(r)}(k)$ is the response of a linear neuron (see below); corresponding to a linear combination of a DC signal equal to one and the signals, $s_j^{(r)}(k)$, $j \in [1; m_r]$.
- $h(u)$ is the nonlinear *activation function*. When dealing with traditional perceptrons the activation function is a *sigmoid function* which in its most general form complies with:

$$h(u) \rightarrow \begin{cases} K_\infty & , u \rightarrow \infty \\ K_{-\infty} & , u \rightarrow -\infty \end{cases} \quad (3.75)$$

where $K_\infty, K_{-\infty}$ are different finite real numbers. Two typical sigmoid functions with $K_{\pm\infty} = \pm 1$ are the signum function,

$$\text{sgn}(x) = \begin{cases} 1 & , x > 0 \\ 0 & , x = 0 \\ -1 & , x < 0 \end{cases} \quad (3.76)$$

and the hyperbolic tangent, $\tanh(\cdot)$ which are depicted in Fig. 3.12.

A geometrical interpretation of the processing within the perceptron is possible. Consider first the case where $h(u) = \text{sgn}(u)$. According to nf:perceproc the output of a neuron equals:

$$s_i^{(r)}(k) = \text{sgn} \left(\left(\mathbf{w}_i^{(r)} \right)^\top \mathbf{s}^{(r-1)}(k) \right). \quad (3.77)$$

Let the $\mathcal{H}_i^{(r)+}$, $\mathcal{H}_i^{(r)-}$ be the positive and negative region, respectively, which obey:

$$\mathcal{H}_i^{(r)+} = \left\{ \tilde{\mathbf{s}}^{(r-1)} : s_i^{(r)} = 1 \right\}, \quad (3.78)$$

$$\mathcal{H}_i^{(r)-} = \left\{ \tilde{\mathbf{s}}^{(r-1)} : s_i^{(r)} = -1 \right\}. \quad (3.79)$$

The regions are separated by the hyperplane:

$$\mathcal{H}_i^{(r)} : \left(\mathbf{w}_i^{(r)} \right)^\top \cdot \mathbf{s}^{(r-1)}(k) = \left(\tilde{\mathbf{w}}_i^{(r)} \right)^\top \cdot \tilde{\mathbf{s}}^{(r-1)}(k) + w_{i,0}^{(r)} = 0 \quad (3.80)$$

where $\tilde{\mathbf{w}}_i^{(r)} = [w_{i,1}^{(r)}, w_{i,2}^{(r)}, \dots, w_{i,m_{r-1}}^{(r)}]^\top$ is the normal vector of the hyperplane and $w_{i,0}^{(r)}$ is the threshold (also denoted a bias weight). The individual perceptrons thus divide the $\tilde{\mathbf{s}}^{(r-1)}$ -space by a hyperplane which orientation is specified by the weight vector $\mathbf{w}_i^{(r)}$. The output of the perceptron is positive in the region pointed out by the normal vector, $\tilde{\mathbf{w}}_i^{(r)}$, and negative otherwise. When using the activation function $h(u) = \tanh(u)$ the above interpretations hold except that a smooth transition between the positive and negative regions are introduced. In Fig. 3.13 the geometry of a $\tanh(\cdot)$ -perceptron is shown. Unless anything else is mentioned we will employ the hyperbolic tangent as the activation function in the following.

Linear Perceptron The output neuron is a linear neuron which also is referred to as a linear perceptron [Hertz et al. 91]. Recall from int:linneu that the processing of the linear output neuron is:

$$\hat{y}(k) = \left(\mathbf{w}_1^{(l)} \right)^\top \mathbf{s}^{(l-1)}. \quad (3.81)$$

The reason for employing a linear output neuron lies in the fact that the desired range of the output is all real numbers. Furthermore, a bias compensation on the output is normally necessary. This is voluntarily provided when using a linear output neuron due to the weight, $w_{10}^{(l)}$. Furthermore, a linear output neuron is required in order to accomplish universal approximation (see below).

Signal Processing within the MFPNN The weights associated with all neurons in the r layer are assembled in the $m_r \times (m_{r-1} + 1)$ weight matrix:

$$\mathbf{W}^{(r)} = \left[\mathbf{w}_1^{(r)}, \mathbf{w}_2^{(r)}, \dots, \mathbf{w}_{m_r}^{(r)} \right]^\top = \begin{bmatrix} w_{10}^{(r)} & w_{11}^{(r)} & \cdots & w_{1,m_{r-1}}^{(r)} \\ w_{20}^{(r)} & w_{21}^{(r)} & \cdots & w_{2,m_{r-1}}^{(r)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m_r,0}^{(r)} & w_{m_r,1}^{(r)} & \cdots & w_{m_r,m_{r-1}}^{(r)} \end{bmatrix}. \quad (3.82)$$

With this notation the processing in the r 'th layer of the network simply is expressed as:

$$\mathbf{u}^{(r)}(k) = \mathbf{W}^{(r)} \mathbf{s}^{(r-1)}(k) \quad (3.83)$$

$$\tilde{\mathbf{s}}^{(r)}(k) = \mathbf{h} \left(\mathbf{u}^{(r)}(k) \right) \quad (3.84)$$

where $\mathbf{h}(\cdot)$ is the vector activation function:

$$\mathbf{h} \left(\mathbf{u}^{(r)}(k) \right) = \left[h(u_1^{(r)}(k)), h(u_2^{(r)}(k)), \dots, h(u_{m_r}^{(r)}(k)) \right]^\top. \quad (3.85)$$

Hence, the processing in the MFPNN with linear output neuron yields:

$$\begin{aligned}\hat{y}(k) &= f_n(\mathbf{z}(k); \mathbf{w}) \\ &= \mathbf{W}^{(l)} \left(\mathbf{h} \left(\mathbf{W}^{(l-1)} \dots \mathbf{h} \left(\mathbf{W}^{(1)} \mathbf{s}^{(0)}(k) \right) \dots \right) \right)\end{aligned}\quad (3.86)$$

where \mathbf{w} is the weight vector containing all weights, i.e.,

$$\mathbf{w} = \left[\left(\mathbf{w}_1^{(1)} \right)^\top, \left(\mathbf{w}_2^{(1)} \right)^\top, \dots, \left(\mathbf{w}_{m_l}^{(l)} \right)^\top \right]^\top. \quad (3.87)$$

The output thus results by repeatedly affine¹⁶ and nonlinear mappings of the input vector signal, $\mathbf{z}(k)$. The total number of weights, m in the MFPNN is given by:

$$m = \sum_{r=1}^l (m_{r-1} + 1) \cdot m_r. \quad (3.88)$$

Per definition: $m_0 = p$, $m_l = 1$, and in particular, a 2-layer network contains $m = m_1 \cdot (p + 2) + 1$ weights. In the table below typical values of m are given.

l	p	m_1	m_2	m
2	5	10	-	71
2	5	20	-	141
2	10	10	-	121
2	10	20	-	241
3	5	10	10	181
3	5	20	20	561
3	10	10	10	231
3	10	20	20	661

Table 3.3: The number of weights, m , in an MFPNN. When $l = 2$ then $q = m_1$ and $l = 3$ implies $q = m_2$.

By comparison of `nf:mlnn` with `nf:nfcan` we get the following canonical representation of the MFPNN:

$$\hat{y}(k) = \alpha_0 + \sum_{i=1}^q \alpha_i b_i(\mathbf{z}(k); \beta_i) \quad (3.89)$$

where

- $q = m_{l-1}$.

- α is the $q + 1$ dimensional vector: $\alpha = \left(\mathbf{W}^{(l)} \right)^\top$.

-

$$b_i(\mathbf{z}(k); \beta_i) = h \left(\left(\mathbf{w}_i^{(l-1)} \right)^\top \left(\mathbf{h} \left(\mathbf{W}^{(l-2)} \dots \mathbf{h} \left(\mathbf{W}^{(1)} \mathbf{s}^{(0)}(k) \right) \dots \right) \right) \right). \quad (3.90)$$

¹⁶The multiplication with a weight matrix constitutes an affine mapping since the first component in the state vector $\mathbf{s}^{(r)}$ always equals one.

- β_i , $i \in [1; q]$ is defined as:

$$\beta_i = \left[\left(\mathbf{w}_1^{(1)} \right)^\top, \left(\mathbf{w}_2^{(1)} \right)^\top, \dots, \left(\mathbf{w}_{m_{l-2}}^{(l-2)} \right)^\top, \left(\mathbf{w}_i^{(l-1)} \right)^\top \right]^\top, \quad (3.91)$$

and $\forall i$

$$\dim(\beta_i) = m_{l-2} + 1 + \sum_{r=1}^{l-2} (m_{r-1} + 1) \cdot m_r.$$

In contrast to the polynomial filters the basis functions are parameterized (adaptable) when using an MFPNN architecture. This circumstance may be profitable as it provides a greater flexibility. For instance, even though a MFPNN, in principle, is a global approximation approach it is possible to make local representations of the input domain. This is done by letting some the neurons in the first layer cooperate in order to constitute a *semi localized covering function*. A semi localized covering function is defined as a localized covering function (see p. 32) which degenerates in some directions of the input space, i.e., no localization is present in these directions. It should be emphasized that localization is not guaranteed in advance. Only if the filter performance is diminished by forming a localization it is going to appear.

Consider the contribution, say \tilde{u} , from the first and second neuron in the first layer to the output of the first neuron in the second layer. Now,

$$\tilde{u} = w_{11}^{(2)} \tanh \left(\left(\tilde{\mathbf{w}}_1^{(1)} \right)^\top \mathbf{s}^{(0)} + w_{10}^{(1)} \right) + w_{12}^{(2)} \tanh \left(\left(\tilde{\mathbf{w}}_2^{(1)} \right)^\top \mathbf{s}^{(0)} + w_{20}^{(1)} \right). \quad (3.92)$$

Suppose that

1. $\text{sgn} \left(w_{11}^{(2)} \right) = -\text{sgn} \left(w_{12}^{(2)} \right)$ and $w_{11}^{(2)} \neq 0$, $w_{12}^{(2)} \neq 0$,
2. $\frac{\left(\tilde{\mathbf{w}}_1^{(1)} \right)^\top \tilde{\mathbf{w}}_2^{(1)}}{\left| \tilde{\mathbf{w}}_1^{(1)} \right| \cdot \left| \tilde{\mathbf{w}}_2^{(1)} \right|} \approx 1$,
3. $w_{10}^{(1)} \neq w_{20}^{(1)}$,

then \tilde{u} will form a semi localized covering function. This mechanism was first mentioned in [Lapedes & Farber 87] as the ability of forming a “bump”. Fig. 3.14 shows an example of a semi localized covering function when considering a two dimensional input, i.e., $\dim(\mathbf{z}) = 2$. Here we used: $w_{11}^{(2)} = -w_{12}^{(2)} = 1$, $\tilde{\mathbf{w}}_1^{(1)} = \tilde{\mathbf{w}}_2^{(1)} = [1, 0]^\top$, and $w_{10}^{(1)} = -w_{20}^{(1)} = 0.5$. Note that a localized covering function can be formed by adding two more neurons with hyperplane normal vectors perpendicular to those used in nf:utilde, i.e., perpendicular to $\tilde{\mathbf{w}}_1^{(1)}$ and $\tilde{\mathbf{w}}_2^{(1)}$.

Further elaborations on the flexibility induced by using parameterized basis functions are provided in Sec. 3.2.2.

The parameterization of $f_n(\cdot)$ using an MFPNN is characterized by two properties: *Ambiguosness* and *degeneration*.

The nonlinear filtering function, $f_n(\cdot)$ is said to be ambiguous if there exist two or more weight vectors which implement exactly the same nonlinear function. Ambiguousness within an MFPNN arises from several symmetries:

- Two neurons within the same layer can be permuted without altering the nonlinear filtering function, $f_n(\cdot)$, if the corresponding weights into and from the neurons are interchanged. To be concise, consider the neurons i_1, i_2 in the r 'th layer. Then the weight matrices $\mathbf{W}^{(r)}, \mathbf{W}^{(r+1)}$ should be altered as follows: From

$$\begin{bmatrix} \vdots & & & & \\ w_{i_1,0}^{(r)} & w_{i_1,1}^{(r)} & \cdots & w_{i_1,m_{r-1}-1}^{(r)} & w_{i_1,m_{r-1}}^{(r)} \\ \vdots & & & & \\ w_{i_2,0}^{(r)} & w_{i_2,1}^{(r)} & \cdots & w_{i_2,m_{r-1}-1}^{(r)} & w_{i_2,m_{r-1}}^{(r)} \\ \vdots & & & & \end{bmatrix} \quad (3.93)$$

to

$$\begin{bmatrix} \vdots & & & & \\ w_{i_2,0}^{(r)} & w_{i_2,1}^{(r)} & \cdots & w_{i_2,m_{r-1}-1}^{(r)} & w_{i_2,m_{r-1}}^{(r)} \\ \vdots & & & & \\ w_{i_1,0}^{(r)} & w_{i_1,1}^{(r)} & \cdots & w_{i_1,m_{r-1}-1}^{(r)} & w_{i_1,m_{r-1}}^{(r)} \\ \vdots & & & & \end{bmatrix}, \quad (3.94)$$

and from

$$\begin{bmatrix} \cdots & w_{1,i_1+1}^{(r+1)} & \cdots & w_{1,i_2+1}^{(r+1)} & \cdots \\ \cdots & w_{2,i_1+1}^{(r+1)} & \cdots & w_{2,i_2+1}^{(r+1)} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \cdots & w_{m_{r+1}-1,i_1+1}^{(r+1)} & \cdots & w_{m_{r+1}-1,i_2+1}^{(r+1)} & \cdots \\ \cdots & w_{m_{r+1},i_1+1}^{(r+1)} & \cdots & w_{m_{r+1},i_2+1}^{(r+1)} & \cdots \end{bmatrix} \quad (3.95)$$

to

$$\begin{bmatrix} \cdots & w_{1,i_2+1}^{(r+1)} & \cdots & w_{1,i_1+1}^{(r+1)} & \cdots \\ \cdots & w_{2,i_2+1}^{(r+1)} & \cdots & w_{2,i_1+1}^{(r+1)} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \cdots & w_{m_{r+1}-1,i_2+1}^{(r+1)} & \cdots & w_{m_{r+1}-1,i_1+1}^{(r+1)} & \cdots \\ \cdots & w_{m_{r+1},i_2+1}^{(r+1)} & \cdots & w_{m_{r+1},i_1+1}^{(r+1)} & \cdots \end{bmatrix} \dots \quad (3.96)$$

- If the activation function is odd, i.e., $h(-u) = -h(u)$, like e.g., $\tanh(\cdot)$, then $f_n(\cdot)$ remains unaffected by a change of the sign on all weights into and from a specific neuron. Regard the i 'th neuron in the r 'th layer, then

$$\mathbf{w}_i^{(r)} \leftrightarrow -\mathbf{w}_i^{(r)} \quad (3.97)$$

and

$$w_{ji}^{(r+1)} \leftrightarrow -w_{ji}^{(r+1)}, \quad j \in [1; m_{r+1}] \quad (3.98)$$

in order to ensure that $f_n(\cdot)$ is unaffected. The result follows immediately from `nf:perceproc` and (3.86).

The ambiguousness of the parameterization of $f_n(\cdot)$ is not in general a drawback. One may expect that irrespective of an arbitrary chosen point in the m -dimensional space spanned by the weight vector, \mathbf{w} , the weight vectors which minimize the performance criterion – the optimal weight vectors – will always be fairly close to this point. This fact implies that the initial guess of an weight estimation algorithm may be fairly uncritical. However, if the optimal weight vectors are dense the applied weight estimation algorithm may converge to a poor solution¹⁷.

Degeneration is another property of the parameterization. $f_n(\mathbf{z}(k); \mathbf{w})$ is said to be degenerated if there exists one or more restrictions on the weights which have the general form:

$$\xi(\mathbf{w}) = 0 \quad (3.99)$$

where $\xi(\cdot)$ is a scalar *restriction function*. If $\xi(\mathbf{w}) \approx 0$ we shall call the degeneration a quasi degeneration. Below is listed a number of cases where degeneration occurs:

- Consider a Taylor series expansion of the activation function, i.e., the hyperbolic tangent (e.g., [Spiegel 68, p. 112]):

$$\tanh(u) = u - \frac{u^3}{3} + \frac{2u^5}{15} - \dots, \quad |u| < \frac{\pi}{2}. \quad (3.100)$$

If u is appropriately close to zero then: $\tanh(u) \approx u$. The neuron thus effectively acts as a linear neuron. This induces several degenerations.

Suppose (for the sake of convenience) that the first neuron in the r 'th layer provides a linear output $u_1^{(r)}(k)$ which is virtually zero so that the above linear approximation can be used. The linear output of the i 'th neuron in the $(r+1)$ 'th layer then becomes:

$$\begin{aligned} u_i^{(r+1)} &= w_{i,0}^{(r+1)} + \sum_{j=1}^{m_r} w_{i,j}^{(r+1)} s_j^{(r)}(k) \\ &= w_{i,1}^{(r+1)} \left(\mathbf{w}_1^{(r)} \right)^\top \mathbf{s}^{(r-1)}(k) + w_{i,0}^{(r+1)} + \sum_{j=2}^{m_r} w_{i,j}^{(r+1)} s_j^{(r)}(k). \end{aligned} \quad (3.101)$$

The contributions to the neurons in the $(r+1)$ 'th layer from neuron 1 in layer r are thus scaled versions (multiplication with $w_{i,1}^{(r+1)}$) of a linear combination (given by $\mathbf{w}_1^{(r)}$) of the signals $s_j^{(r-1)}(k)$. This induces one restriction on the weights of the form:

$$w_{i,1}^{(r+1)} \mathbf{w}_1^{(r)} = \mathbf{w}' \quad (3.102)$$

where \mathbf{w}' is an arbitrary vector with dimension $m_{r-1} + 1$ and i_1 denotes one specific $i \in [1; m_{r+1}]$. Hence we face a single quasi degeneration which can be circumvented e.g., by setting $w_{i_1,1}^{(r+1)} \equiv 1$.

¹⁷Consider as an example the permutation of the two optimal weights, w_1^* , w_2^* , i.e., both $[w_1^*, w_2^*]$ and $[w_2^*, w_1^*]$ are optimal weight vectors. If the optimal weights are close to the line $w_1 = w_2$ in the $[w_1, w_2]$ -space then the weight estimation algorithm on the average may converge to the intermediate point $[(w_1 + w_2)/2, (w_1 + w_2)/2]$ which possibly results in poor performance.

Another degeneration arises if two or more neurons in the $(l-1)$ 'th have linear output responses (i.e., the u -signals) close to zero. Imagine that the two first neurons in the $(l-1)$ 'th are effectively linear then the filter output $\hat{y}(k)$ becomes:

$$\begin{aligned}
\hat{y}(k) &= w_{10}^{(l)} + \sum_{j=1}^{m_{l-1}} w_{1j}^{(l)} s_j^{(l-1)}(k) \\
&= w_{11}^{(l)} \left(\mathbf{w}_1^{(l-1)} \right)^\top \mathbf{s}^{(l-2)} + w_{12}^{(l)} \left(\mathbf{w}_2^{(l-1)} \right)^\top \mathbf{s}^{(l-2)} + w_{10}^{(l)} + \sum_{j=3}^{m_{l-1}} w_{1j}^{(l)} s_j^{(l-1)}(k) \\
&= \left(w_{11}^{(l)} \left(\mathbf{w}_1^{(l-1)} \right)^\top + w_{12}^{(l)} \left(\mathbf{w}_2^{(l-1)} \right)^\top \right) \mathbf{s}^{(l-2)} + w_{10}^{(l)} + \sum_{j=3}^{m_{l-1}} w_{1j}^{(l)} s_j^{(l-1)}(k). \quad (3.103)
\end{aligned}$$

It is obvious that the first parenthesis can be substituted by a new arbitrary vector \mathbf{w}' with dimension $m_{l-2} + 1$. As the parenthesis contains $2 + 2(m_{l-2} + 1)$ weights we thus make $m_{l-2} + 3$ restrictions leading to a multiple quasi degeneration. To avoid the degenerations we could e.g., remove one of the neurons and fix the weight from the remaining neuron to the output neuron at one.

- Existence of a linear dependence among the variables in the input vector, $\mathbf{z}(k)$, implies a multiple degeneration. Say, for instance, that $z_1(k) \propto z_2(k)$. Then only the sum of the second and the third columns in the weight matrix $\mathbf{W}^{(1)}$ can be determined, i.e., we get the following restrictions:

$$w_{i,1}^{(1)} + w_{i,2}^{(1)} = w'_i, \quad \forall i \in [1; m_1] \quad (3.104)$$

where w'_i are some arbitrary weights.

- If the inputs and the weights associated with a neuron are of such nature that the output is saturated, i.e., it is almost equal to a DC with amplitude K_∞ or $K_{-\infty}$. In this case it is not possible to distinguish the output of the neuron from the bias term (the first component of the state vector, $\mathbf{s}(k)$) equal to one. Let the output of the neuron be $s_j^{(r)}(k)$. Then the restriction in the weight-space becomes:

$$w_{i,0}^{(r+1)} + w_{i,j}^{(r+1)} = w'_i, \quad \forall i \in [1; m_{r+1}] \quad (3.105)$$

where w'_i are arbitrary weights. The quasi degenerations can be avoided by simply removing the concerned neuron.

- If the associated weights of two neurons are fairly equal, e.g., $\mathbf{w}_{j_1}^{(r)} \approx \mathbf{w}_{j_2}^{(r)}$, then the outputs of the neurons will be fairly equal. In consequence, we will face the following restrictions:

$$w_{i,j_1}^{(r+1)} + w_{i,j_2}^{(r+1)} = w'_i, \quad \forall i \in [1; m_{r+1}] \quad (3.106)$$

where w'_i are arbitrary weights. The quasi degenerations can be abolished by removing one of the neurons in question.

Degeneration is in general an obstacle which has to be circumvented in order to ensure that the employed algorithm for weight estimation works and furthermore to ensure a high filter performance. These topics are further treated in Ch. 5 and Ch. 6.

The neural networks employed in the simulation study Ch. 8 are restricted to be 2-layer MFPNN's. The reasons for treating 2-layer MFPNN's only are:

- A 2-layer MFPNN has the capability of universal approximation. This is treated in the next paragraph.
- When using a 2-layer network the only architecture parameter one has to determine initially is the number of neurons in the hidden layer, m_1 (provided that the number of input signals, $p = \dim(\mathbf{z}(k))$, is determined a priori). This may ease the search for a proper architecture.
- Adding layers to the network will slow down the convergence of the employed weight estimation algorithm. This behavior can be explained by qualitative arguments. When using a single-layer network – corresponding to a single linear neuron – the contribution to the filter output, $\hat{y}(k)$, associated with a specific weight depends on the concerned weight only; it is not influenced by the remaining weights. On the other hand, within a multi-layer network the contribution associated with a specific weight depends on weights in the preceding layers. Consider for instance the i 'th weight in the l 'th layer which contributes with $w_{1i}^{(l)} \cdot s_i^{(l-1)}(k)$. Clearly this contribution depends on a number of weights in the preceding layers via $s_i^{(l-1)}(k)$. Then the correction of a possible error on the filter output by adjusting $w_{1i}^{(l)}$ is highly dependent on values of a number of other weights.

However, in some applications one may argue in favor of using more than two layers.

For instance when dealing with classification tasks¹⁸ one may regard the internal state vectors, $\mathbf{s}^{(r)}$, $r = 1, 2, \dots$, as a sequence of more and more compact features, i.e., the hidden layers provide a successive feature extraction. Finally, the output layer performs the final decision of the class to which the input pattern belongs. An example of employing this procedure for recognition of hand-written digits is summarized in [Hertz et al. 91, pp. 139–141].

Further, theoretical considerations have established that under certain conditions a 3-layer network is more powerful than a 2-layer network [Obradovic & Yan 90].

Universal Approximation Capabilities In this paragraph we address the universal approximation capabilities of MFPNN's. Most of the theorems regarding universal approximation are existence theorems, i.e., they do not provide any statements concerning the choice of the number of neurons in the hidden layers, the inter-connectivity, etc.. However, the theorems still consolidate the use of neural networks so that only mild conditions on the nonlinear system under consideration are necessary. In recent years many authors have dealt with formulation of universal approximation theorems, see e.g., [Cotter 90], [Cybenko 89], [Hornik et al. 90]. The theorems lead to different two- and three-layer architectures which not necessarily possess the same activation function, $h(\cdot)$, for all neurons in the network.

The general result is that a 2-layer network with a single linear output neuron is sufficient for perfectly approximating an arbitrary nonlinear filtering function provided that the number of neurons in the hidden layer, m_1 , reaches infinity. The existing universal approximation theorems concerning 2-layer networks differ w.r.t. assumptions concerning the nonlinear filtering function, $f_n(\cdot)$ and the activation function, $h(\cdot)$. The less restrictive theorems – which the author know of – are given in [Hornik 91].

¹⁸That is, the task is to classify the input vectors into a number of classes. Typically the network has one sigmoidal output neuron for each class. If the current input vector belongs to, say class number one, then the first output neuron is on, i.e., it equals 1; the other neurons are off, i.e., they equal -1 .

Let $\|\varphi(\mathbf{z})\|_{s,\mu}$ be the s -norm of the function $\varphi(\mathbf{z})$ w.r.t. a finite measure¹⁹, μ , on \mathbb{R}^p , i.e.,

$$\|\varphi(\mathbf{z})\|_{s,\mu} = \left(\int_{\mathbb{R}^p} |\varphi(\mathbf{z})|^s \mu(\mathbf{z}) d\mathbf{z} \right)^{\frac{1}{s}}. \quad (3.107)$$

A typical measure in the context of stochastic signals is: $\mu(\mathbf{z}) = p(\mathbf{z})$, where $p(\cdot)$ is the p.d.f. of \mathbf{z} . Further, often $s = 2$ is employed. In this case: $\|\varphi(\mathbf{z})\|_{2,p} = (E\{\varphi^2(\mathbf{z})\})^{1/2}$.

Consider $g_n(\mathbf{z}(k))$ to be the nonlinearity of the system which the filter have to model and assume that $\|g_n(\mathbf{z}(k))\|_{s,\mu} < \infty$. The performance of the 2-layer neural network with m_1 hidden neurons is expressed in terms of the above norm as follows:

$$\text{dist}(f_n, g_n) = \|f_n(\mathbf{z}; \mathbf{w}) - g_n(\mathbf{z})\|_{s,\mu} \quad (3.108)$$

where $f_n(\cdot)$ is given by `nf:mlnn` with $l = 2$. These prerequisites enable us to restate [Hornik 91, Theorem 1,2]:

THEOREM 3.1 (HORNİK) *If the activation function $h(u)$ is bounded and non-constant then*

$$\forall \epsilon > 0, \exists m_1 \geq 1, \mathbf{w} \in \mathbb{R}^m : \text{dist}(f_n, g_n) \leq \epsilon. \quad (3.109)$$

THEOREM 3.2 (HORNİK)²⁰ *Suppose that the input domain, $\mathcal{D} \subset \mathbb{R}^p$, is compact and that $g_n(\cdot)$ is continuous on \mathcal{D} . If the activation function $h(u)$ is continuous, bounded and non-constant then:*

$$\forall \epsilon > 0, \exists m_1 \geq 1, \mathbf{w} \in \mathbb{R}^m : \sup_{\mathcal{D}} |f_n(\mathbf{z}; \mathbf{w}) - g_n(\mathbf{z})| \leq \epsilon. \quad (3.110)$$

Note that m_1 possibly has to be infinite in order to satisfy the inequalities when ϵ is small.

Furthermore [Hornik 91] contains two theorems regarding how well the partial derivatives of $f_n(\cdot)$ w.r.t. \mathbf{z} approximate the partial derivatives of $g_n(\cdot)$. The essence is that if the activation function is bounded, non-constant and d times differentiable then all partial derivatives up to d 'th order of $g_n(\cdot)$ are approximated arbitrarily well by the network provided that the number of hidden neurons, m_1 , is sufficiently large.

Example of Functional Approximation with an MFPNN The discussion of the signal processing and approximation capability of an MFPNN is finalized by an illustrative one dimensional (i.e., $p = 1$) example. The nonlinear system to be modeled is noiseless and given by:

$$y(k) = g(x(k)) \quad (3.111)$$

¹⁹A measure, $\mu(\mathbf{z})$, is said to be finite on \mathbb{R}^p if

$$\int_{\mathbb{R}^p} \mu(\mathbf{z}) d\mathbf{z} < \infty.$$

Note that,

1. A measure per definition is non-negative and not identical zero.
2. We are content with $\mu(\cdot)$ being finite on the input domain, $\mathcal{D} \subseteq \mathbb{R}^p$.

²⁰Note in both theorems that m_1 possibly has to be infinite when $\epsilon = 0$ is required.

where $x(k)$ is an i.i.d.²¹ stochastic signal uniformly distributed over the interval $[-1; 1]$, and (omitting k for simplicity)

$$g(x) = \left[\exp(-20(x - 0.5)^2) - \exp(-10(x + 0.6)^2) \right] \cdot \left[\frac{1}{1 + \exp(4x - 2)} \right] \cdot \left[1 - \frac{1}{1 + \exp(4x + 2)} \right]. \quad (3.112)$$

In Fig. 3.15 the graph of $g(x)$ is shown. The model of the system is a $[1, 8, 1]$ -network with the hyperbolic tangent as activation function, i.e.,

$$\begin{aligned} \hat{y}(k) &= f(x(k); \mathbf{w}) \\ &= w_{10}^{(2)} + \sum_{i=1}^8 w_{1i}^{(2)} \tanh(w_{i,0}^{(1)} + w_{i,1}^{(1)} x(k)). \end{aligned} \quad (3.113)$$

The weights, i.e., \mathbf{w} , are estimated by a LS-criterion, i.e.,

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} S_N(\mathbf{w}) \quad (3.114)$$

where $\hat{\mathbf{w}}$ denotes the estimated weights and $S_N(\mathbf{w})$ is the LS cost function given by

$$S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N (y(k) - \hat{y}(k))^2. \quad (3.115)$$

In this example we used $N = 500$ training examples. A recursive Gauss-Newton algorithm – more precisely the RGNB-algorithm cf. Ch. 5 and Ch. 8 – was used to estimate the weights.

The performance of the filter was estimated by calculating the *relative cross-validation error index*²²:

$$E_c = \sqrt{\frac{\frac{1}{N_c} \sum_{k=1}^{N_c} e^2(k)}{\frac{1}{N_c - 1} \sum_{k=1}^{N_c} [y(k) - \langle y(k) \rangle]^2}} \quad (3.116)$$

where

- N_c is the size of a cross-validation set, i.e., a set of samples $\{x(k); y(k)\}$, $k = 1, 2, \dots, N_c$, which are independent of those contained in the training set. In this particular example: $N_c = 500$.
- $e(k)$ is the error signal, $e(k) = y(k) - \hat{y}(k)$.
- $\langle y(k) \rangle = N_c^{-1} \sum_{k=1}^{N_c} y(k)$.

As the error has zero mean (cf. Ch. 5) E_c measures the standard deviation of the error signal relative to the standard deviation of the model output $y(k)$. $E_c = 1$ corresponds to using the average value as an estimate of $y(k)$, i.e., $\hat{y}(k) \equiv \langle y(k) \rangle$, $\forall k$. If the filter perfectly models the task under consideration then $E_c = 0$. It should be clear that

²¹Independent identically distributed.

²²The method of cross-validation is described in Sec. 6.5.3

perfectly modeling can not occur in the present case. This is due to the fact that $g(x)$ has a shape which is dominated by the Gaussian bell, $\exp(-x^2)$, which not can be modeled perfectly by a finite sum of hyperbolic tangent functions according to the formula:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (3.117)$$

However, the sum of two hyperbolic tangent functions allows for approximating the Gaussian bell fairly well according to the fact that the Gaussian bell is a localized covering function cf. page 56. Fig. 3.16 shows the approximation of the Gaussian bell by the sum of two $\tanh(\cdot)$ functions which is given by:

$$\varphi(x) = \frac{\tanh(x + \nu) - \tanh(x - \nu)}{2 \tanh(\nu)} \quad (3.118)$$

with $\nu = 3.98 \cdot 10^{-8}$. The denominator represents a scaling which ensures that $\varphi(0) = 1$ since $\exp(0) = 1$. ν is estimated so that the mean squared error between $\varphi(x)$ and $\exp(-x^2)$ is minimized²³ within the interval $[-5; 5]$.

The optimization of the MFPNN results in the performance $E_c = 5.52 \cdot 10^{-4}$ with the following weight estimates:

$$\widehat{\mathbf{W}}^{(1)} = \begin{bmatrix} 3.65 & 4.06 & 1.87 & 1.19 & -1.84 & -2.88 & -4.25 & -5.82 \\ 4.94 & 6.65 & 5.39 & 6.19 & 7.26 & 8.05 & 7.41 & 8.37 \end{bmatrix}^\top \quad (3.119)$$

$$\widehat{\mathbf{W}}^{(2)} = [0.0027, -0.218, -0.0793, 0.226, 0.0668, 0.159, 0.177, -0.259, -0.0758]. \quad (3.120)$$

In order to comprehend the nature of approximation capabilities within the 2-layer MF-PNN we first consider the activation functions as hard limiters, i.e.,

$$\begin{aligned} \widehat{y}(k) &= f(x; \mathbf{w}) \\ &= w_{10}^2 + \sum_{i=1}^8 w_{1i}^2 \cdot \text{sgn} \left(w_{i,0}^1 + w_{i,1}^1 x(k) \right). \end{aligned} \quad (3.121)$$

Thus $\widehat{y}(k)$ is a piecewise-constant function of the input $x(k)$ with discontinuity points given by the hyperplanes (in this case points), $\mathcal{H}_i^{(1)}$, $i \in [1; 8]$ associated with neurons in the first layer. These are given by:

$$\mathcal{H}_i^{(1)} : \widehat{w}_{i1}^{(1)} x_i + \widehat{w}_{i0}^{(1)} = 0 \Leftrightarrow x_i = \frac{-\widehat{w}_{i0}^{(1)}}{\widehat{w}_{i1}^{(1)}}. \quad (3.122)$$

Fig. 3.17 depicts the hyperplane locations along with the approximation of $g(x)$ produced by the network when using the hard limiters²⁴. The curve produced by the neural network can be regarded as quantization of $g(x)$. The quantization is characterized by two facts:

²³The minimization is performed by using the “fmin” routine of the software package MATLAB [MATLAB 94]. This routine employs parabolic interpolation according to Brent’s method, see further [Press et al. 88, Ch. 10.1 & 10.2].

²⁴It should be noted that the weights in the first layer are given by $\widehat{\mathbf{W}}^{(1)}$ cf. nf:whatfl while the weights in the second layer are reestimated in order to compensate that the activation functions are replaced by hard limiters.

- It is non-uniform. That is, the widths of the individual quanta are not equal. The width of a quantum is defined as: $|x_i - x_{i-1}|$, $i \in [1; 9]$ where $x_0 = -1$, $x_9 = 1$ and the remaining x_i given by `nf:hypneu`. Furthermore, the resolution, i.e., $|f(x_i; \hat{\mathbf{w}}) - f(x_{i-1}; \hat{\mathbf{w}})|$, is variable.
- It is adapted to the task under consideration, viz. modeling $g(x)$, by estimating the weights of the neural network. Consequently, altering $g(x)$ will result in a change in quantum widths and resolution.

Using the hyperbolic tangent as the activation function results in a smoothing of the piecewise-constant curve. This is in general profitable as the functions which are going to be modeled normally are smooth, i.e., it is expected that fewer hidden neurons are necessary in order to ensure a proper performance.

The contributions from the individual hidden neurons to the filter output yield:

$$\hat{y}_i(k) = w_{1i}^2 \tanh(w_{i,0}^1 + w_{i,1}^1 x(k)), \quad i \in [1; 8], \quad (3.123)$$

and $\hat{y}_0(k) = w_{10}^{(2)}$. In Fig. 3.18 these contributions are shown. The filter output, $\hat{y}(k)$, results by adding all these terms.

3.2.3 Gram-Schmidt Neural Networks

In this section a modification of the standard MFPNN is treated. The object is to speed-up the convergence of the weight estimation algorithm which often is a major drawback when using the back-propagation algorithm (BP-algorithm) cf. Ch. 5. Recall from Ch. 5 that the condition number (eigenvalue spread) of the Hessian matrix (second derivative of the cost function w.r.t. the weights) determines the speed of convergence. A straight forward way to speed up convergence thus to employ a second order optimization method (a Newton scheme), cf. Ch. 5. In [Orfanidis 90] a pseudo second order method can be accomplished by introducing a layer of linear neurons between every layer in the network. The purpose is to transform the state vector, $\mathbf{s}^{(r)}(k)$, into a modified state vector of uncorrelated components with equal variances. This ensures that the eigenvalue spread of the covariance matrix associated with the modified state vector equals one. This recipe is in keeping with techniques known from linear adaptive filters [Marshall et al. 89]. However, the suggested modification is suboptimal since it does not guarantee that condition number of the Hessian decreases. Only when the filter is an LX-filter the above choice is optimal (see Ch. 5 for further details).

Even though the modification possibly leads to improved convergence and better numerical properties it should be emphasized that the approximation capabilities of the filter architecture are not altered cf. Sec. 6.3.6.

The processing of the linear layers is simply expressed in terms of a matrix transformation, i.e.,

$$\boldsymbol{\zeta}^{(r)}(k) = \mathbf{Q} \mathbf{s}^{(r)}(k) \quad (3.124)$$

where

- $\boldsymbol{\zeta}^{(r)}(k)$ is the $(m_r + 1)$ -dimensional modified output vector signal of the r 'th layer.

- \mathbf{Q} is the transformation matrix:

$$\mathbf{Q} = \mathbf{\Sigma}^{-1} \left(\mathbf{U}^{(r)} \right)^\top \quad (3.125)$$

with $\left(\mathbf{U}^{(r)} \right)^\top$ being the $(m_r + 1) \times (m_r + 1)$ unit lower triangular matrix,

$$\left(\mathbf{U}^{(r)} \right)^\top = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ u_{10}^{(r)} & 1 & 0 & \cdots & 0 & 0 \\ u_{20}^{(r)} & u_{21}^{(r)} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m_r,0}^{(r)} & u_{m_r,1}^{(r)} & u_{m_r,2}^{(r)} & \cdots & u_{m_r,m_r-1}^{(r)} & 1 \end{bmatrix} \quad (3.126)$$

which constitutes a Gram-Schmidt orthogonalization, and $\mathbf{\Sigma}$ is the diagonal matrix

$$\mathbf{\Sigma} = \text{diag} \left\{ \left[\sigma_1^2, \sigma_2^2, \cdots, \sigma_{m_r+1}^2 \right]^\top \right\}. \quad (3.127)$$

Here the following definitions are made:²⁵

$$\boldsymbol{\varphi}(k) = \left(\mathbf{U}^{(r)} \right)^{-\top} \mathbf{s}^{(r)}(k), \quad (3.128)$$

$$\sigma_i^2 = E \left\{ \varphi_i^2 \right\}. \quad (3.129)$$

In Fig. 3.19 the modification of the MFPNN is depicted. $\boldsymbol{\varphi}(k)$ implies that $\mathbf{s}^{(r)}(k) = \left(\mathbf{U}^{(r)} \right)^\top \boldsymbol{\varphi}(k)$. Accordingly²⁶:

$$\begin{aligned} \varphi_1 &= s_1 - u_{10} \\ \varphi_2 &= s_2 - u_{20} - u_{21}\varphi_1 \\ &\vdots \\ \varphi_{m_r} &= s_{m_r} - u_{m_r,0} - u_{m_r,1}\varphi_1 - \cdots - u_{m_r,m_r-1}\varphi_{m_r-1}. \end{aligned} \quad (3.130)$$

No correlation among the φ_i signals – which corresponds to $E\{\boldsymbol{\varphi}\boldsymbol{\varphi}^\top\}$ being a diagonal matrix – is ensured if a particular φ_i is uncorrelated with all the previous, i.e., φ_{i-j} , $j = 1, 2, \cdots, i-1$ and $\{\varphi_i\} = 0$. The latter requirement ensures that φ_i becomes uncorrelated with the unit-signal. Consider the expression for φ_i which cf. `nf:zetaeqns` is given by:

$$\varphi_i = s_i - \left(u_{i0} + \sum_{j=1}^{i-1} u_{ij}\varphi_j \right). \quad (3.131)$$

φ_i is now regarded as the error done when using a linear combination of the variables $1, \varphi_1, \varphi_2, \cdots, \varphi_i$ in order to explain s_i . Minimization of the mean squared error, i.e., $\sigma_i^2 = E\{\varphi_i^2\}$ w.r.t. $\mathbf{u}_i = [u_{i0}, u_{i1}, \cdots, u_{i,i-1}]^\top$ is equivalent to determining \mathbf{u}_i so that

$$E\{\varphi_i\varphi_j\} = 0, \quad j = 1, 2, \cdots, i-1 \quad (3.132)$$

²⁵Recall that $^{-\top}$ denotes the transposed inverse.

²⁶The time k and the layer index $^{(r)}$ is omitted for simplicity, and recall that the first component in both $\mathbf{s}^{(r)}$ and $\boldsymbol{\zeta}^{(r)}$ equals one.

according to the *principle of orthogonality* [Haykin 91, Ch. 3.6] (see also Ch. 5). Consequently, in order to obtain uncorrelated φ_i signals successive minimization tasks are accomplished, i.e.,

$$\hat{\mathbf{u}}_i = \arg \min_{\mathbf{u}_i} E\{\varphi_i^2\}, \quad i = 1, 2, \dots, m_r. \quad (3.133)$$

In [Orfanidis 90] a BP-algorithm for estimating the matrices $\mathbf{U}^{(r)}$, along the weight matrices $\mathbf{W}^{(r)}$ are developed. The algorithm was tested on a 3-input parity problem which showed a speed-up in the necessary number of iterations with a factor of 3–4 in contrast to employing the conventional MFPNN.

3.2.4 Canonical Piecewise-Linear Filters

A canonical piecewise-linear representation of a multidimensional function was introduced in [Kang & Chua 78]. Regarded as an XN-filter the filtering in the piecewise-linear filter (PWLF) yields:

$$\begin{aligned} \hat{y}(k) &= f_n(\mathbf{z}(k); \mathbf{w}) \\ &= \beta_{10} + \tilde{\boldsymbol{\beta}}_1^\top \mathbf{z}(k) + \sum_{i=2}^q \alpha_i \left| \beta_{i0} + \tilde{\boldsymbol{\beta}}_i^\top \mathbf{z}(k) \right| \end{aligned} \quad (3.134)$$

where

$$\begin{aligned} \tilde{\boldsymbol{\beta}}_i &= [\beta_{i1}, \beta_{i2}, \dots, \beta_{ip}]^\top, \\ \boldsymbol{\beta}_i &= [\beta_{i0}, \tilde{\boldsymbol{\beta}}_i^\top]^\top, \text{ and} \\ \mathbf{w} &= [\boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top, \dots, \boldsymbol{\beta}_q^\top, \alpha_2, \dots, \alpha_q]^\top. \end{aligned}$$

That is, $m = \dim(\mathbf{w}) = q(p+2) + 1$. According to `nf:nfcan` the canonical filter representation immediately becomes:

$$\hat{y}(k) = \alpha_0 + \sum_{i=1}^q \alpha_i b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) \quad (3.135)$$

with $\alpha_0 = 0$, $\alpha_1 = 1$,

$$b_1(\mathbf{z}(k); \boldsymbol{\beta}_1) = \boldsymbol{\beta}_1^\top [1, \mathbf{z}^\top(k)]^\top \quad (3.136)$$

and

$$b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) = \left| \boldsymbol{\beta}_i^\top [1, \mathbf{z}^\top(k)]^\top \right|, \quad i = 2, 3, \dots, q. \quad (3.137)$$

The PWLF in `nf:pwl` is capable of representing any continuous piecewise-linear non-linear filtering function, $f_n(\cdot)$, with a nondegenerate linear partition²⁷ [Chua & Ying 83]. The discontinuous and degenerated cases can be treated as well [Kahlert & Chua 90],

²⁷The linear partition is defined as the set of hyperplanes, $\{\mathcal{H}_i\}$, where

$$\mathcal{H}_i : \beta_{i0} + \tilde{\boldsymbol{\beta}}_i^\top \mathbf{z} = 0, \quad i = 2, 3, \dots, q.$$

The linear partition is said to be nondegenerate if the dimension of all intersections among the $q-1$ hyperplanes is less than or equal to $p-j$, where j is the number of intersecting hyperplanes. If for example $p=2$ the hyperplanes become straight lines. If three of these straight lines intersect in a common point the linear partition is degenerated.

[Kang & Chua 78] by introducing some additional terms. However, these subjects will be ignored in this description.

Comparing the PWLF with the canonical filter representation of a MFPNN (see page 55) we observe that the PWLF can be interpreted as a 2-layer MFPNN where the first neuron in the hidden layer is linear (corresponding to the term $\beta_1^\top [1, \mathbf{z}(k)^\top]^\top$) while the other neurons are perceptrons with the activation function: $h(u) = |u|$. Note that the activation is not sigmoidal; nor is it bounded. Consequently, universal approximation can not be guaranteed by using Th. 3.1. Furthermore, the bias weight of the linear output neuron is equal to zero and the weight from the linear neuron in the hidden layer to the output neuron is equal to one. In Fig. 3.20 the PWLF is shown. Another view of the PWLF appears by using the identity:

$$|u| = u \cdot \text{sgn}(u). \quad (3.138)$$

nf:pwlf now becomes

$$\begin{aligned} \hat{y}(k) &= \beta_{10} + \tilde{\beta}_1^\top \mathbf{z}(k) + \sum_{i=2}^q \alpha_i \left(\beta_{i0} + \tilde{\beta}_i^\top \mathbf{z}(k) \right) \text{sgn} \left(\beta_{i0} + \tilde{\beta}_i^\top \mathbf{z}(k) \right) \\ &= \vartheta_1 + \sum_{i=2}^q \vartheta_i \text{sgn} \left(\beta_{i0} + \tilde{\beta}_i^\top \mathbf{z}(k) \right) \end{aligned} \quad (3.139)$$

where we made the definitions:

$$\vartheta_1 = \beta_{10} + \tilde{\beta}_1^\top \mathbf{z}(k) \quad (3.140)$$

and

$$\vartheta_i = \alpha_i \left(\beta_{i0} + \tilde{\beta}_i^\top \mathbf{z}(k) \right), \quad i \in [2; q]. \quad (3.141)$$

The structure of nf:plwfn is indeed a 2-layer MFPNN with hard limiting activation functions. β_i , $i \in [2; q]$ are the weights in the first layer associated with the $i - 1$ 'st neuron and ϑ_i , $i \in [1; q]$ are the “weights” of the second layer. However, these “weights” are not constant but linearly dependent on $\mathbf{z}(k)$. When using a 2-layer MFPNN recollect that the input-output surface (with $h(u) = \text{sgn}(u)$) consists of horizontal²⁸ p -dimensional hyperplanes (see e.g., Fig. 3.17) which are delimited by the hyperplanes, $\mathcal{H}_i^{(1)}$, $i \in [1; q]$, cf. nf:hypmfn. On the other hand, dealing with PWLF's the hyperplanes in the input-output space are not horizontal but may have any orientation which is specified by ϑ_i . An example of an input-output surface is shown in Fig. 3.21. Considering nf:plwfn an obvious extension of the PWLF is to replace the signum function by a smooth function – e.g., the hyperbolic tangent. This may result in some advantages:

- The input-output surface becomes smooth.
- The required number of hidden neurons, i.e., q , may become smaller.
- It is possible to derive an algorithm, e.g., a BP-algorithm (see Ch. 5), for estimating the parameters.

²⁸By that is meant that the hyperplanes are parallel to the p -dimensional space spanned by \mathbf{z} .

In [Lin & Unbehauen 90] the idea of using a piecewise-linear representation within nonlinear filtering was proposed. The shape of the basis function was slightly changed to comply with:

$$b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) = \left| \tilde{\boldsymbol{\beta}}_i^\top \mathbf{z}(k)^\top - 1 \right| - \left| \tilde{\boldsymbol{\beta}}_i^\top \mathbf{z}(k)^\top + 1 \right|, \quad i = 2, 3, \dots, q. \quad (3.142)$$

Now let:

$$\zeta_i \triangleq \frac{\tilde{\boldsymbol{\beta}}_i^\top \mathbf{z}^\top}{|\tilde{\boldsymbol{\beta}}_i|}. \quad (3.143)$$

In Fig. 3.22 it is shown that basis functions possess sigmoidal shapes. An LMS (Least Mean Squares) based stochastic gradient algorithm (see further Ch. 5) was developed in [Lin & Unbehauen 90]. In addition, bounds on the step-sizes of the algorithm which ensure convergence, were provided. The filter is favorable as regards implementation in either soft- or hardware since the only nonlinear operations involved are the absolute value and the signum function. Three examples of identifying nonlinear systems (see p. 7) were presented and compared to using Volterra and linear filters (i.e., LL-filters). The general result is that the PWLF is superior to both the linear and the Volterra filter w.r.t. speed of convergence and mean square error. Furthermore, the computational burden and the memory requirements involved with the PWLF are less than those involved with the Volterra filter approach.

3.2.5 Semilocal Units

The semilocal unit approach [Hartmann & Keeler 91] consists in an MFNN with nonlinear neurons which are not of the perceptron type. The signal processing within the i 'th semilocal neuron (unit) in the r 'th layer yields:

$$s_i^{(r)}(k) = \sum_{j=1}^{m_r-1} w_{i,j}^{(r)} \exp \left(-\frac{1}{2} \left[\frac{s_j^{(r-1)}(k) - \nu_{ij}^{(r)}}{\eta_{ij}^{(r)}} \right]^2 \right) \quad (3.144)$$

where $\nu_{ij}^{(r)}$ and $\eta_{ij}^{(r)}$ are the position and scale parameters of the i 'th neuron in the r 'th layer w.r.t. the j 'th input signal, $s_j^{(r)}$, respectively. $w_{i,j}^{(r)}$ is the weight with which the j 'th input signal contribute to the output. A 2-layer XN-filter then becomes:

$$\hat{y}(k) = w_{10}^{(2)} + \sum_{i=1}^q \sum_{j=1}^p w_{i,j}^{(1)} \exp \left(-\frac{1}{2} \left[\frac{z_j(k) - \nu_{ij}^{(1)}}{\eta_{ij}^{(1)}} \right]^2 \right). \quad (3.145)$$

Notice that all weights of the linear output neuron except the bias weight, $w_{10}^{(2)}$, are equal to one (i.e., they appear indirectly) in order to prevent a multiple degeneration (see p. 58). The number of adjustable parameters, m , in the 2-layer network equals $m = 3pq + 1$ in contrast to $m = q(p + 2) + 1$ in the conventional MFPNN. That is, roughly three times as many parameters have to be estimated. In [Hartmann & Keeler 91] a BP-algorithm for estimating the parameters was suggested.

In Fig. 3.23 below a typical response of a semilocal neuron with two inputs is shown. In contrast to the conventional MFPNN a localization is embedded in the semilocal neurons since the output of the neuron is composed of Gaussian bell-shape functions which form

a localization. However, the neural network based semilocal neurons do not perform a local approximation; merely a semilocal approximation. This is due to the following facts: First notice that the output (see `nf:semiuni`) is the weighted sum of Gaussian bell functions which separately depend on a single input signal only. When a particular input, say $s_j^{(r-1)}$, becomes large relative to the width of the Gaussian bell (specified by the $\eta_{ij}^{(r)}$ parameter) the localization in the $s_j^{(r-1)}$ -direction is probably absent since the Gaussian bell functions in the remaining directions still may contribute significantly. This phenomenon is evidently illustrated in Fig. 3.23.

In [Hartmann & Keeler 91] capabilities of the semilocal MFNN were tested by predicting the chaotic time series generated by the Mackey-Glass equation (see Ch. 8). Simulations²⁹ showed that the computational complexity required in order to predict the time series 85 samples ahead (i.e., predicting $y(k + 85)$) with a relative cross-validation error index, E_c , (cf. `nf:rcvei`) at 0.08 was reduced by a factor of 40 when using a semilocal MFNN rather than a conventional MFPNN. Furthermore, the semilocal MFNN was compared to a localized receptive field network which is treated in Sec. 3.3.1. These simulations showed that the semilocal MFNN used twice as much computational power to reach $E_c = 0.08$.

3.2.6 Projection Pursuit

The structure of a 2-layer MFPNN (with linear output neuron) resembles the Projection Pursuit technique³⁰ [Friedman & Stuetzle 81], [Huber 85] known from the statistic literature. The Projection Pursuit filter is given by:

$$\hat{y}(k) = \sum_{i=1}^q h_i(\mathbf{w}_i^\top \mathbf{z}) \quad (3.146)$$

where \mathbf{w}_i are some weights and $h_i(\cdot)$ are the activation functions which are required to be smooth functions and not necessarily identical. In fact, the main difference of the Projection Pursuit filter and the 2-layer MFPNN consists in how the weights and the activation functions are adjusted; however, this will not be further considered. The idea guiding Projection Pursuit is that the input-output surface can be constructed properly by a sum of low-dimensional projections in the p -dimensional input space. Note, that the contribution (from the i 'th neuron), $h_i(\mathbf{w}_i^\top \mathbf{z})$, indeed constitutes a projection onto a one-dimensional subspace.

Further discussion on Projection Pursuit can be found in [Cherkassky 92], [Friedman 91], [Friedman & Stuetzle 81] and [Huber 85]. Here we merely highlight some of the stated advantages and drawbacks:

- Even for a small number of hidden units (q small) many classes of functions can be approximated closely.
- The estimated filter is invariant to any nonsingular linear transformation of the input vector signal. That is, if the input vector is transformed according to: $\mathbf{z} \leftarrow \mathbf{Q}\mathbf{z}$, where \mathbf{Q} is a nonsingular matrix then the estimated weight vectors are transformed according to: $\hat{\mathbf{w}}_i \leftarrow \mathbf{Q}^{-\top} \hat{\mathbf{w}}_i$ and the activation functions, $h_i(\cdot)$, are maintained.
- If q is relatively small it may be possible to interpret the functionality of the filter by inspecting the individual functions $h_i(\cdot)$.

- There exist simple functions which require a large number (possibly an infinite number) of hidden units in order to ensure a close approximation. An example is the function, $g_n(\mathbf{z}) = \exp(z_1 z_2)$, which requires $q \rightarrow \infty$ to get a perfect approximation irrespective of the choice of $h_i(\cdot)$ [Huber 85, p. 454].

3.2.7 Neural Network with FIR/IIR Synapses

In order to accomplish signal processing tasks [Back & Tsoi 91] suggests to use an MFNN which incorporate memory within the neurons. The processing of the i 'th nonlinear perceptron in the r 'th layer, cf. `nf:percepproc`, is replaced by:

$$u_i^{(r)}(k) = w_{i,0}^{(r)} + \sum_{j=1}^{m_{r-1}} s_j^{(r-1)}(k) * \omega_{ij}^{(r)}(k), \quad (3.147)$$

$$s_i^{(r)}(k) = h(u_i^{(r)}(k)) \quad (3.148)$$

where $\omega_{ij}^{(r)}(k)$ are impulse responses of linear filters and $*$ denotes convolution. A neuron of this type is evidently called a perceptron with memory. Note that the convolution in `nf:filin` substitutes the product $s_j^{(r-1)}(k)w_{i,j}^{(r)}$ within the perceptron. The remaining processing stay unchanged. The impulse responses may be finite or infinite by letting the corresponding transfer functions, $\Omega_{ij}^{(r)}(z)$, (the z -transforms of $\omega_{ij}^{(r)}(k)$) be rational polynomials in z^{-1} .

The processing within the perceptron with memory is given by `nf:filin` and (3.148) is shown in Fig. 3.24. The use of infinite impulse responses filters (IIR-filters) open up the prospect for dealing with XN-filters which are recurrent, i.e., obeying:

$$\hat{y}(k) = f(x(k), x(k-1), \dots, x(k-L+1), \hat{y}(k-1), \hat{y}(k-2), \dots, \hat{y}(k-P)). \quad (3.149)$$

However, an MFNN consisting of perceptrons with memory does not permit all possible filters of the form `nf:recur` to be implemented. This is due to the fact that only linear feed-back is present in the neural network while nonlinear feed-back obviously may occur in the filter `nf:recur`. The restriction to considering only linear feed-back is beneficial with respect to controlling the behavior of the neural network filter. If the poles of all filters, $\Omega_{ij}^{(r)}(z)$, remain inside the unit circle then the overall filtering will stay stable. Furthermore, since only linear feed-back is present phenomena such as limit cycles and chaos will not be possible.

In [Back & Tsoi 91] a BP-algorithm for adaptation of weights and impulse responses were developed; however, no comparison with a conventional MFPNN was provided. In consequence, the issue concerning the circumstances under which the architecture may show to be advantageous still remains open.

3.3 Local Approximation

3.3.1 Localized Receptive Field Networks

The so-called Localized Receptive Field networks – or Radial Basis Function networks [Moody & Darken 88], [Moody & Darken 89], [Niranjan & Kardirkamanathan 91],

²⁹The training and the cross-validation sets consisted of 500 samples each.

³⁰This was also pointed out in [Cherkassky 92].

[Platt 91], [Sanger 91], [Stokbro et al. 90] – fall into the class of localized covering function filters cf. Sec. 3.1.2.1. The canonical filter representation yields in this case³¹:

$$\hat{y}(k) = \sum_{i=1}^q b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) \cdot D_i(\mathbf{z}(k)) \quad (3.150)$$

where

- The basis functions are normally chosen to be constants (e.g., [Moody & Darken 88], [Moody & Darken 89]), i.e.,

$$b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) = \beta_{i0}. \quad (3.151)$$

This filter is denoted a zero-order covering function filter. Also linear basis functions (first-order filters) have been suggested [Stokbro et al. 90], i.e.,

$$b_i(\mathbf{z}(k); \boldsymbol{\beta}_i) = [1; \mathbf{z}^\top(k)] \boldsymbol{\beta}_i \quad (3.152)$$

where $\boldsymbol{\beta}_i$ is a $p + 1$ -dimensional vector. In principle, any kind of basis function may be employed; however, there will be a trade off between q and the complexity of the basis functions in order to avoid the “curse of dimensionality” which may obstruct a proper estimation from the available training data.

- The domain functions – i.e., the localized covering functions – obey in general:

$$D_i(\mathbf{z}(k)) = \varphi \left[(\mathbf{z}(k) - \mathbf{z}_i^o)^\top \boldsymbol{\Sigma}_i^{-1} (\mathbf{z}(k) - \mathbf{z}_i^o) \right] \quad (3.153)$$

where

- $\varphi(\cdot) : \mathbb{R}_+ \cup \{0\} \mapsto]0; 1]$ is a nonlinear monotonously decreasing function with $\varphi(0) = 1$.
- \mathbf{z}_i^o is the *center vector* of the i 'th covering function.
- $\boldsymbol{\Sigma}_i$ is a positive definite scaling matrix which defines the effective area of the subset $\mathcal{D}_i \subset \mathcal{D}$, where \mathcal{D} is the input domain.

Normally, a Gaussian bell-shaped function is employed, and furthermore the individual directions of the input space are uncoupled [Moody & Darken 89], [Niranjan & Kardirkamanathan 91], i.e.,

$$\boldsymbol{\Sigma}_i = \text{diag} \left\{ [\sigma_{i,1}^2, \sigma_{i,2}^2, \dots, \sigma_{i,p}^2] \right\} \quad (3.154)$$

or $\boldsymbol{\Sigma}_i = \sigma_i^2 \mathbf{I}$. In the latter case:

$$D_i(\mathbf{z}(k)) = \exp \left(-\frac{|\mathbf{z}(k) - \mathbf{z}_i^o|^2}{\sigma_i^2} \right) \quad (3.155)$$

where σ_i^2 is the width of the radial symmetric domain function.

In Fig. 3.25 a simple zero-order localized covering function filter is shown. The capabilities of zero-order localized covering function filters are e.g., demonstrated in terms of universal approximation. In [Hartmann et al. 90] it was shown that by using Gaussian covering functions `mf_gausker` all continuous functions – defined on a compact input domain \mathcal{D} – can be implemented by letting $q \rightarrow \infty$. Furthermore, [Park & Sandberg 91] proved

universal approximation under even weaker conditions: $\varphi(\cdot)$ can be any bounded integrable continuous function with $\int \varphi(t)dt \neq 0$, any norm³² may be used to measure the distance from $\mathbf{z}(k)$ to the centers \mathbf{z}_i^o , the input domain $\mathcal{D} = \mathbb{R}^p$, and finally, $\sigma_i^2 \equiv \sigma^2, \forall i$.

The parameters associated with the zero- or first-order localized covering function filters can be estimated according to different strategies:

- The output is linear in the β_i parameters, i.e., the LS cost function is quadratic with a single global minimum cf. Sec. 5.2. An underlying assumption is that the parameters of the domain functions – e.g., \mathbf{z}_i^o , and σ_i^2 – are predetermined. In [Moody & Darken 88], [Moody & Darken 89] it was suggested to use the k-means clustering algorithm (e.g., [Duda & Hart 73]) for determination of the centers, \mathbf{z}_i^o . This is done by minimizing the cost function:

$$E = \sum_{i=1}^q \sum_{k=1}^N M_{i,k} |\mathbf{z}(k) - \mathbf{z}_i^o|^2 \quad (3.156)$$

where $\mathbf{M} = \{M_{i,k}\}$ is the $q \times N$ (N is the number of training examples) cluster membership matrix which defines which cluster a particular input sample, $\mathbf{z}(k)$ belongs to. That is³³

$$M_{i,k} = \begin{cases} 1 & \text{if } \mathbf{z}(k) \text{ belongs to cluster } i \\ 0 & \text{otherwise} \end{cases} \quad (3.157)$$

In particular, in [Moody & Darken 89] the minimization task is solved by using a simple stochastic approach which consists in:

1. Initially choose q centers, $\mathbf{z}_i^o, i \in [1; q]$, randomly, and select a small step-size $\mu > 0$.
2. **for** $k = 1, 2, \dots, N$

$$i_{\min} = \arg \min_{i \in [1; q]} |\mathbf{z}(k) - \mathbf{z}_i^o|^2$$

$$\text{Update: } \mathbf{z}_{i_{\min}}^o \leftarrow \mu (\mathbf{z}(k) - \mathbf{z}_{i_{\min}}^o)$$

end

What remains is the determination of the widths, σ_i^2 . [Moody & Darken 89] suggests to use a nearest-neighbor heuristics, e.g.,

$$\sigma_i^2 = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} |\mathbf{z}(k) - \mathbf{z}_i^o|^2 \quad (3.158)$$

where \mathcal{I} is the set of indices defining the neighbors of the center \mathbf{z}_i^o , and $|\mathcal{I}|$ is the cardinality, i.e., the number of elements.

- Another strategy is to estimate all parameters (β_i, \mathbf{z}_i^o , and σ_i^2) by minimizing the overall cost function (e.g., the LS cost function) which is non-quadratic and afflicted with local minima. Various first order algorithms have been derived for this purpose, e.g., [Niranjan & Kardirkamanathan 91], [Stokbro et al. 90]. In addition, the general BP-algorithm for layered architectures presented in Sec. 5.3.3 may also be applied in conjunction with a stochastic gradient – or recursive Gauss-Newton – approach, see Ch. 5.

3.3.2 Tree-Structured Piecewise-Linear Filter

The Tree-Structured Piecewise-Linear Filter [Gelfand et al. 91] is a proposal within the class of partition function filters cf. Sec. 3.1.2.1 in which the basis functions are linear (first-order filter). The filter output is accordingly given by:

$$\hat{y}(k) = \sum_{i=1}^q [1, \mathbf{z}^\top(k)] \cdot \boldsymbol{\beta}_i \cdot D_i(\mathbf{z}(k)) \quad (3.159)$$

where $\boldsymbol{\beta}_i = [\beta_{i,0}, \tilde{\boldsymbol{\beta}}_i^\top]^\top$ is the $p+1$ -dimensional parameter vector of the i 'th basis function, $D_i(\mathbf{z}(k))$ is the partition function. The filter is shown in Fig. 3.26. In the present case the partitions are separated by hyperplanes, $\mathcal{H}_{rs} : \mathbf{a}_{rs}^\top \mathbf{z}(k) - t_{rs} = 0$ where \mathbf{a} is the normal vector and t the threshold. In [Gelfand et al. 91] the partition function filter is implemented by successively constructing a tree – consisting of the nodes, $\mathcal{O}_{rs} = \{\mathbf{a}_{rs}, t_{rs}\}$ – until the number of terminal nodes equals a prescribed q . In practice, this is done by combining the Tree Traversing algorithm p. 33 with a minimization of the LS cost function in order to estimate \mathbf{a}_{rs} , t_{rs} , and $\boldsymbol{\beta}_i$. In that context consider the following items:

- Recall that the hyperplane \mathcal{H}_{rs} separates the input space into two sets:

$$\mathcal{H}_{rs}^+ = \left\{ \mathbf{z}(k) : \mathbf{a}_{rs}^\top \mathbf{z}(k) \geq t_{rs} \right\}, \quad (3.160)$$

$$\mathcal{H}_{rs}^- = \left\{ \mathbf{z}(k) : \mathbf{a}_{rs}^\top \mathbf{z}(k) < t_{rs} \right\} \quad (3.161)$$

- We employ a slightly different notation so that $i = 1, 2, \dots, q$ is equivalent to (r, s) running over the set of indices \mathcal{S} defining the current terminal nodes.
- The employed LS cost function is defined as (see further Ch. 5):

$$S_N = \frac{1}{N} \sum_{k=1}^N \left[y(k) - \sum_{(r,s) \in \mathcal{S}} [1, \mathbf{z}^\top(k)] \cdot \boldsymbol{\beta}_{rs} \cdot D_{rs}(\mathbf{z}(k)) \right]^2 \quad (3.162)$$

where $y(k)$ is the model output and N is the number of training examples. Since the cost is linear in the weights, $\boldsymbol{\beta}_{rs}$, $(r, s) \in \mathcal{S}$ and the partition function filter is an orthogonal filter, the estimates, $\boldsymbol{\beta}_{rs}$, are unique and independently determined, cf. Sec. 5.2.

- The thresholds are determined by:

$$t_{rs} = \frac{1}{|\mathcal{K}_{rs}|} \sum_{k \in \mathcal{K}_{rs}} y(k) \quad (3.163)$$

where \mathcal{K}_{rs} is the set of indices k among which $\mathbf{z}(k) \in \mathcal{D}_{rs}$. $|\mathcal{K}_{rs}|$ is the cardinality. That is, the thresholds are determined so that each of the daughter nodes, $\mathcal{O}_{r+1,2s}$, $\mathcal{O}_{r+1,2s+1}$, contain approximately 50% of the data points. However, other strategies may as well be suggested.

³¹That is, $\alpha_0 = 0$ and $\alpha_i = 1, \forall i \in [1; q]$.

³²Note that the 2-norm was considered in `nlrffun`.

³³A particular input sample, $\mathbf{z}(k)$, belongs to one cluster only, i.e., the clusters are disjoint.

The algorithm now proceeds as follows:

Tree-Structured Piecewise-Linear Filtering Algorithm

- Step 1* *a.* Start at node \mathcal{O}_{00} and initialize: $(r, s) = (0, 0)$
 b. Estimate β_{00} by minimizing `nf:treecost`.
 c. Set $\mathbf{a}_{00} = \tilde{\beta}_{00}$.
 d. Estimate the threshold t_{00} cf. `nf:treethr`.
- Step 2* **if** $\mathbf{z}(k) \in \mathcal{H}_{rs}^-$ then continue to $\mathcal{O}_{r+1,2s}$
 Update: $(r, s) \leftarrow (r + 1, 2s)$
 otherwise continue to $\mathcal{O}_{r+1,2s}$
 Update: $(r, s) \leftarrow (r + 1, 2s + 1)$
- end**
- Step 3* Estimate β_{rs} by minimizing the cost cf. `nf:treecost`.
- Step 4* Set $\mathbf{a}_{rs} = \tilde{\beta}_{rs}$.
- Step 5* Estimate t_{rs} cf. `nf:treethr`.
- Step 6* If the number of terminal nodes is less than q , i.e., $|\mathcal{S}| < q$ then go to *Step 2*; otherwise, stop.

3.3.3 Gate Function Filters

Recall that the gate function filters are restrictions of the partition function filters mentioned in the previous section so that the hyperplanes are parallel to the coordinate axes, z_j , $j \in [1; p]$, i.e., the normal vectors comply with:

$$a_j = \begin{cases} 1 & j = j_i \\ 0 & \text{otherwise} \end{cases} \quad (3.164)$$

where j_i defines the direction of the i 'th hyperplane. In order to maintain the filter complexity (number of adjustable parameters) relatively low the basis functions are normally chosen to be constant (zero-order filter), $b_i(\mathbf{z}(k)) = \beta_{i,0}$, or linear (first-order filter), $b_i(\mathbf{z}(k); \beta_i) = \beta_i^\top \mathbf{z}(k)$.

The gate function filters were originally proposed in [Schetzen 89, Ch. 21] as zero-order filters. The filter has n gates at each input dimension, and for each input the thresholds are equally distributed over the amplitude range $\min_k z_j(k) \leq t \leq \max_k z_j(k)$. Consequently, the total number of gates – or the total number of adjustable parameters – equals $q = n^p$, and consequently one often faces the ‘‘curse of dimensionality’’. In order to alleviate this problem an *expand-adapt-reduce cycle* for zero- and first-order gate function filters was suggested in [Hoffmann 92a, Ch. 5]:

1. The expansion is done by splitting the gate which currently has the largest value of an expansion criterion (see also Sec. 6.8.4). The splitting is done by selecting a new hyperplane direction, say j_i , randomly among $j = 1, 2, \dots, p$, and the corresponding threshold, t_i , is estimated as the average of $z_{j_i}(k)$ over all data which enter the gate.

2. The parameters of the new gates are estimated (adapted) by minimizing the LS cost function.
3. Superfluous gates are merged according to a test for significance.

The Threshold Autoregressive model suggested in [Tong & Lim 80] is another proposal of a first-order gate function filter:

$$\hat{y}(k) = \sum_{i=1}^q [1, \mathbf{z}^\top(k)] \boldsymbol{\beta}_i \cdot D_i(z_\ell(k)). \quad (3.165)$$

All gate functions, $D_i(z_\ell(k))$, are thus merely a function of the single input variable $z_\ell(k)$, $\ell = 1, 2, \dots, p$. The parameters $\boldsymbol{\beta}_i$ are estimated according to a LS cost function, and the architecture of the filter is optimized by using Akaike's *AIC* Criterion [Akaike 74] (see e.g., Sec. 6.5.5). That is, a search over possible values of ℓ , p , and the thresholds specifying the gate functions are performed, and finally, the set of parameters which result in minimum *AIC* are selected.

3.4 Nonparametric Approximation

3.4.1 Local Filters

The idea of local filters was presented in [Farmer & Sidorowich 88]. Suppose that a set of input-output data $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, $k = 1, 2, \dots, N$ has been collected. Next consider the prediction of a new output sample y from the input vector, \mathbf{z} , i.e., estimate \hat{y} . The estimation of a local filter can be summarized in the following steps:

1. Find the N_t input vectors, $\mathbf{z}(k_n)$, $k_n \in [1; N]$, $n = 1, 2, \dots, N_t$, which lie in the vicinity of the novel input sample, \mathbf{z} . The vicinity may be defined in many ways, e.g., one may pick the N_t nearest neighbors in a 2-norm sense.
2. Choose a simple parametric model. A low order polynomial model was suggested in [Farmer & Sidorowich 88], e.g., the linear model:

$$\hat{y}(k_n) = \mathbf{w}^\top \mathbf{z}(k_n). \quad (3.166)$$

3. Estimate the parametric model from the N_t samples by minimizing some cost function (see Ch. 5), e.g., the LS cost

$$S_N(\mathbf{w}) = \frac{1}{N_t} \sum_{n=1}^{N_t} [y(k_n) - \mathbf{w}^\top \mathbf{z}(k_n)]^2. \quad (3.167)$$

Denote the estimated parameters by, $\hat{\mathbf{w}}$.

4. The prediction of the new output sample is consequently:

$$\hat{y} = \hat{\mathbf{w}}^\top \mathbf{z}. \quad (3.168)$$

The filter architecture is definitely nonparametric since it requires that the entire training set, \mathcal{T} , is stored, and for each new prediction (of an output sample) a search through the training set is required. The search is normally a very time consuming procedure, and

may be viewed as the counterpart of estimating (training) the parameters of a parametric architecture. A direct implementation of the search for neighbors (see item 1 above) is done by comparing the the new input, \mathbf{z} , to all $\mathbf{z}(k)$, $k = 1, 2, \dots, N$, i.e., N comparisons are required. It is possible to reduce the complexity to $\log_2 N$ comparisons by organizing the training data in a binary tree [Lapedes & Farber 87]. A particular node consists of a threshold, t , and a input dimension, j . Then the left subtree – which branches off from this node – contains all vectors, $\mathbf{z}(k)$ which j 'th coordinate is below the threshold, t – and vice versa for the right subtree. The threshold is, e.g., determined as the median of $z_j(k)$ which enters the gate associated with the considered node. Even with this delicated search the computational complexity may be too large, and moreover, the memory requirements may be too large. Hence, a pruning of the training set may be necessary [Farmer & Sidorowich 88].

There will be a trade off between the size of the region defining the vicinity of the new input sample and the complexity of the local mapping in order to ensure a proper estimation of the parameters, \mathbf{w} , i.e., N_t should be sufficiently large. Thus if the dimension of \mathbf{w} is too large compared to N_t the prediction of the output will be corrupted by inherent noise and/or noise stemming from the fact that the “true” mapping (generally) is different from the mapping employed. A detailed discussion of such matters is given in Ch. 6. Furthermore, in the limit where the region of “dense” input vectors is so large that all data are included we end with a parametric global approximation architecture.

3.4.2 Parzen Window Regression

In [Spect 91] a so-called General Regression Neural Network was presented. The basic idea is to employ the well-known property from the theory of regression, e.g., [Seber & Wild 89], [White 89a]: If the cost function is chosen as the expected squared errors (considering the input as known) – i.e.,

$$E \left\{ [y(k) - \hat{y}(k)]^2 | \mathbf{z} \right\}$$

– then the optimal regressor is given by the conditional mean,

$$\hat{y}(k) = E\{y|\mathbf{z}\} = \int_{-\infty}^{\infty} y \cdot p_{y|\mathbf{z}}(y|\mathbf{z}) dy = \frac{\int_{-\infty}^{\infty} y \cdot p_{\mathbf{z},y}(\mathbf{z}, y) dy}{\int_{-\infty}^{\infty} p_{\mathbf{z},y}(\mathbf{z}, y) dy} \quad (3.169)$$

where $p_{y|\mathbf{z}}(y|\mathbf{z})$ is the conditional probability density function (p.d.f.) of y given \mathbf{z} , and $p_{\mathbf{z},y}(\mathbf{z}, y)$ is the joint p.d.f. of \mathbf{z} and y . Accordingly, if an estimate of $p_{\mathbf{z},y}(\mathbf{z}, y)$ is available then the optimal model – in mean square sense - can be calculated from `mf:conmean`. For this purpose [Spect 91] suggested a particular Parzen window estimator (see e.g., [Duda & Hart 73, Sec. 4.3]) with Gaussian window function, i.e., the joint p.d.f. estimate becomes:

$$\hat{p}_{\mathbf{z},y}(\mathbf{z}, y) = \frac{1}{(\sqrt{2\pi}\sigma)^{p+1}} \cdot \frac{1}{N} \sum_{k=1}^N \exp \left[\frac{-[\mathbf{z} - \mathbf{z}(k)]^\top [\mathbf{z} - \mathbf{z}(k)]}{2\sigma^2} \right] \cdot \exp \left[\frac{-[y - y(k)]^2}{2\sigma^2} \right] \quad (3.170)$$

where N is the number of input-output data, and σ is a width parameter³⁴.

³⁴When employing a N -dependent width, $\sigma(N)$, then provided that $\lim_{N \rightarrow \infty} \sigma(N) = 0$ and $\lim_{N \rightarrow \infty} N\sigma^p(N) = \infty$, it can be shown (see e.g., [Spect 91]) that $\hat{p}_{\mathbf{z},y}(\mathbf{z}, y)$ is a consistent estimator of $p_{\mathbf{z},y}(\mathbf{z}, y)$ for all points (\mathbf{z}, y) at which $p_{\mathbf{z},y}(\cdot)$ is continuous.

Using this estimator results in the nonparametric estimator [Spect 91]:

$$\hat{y}(k) = \frac{\sum_{k=1}^N y(k) \exp \left[\frac{-[\mathbf{z} - \mathbf{z}(k)]^\top [\mathbf{z} - \mathbf{z}(k)]}{2\sigma^2} \right]}{\sum_{k=1}^N \exp \left[\frac{-[\mathbf{z} - \mathbf{z}(k)]^\top [\mathbf{z} - \mathbf{z}(k)]}{2\sigma^2} \right]}. \quad (3.171)$$

This estimator is in fact identical to that obtained by using a zero-order localized covering function filter with Gaussian bell-shaped covering function, cf. Sec. 3.3.1, provided that:

- The number of covering functions, q , is equal to the number of data, N .
- All widths are identical, i.e., $\sigma_i^2 = \sigma^2$, $i \in [1; N]$.
- The centers are located at the data points, i.e., $\mathbf{z}_i^o = \mathbf{z}(i)$, $i \in [1; N]$.
- An LS cost function is used to estimate the parameters of the covering functions, β_{i0} , $i \in [1; N]$ (see `nf:zocf`).

The first three items thus impose very strong restrictions on the general zero-order localized covering function filter with Gaussian covering functions. That is, the modeling flexibility of the general regression network is small. In fact, the architecture is determined by a single parameter only, viz. σ . Indirectly an estimation of N parameters β_{i0} , $i \in [1; N]$ from N training samples takes place. Consequently, one would immediately expect very bad model quality – as explained in Ch. 6; however, to some extent this can be avoided by carefully adjusting the width σ . In fact, [Spect 91] suggests a cross-validation procedure (cf. Sec. 6.5.3, Sec. 6.5.4) for this purpose.

3.5 Discussion

Let us summarize the presentation of various global, local, and nonparametric architectures by listing advantages and drawbacks – from a general point of view.

- The flexibility of local approximation architectures – including nonparametric architectures – may be larger than that of global approximation architectures when pursuing “black-box” modeling. This is due to the fact that the local filter merely is required to yield a close approximation within a restricted part of the input domain. Assuming that the the “true” input-output surface is appropriately smooth over the input domain (which is not particularly restrictive), then locally the surface may be easy to implement by a simple basis function³⁵. On the other hand, using a global approximation filter then one would expect that much more a priori knowledge concerning the “true” system must be available in order to obtain a reasonable parameterization, i.e., avoiding the “curse of dimensionality”. However, do notice that the statements are based on a “black-box” modeling point of view. That is, if specific a priori knowledge is available then this should be incorporated into the choice of architecture; consequently, this might lead to a preference to global approximation filters. In addition, recall that the multi-layer perceptron neural network and the canonical piecewise-linear filter possess the ability to make semi local

³⁵Note that the basis functions are not tailored to the actual problem due to lack of a priori knowledge.

representations; however, they are normally not tailored to this purpose, so local representations will appear only if there is a profit – as regards lower cost. This is possibly impeded by the fact that the optimization algorithm is trapped in a local minimum of the cost function.

- If data are sparse the global architecture often outperforms the local and nonparametric architectures. This is due to the fact that too few data points enter each of the local domains³⁶, and consequently, it becomes impossible to obtain proper estimates of the associated parameters. This indeed lead to poor generalization ability (see further Ch. 6). Hence – with a specific amount of data – there will be a tradeoff between the number of local domains and the complexity of the local filter.
- More of the local filters will result in input–output surfaces which are discontinuous at the boundaries between the local domains. This may cause an reduction of the generalization ability. However, the literature provides several proposals in order to remedy this obstacle, see e.g., [Friedman 91], [Sørensen 92].
- Within many of the mentioned local approximation filters the shape of the local domains is tailored to the data – this includes e.g., the localized covering function filters. This is indeed an important item in order to:
 - Prevent that too few data enter the domains.
 - Ensure that input space obtains a sufficient division in the regions where data are frequent.
- In connection with algorithms which optimize the chosen architecture partition function filters may be advantageous as regards algorithms which gradually expand the architecture cf. Sec. 6.8.4.

3.6 Summary

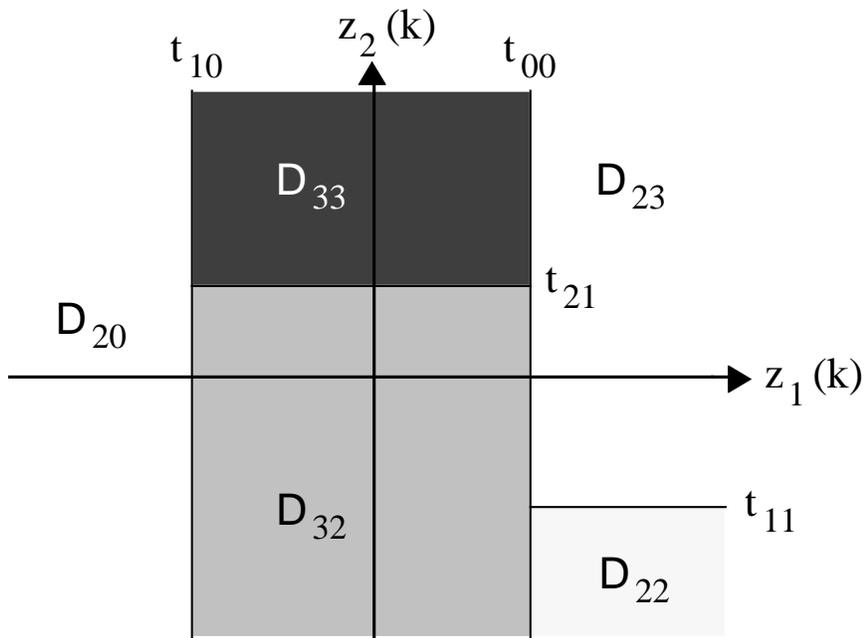
In this chapter we presented a novel taxonomy for nonlinear filter architectures which is based on a few significant architectural properties, including

- Global versus local approximation.
- Parametric versus nonparametric architectures.

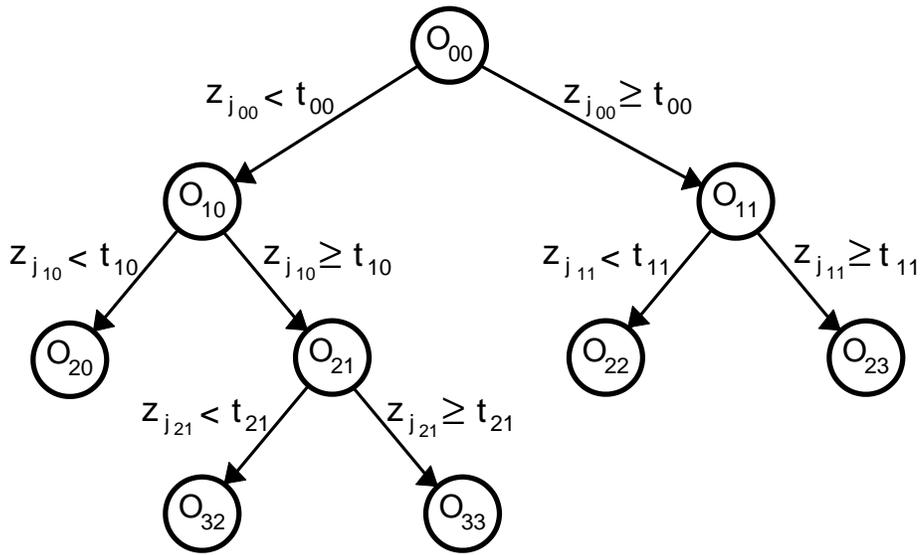
The intention was to provide a unified view of possible architectures, and the taxonomy might be used to guide the choice of a suitable architecture when facing a particular modeling task. Furthermore, it may suggest the development of novel structures.

A number of existing nonlinear filters architectures – mainly based on a neural network approach – were classified according to the taxonomy. Furthermore, we provided reviews, interpretations and comparisons of various selected filter architectures. In particular, we attached importance to the multi-layer perceptron neural network.

³⁶Subsidiary, the vicinity of the new input vector – if one deals with the local filter Sec. 3.4.1.



(a)



(b)

Figure 3.5: Example of a binary tree and the corresponding partitions, \mathcal{D}_{rs} , when the hyperplanes are restricted to be parallel to the co-ordinate axes. The domain function is in this case denoted: Gate functions.

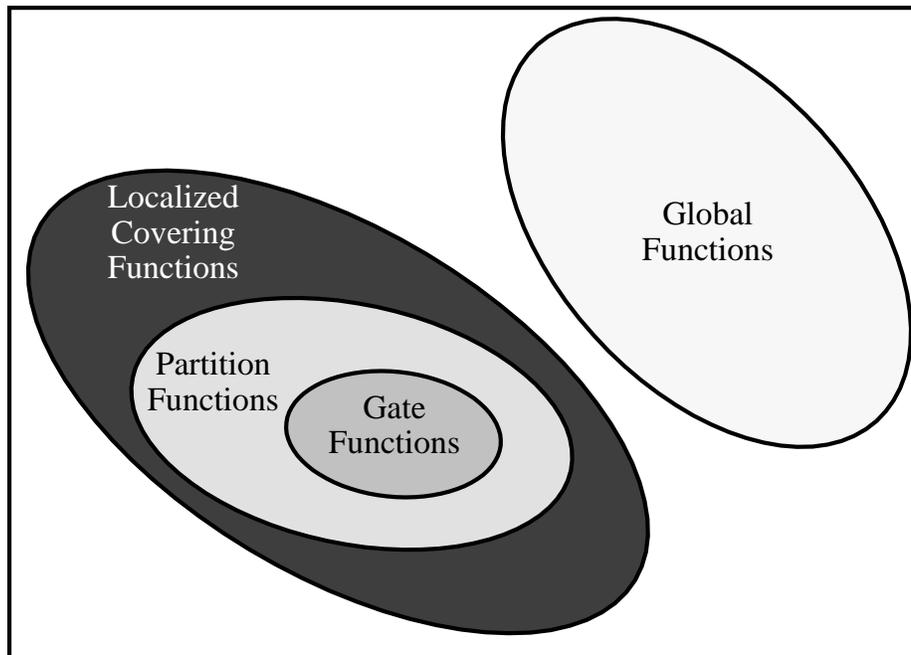


Figure 3.6: Venn-diagram which shows possible special cases of the domain function, $D(\cdot)$. It is noted that no intersection between global functions and localized covering functions is present. Furthermore, the diagram shows the ranking of the localized functions.

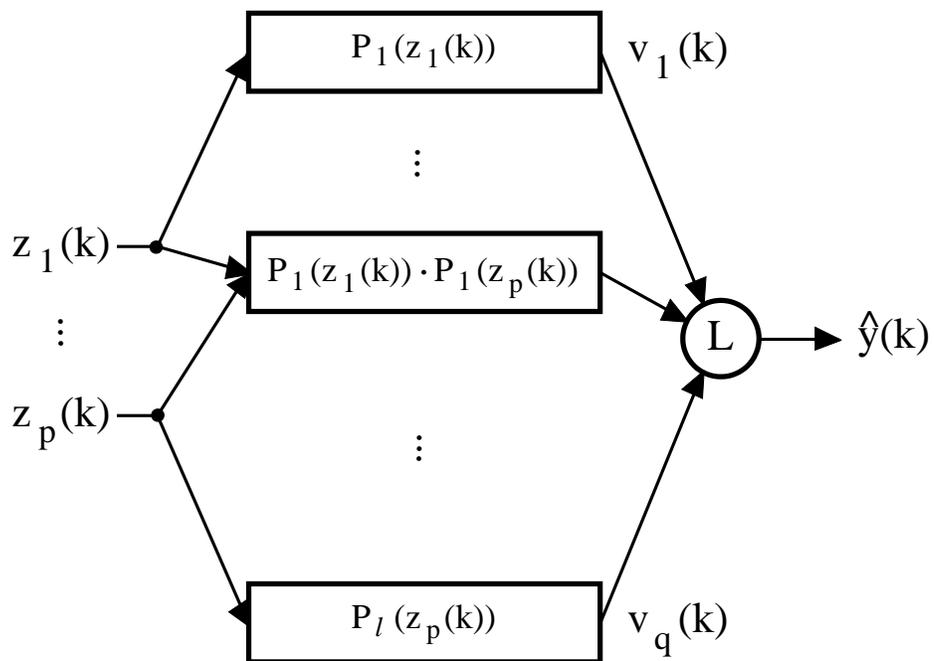


Figure 3.7: The general l 'th order polynomial filter. $P_r(z)$ is a r 'th order polynomial in z . Recall (see page 6), that the processing of the linear neuron (illustrated by the L) in this case yields: $\hat{y}(k) = \alpha_0 + \sum_{i=1}^q \alpha_i v_i(k)$.

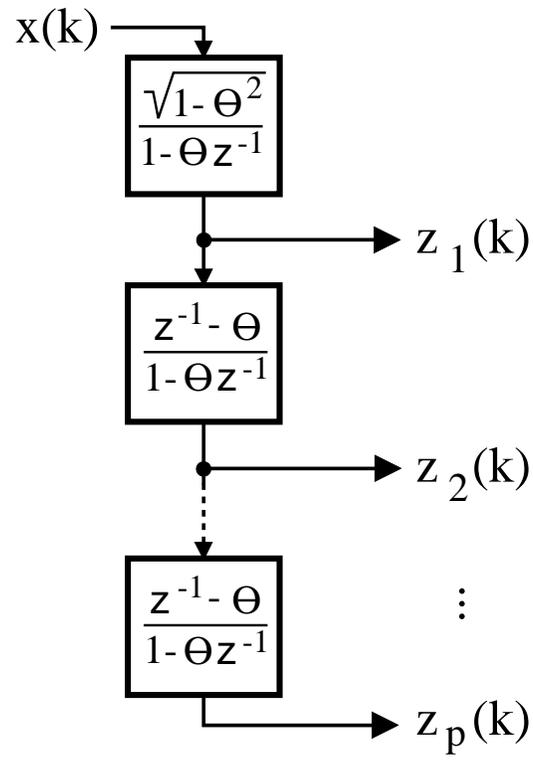


Figure 3.8: Construction of the $z_j(k)$ signals by filtering $x(k)$ with discrete Laguerre function filters.

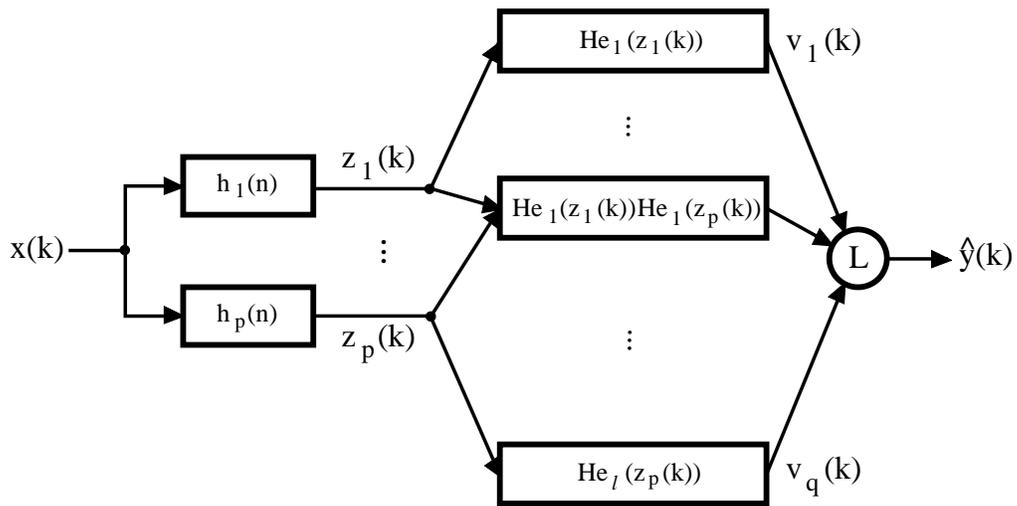


Figure 3.9: The Wiener model based l -order polynomial XN-filter. $x(k)$ is assumed to be a zero mean white Gaussian signal; however, it is possible to relax the zero mean and white conditions. $h_j(k)$ is the Laguerre functions impulse responses (cf. `nf:lagufile`) and $\text{He}_r(z)$ is the Hermite polynomial of degree r (cf. `nf:hermit`).

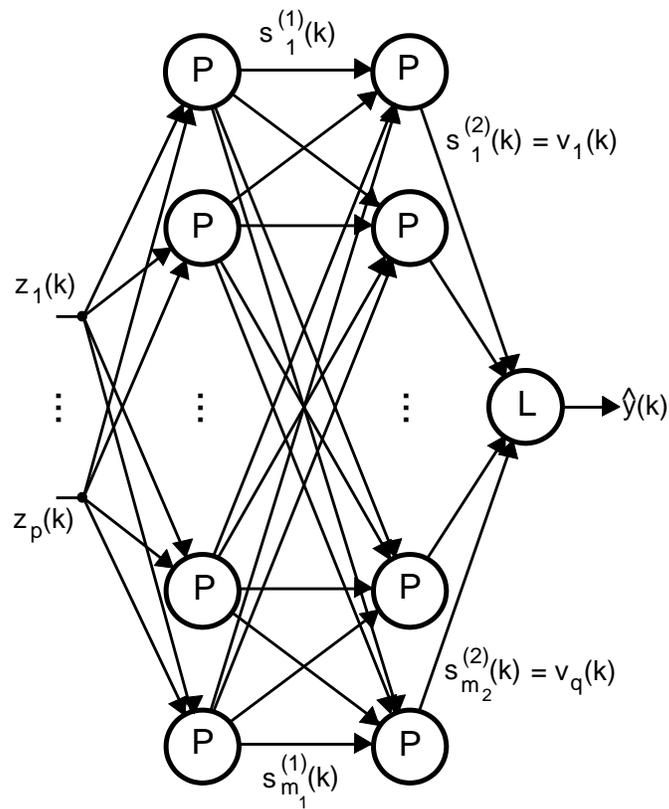


Figure 3.10: XN-filter based on a 3-layer feed-forward neural network. The neurons denoted, P, are nonlinear perceptrons while, L, refers to the linear neuron (linear perceptron).

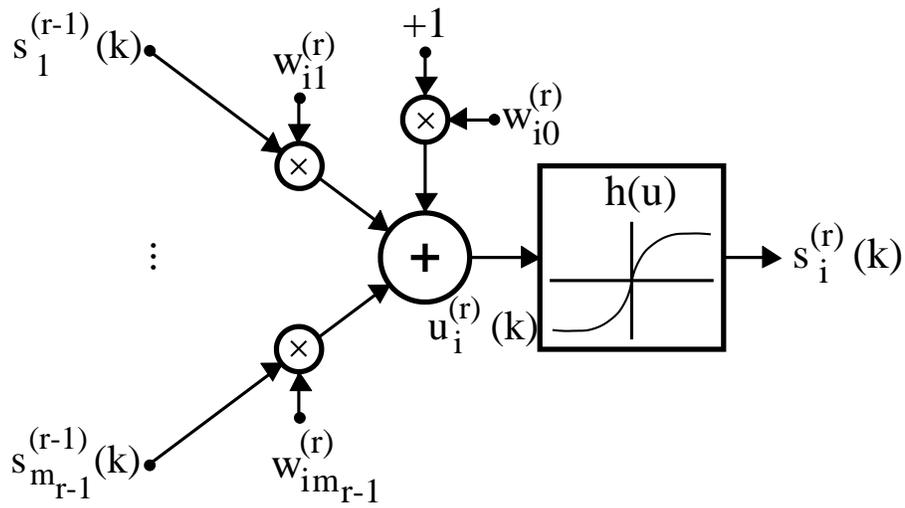


Figure 3.11: The signal processing within a nonlinear perceptron. $w_{i,j}^{(r)}$ are the associated weights which specify the linear combination of the $s_i^{(r-1)}(k)$ signals. $h(\cdot)$ is the nonlinear activation function.

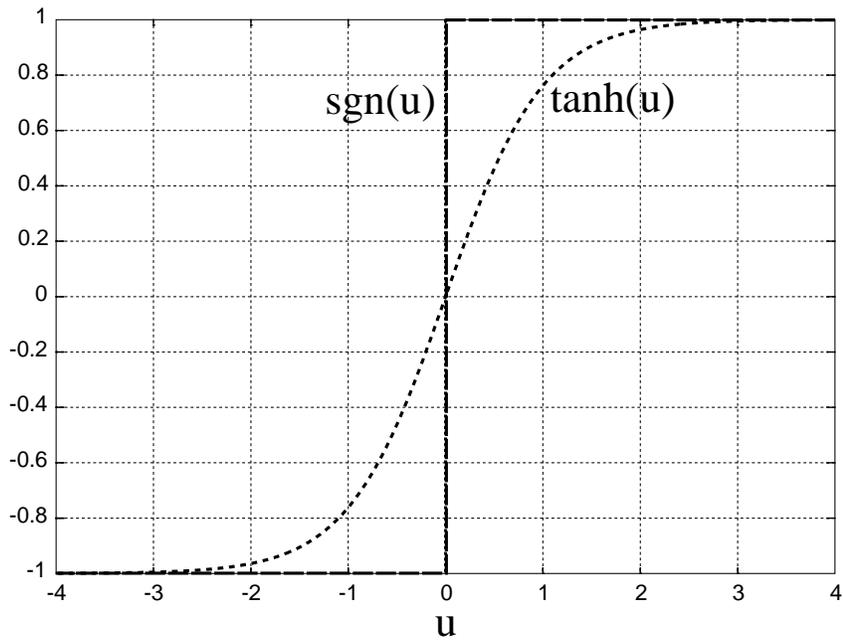


Figure 3.12: Two examples of typical activation functions: The signum function (solid line) and the hyperbolic tangent (dotted line).

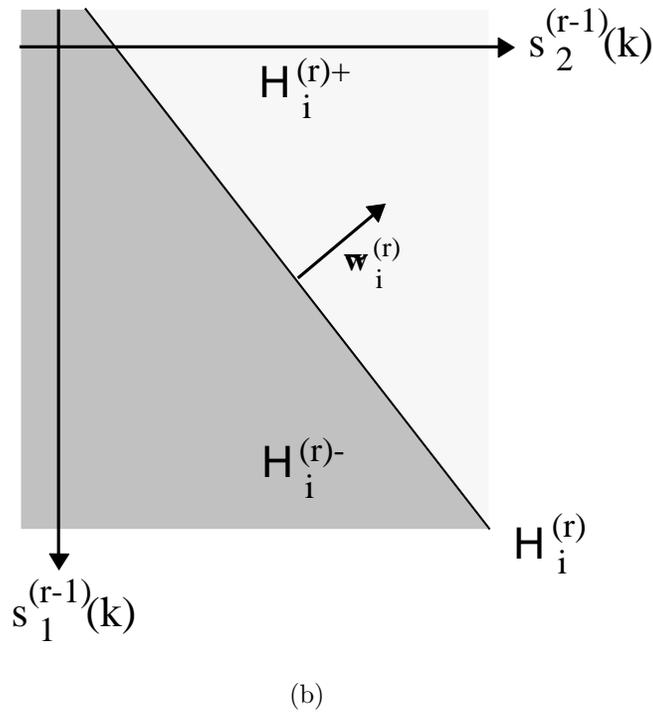
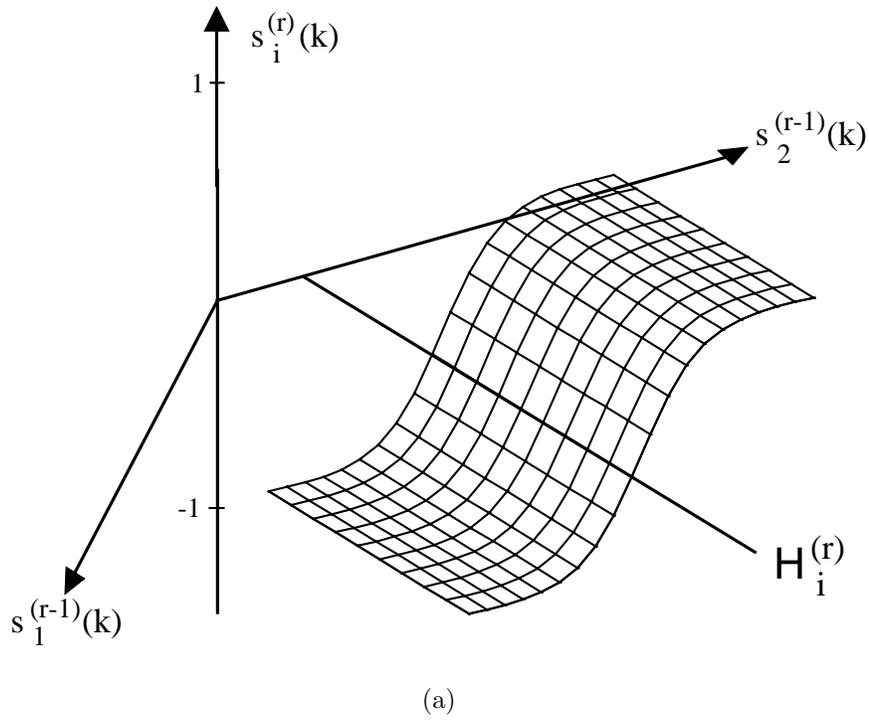


Figure 3.13: The geometrical interpretation of the processing involved in the perceptron: $s_i^{(r)}(k) = \tanh\left((\mathbf{w}_i^{(r)})^\top\right) \mathbf{s}^{(r-1)}(k)$. $\mathcal{H}_i^{(r)}$ is the hyperplane, $(\mathbf{w}_i^{(r)})^\top \mathbf{s}^{(r-1)} = 0$, which discriminates between the regions, $\mathcal{H}_i^{(r)+}$, $\mathcal{H}_i^{(r)-}$.

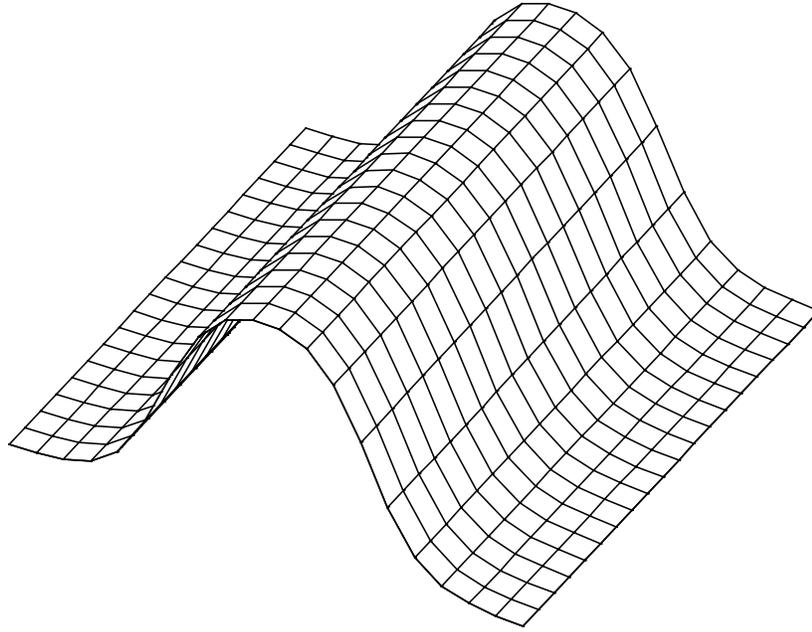


Figure 3.14: A semi localized covering function formed by two hyperbolic tangent neurons.

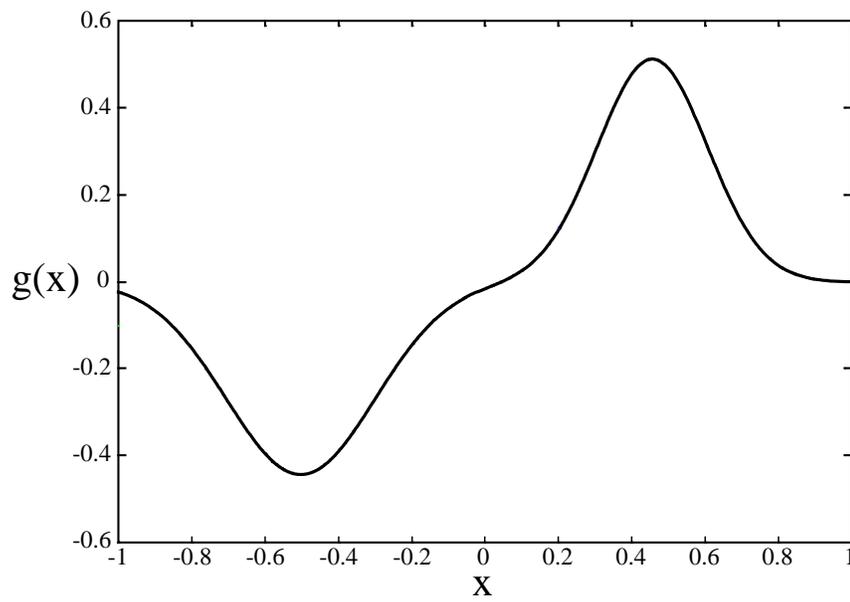


Figure 3.15: Graph of $g(x)$ given by nf:gxexam.

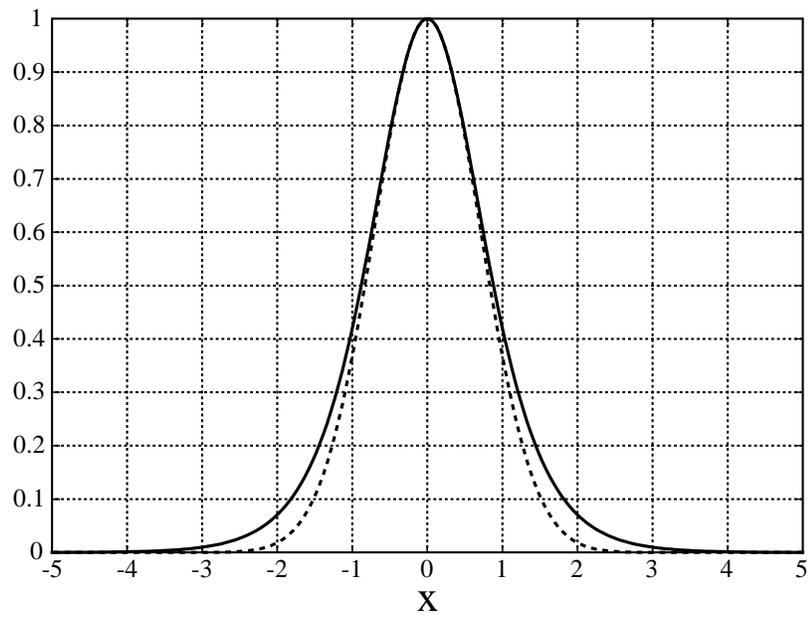


Figure 3.16: Approximation of the Gaussian bell $\exp(-x^2)$ (dotted line) by a sum of two hyperbolic tangent functions (solid line) cf. `nf:tanhsum`.

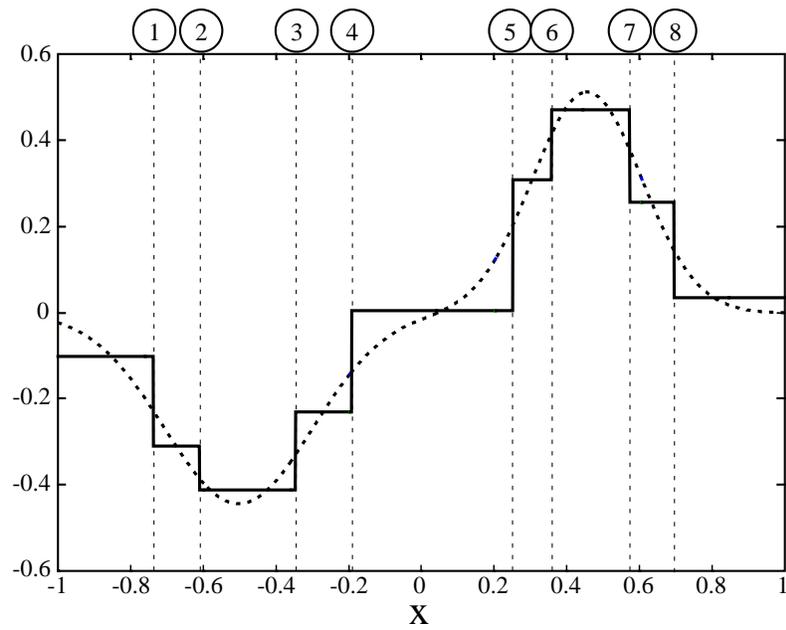


Figure 3.17: $g(x)$ (dash-dotted line) and the piecewise-constant approximation (solid line) produced by the neural network using hard limiters as activation functions cf. `nf:hlapprox`. The vertical dashed lines are the hyperplane locations of the eight neurons cf. `nf:hypneu`.

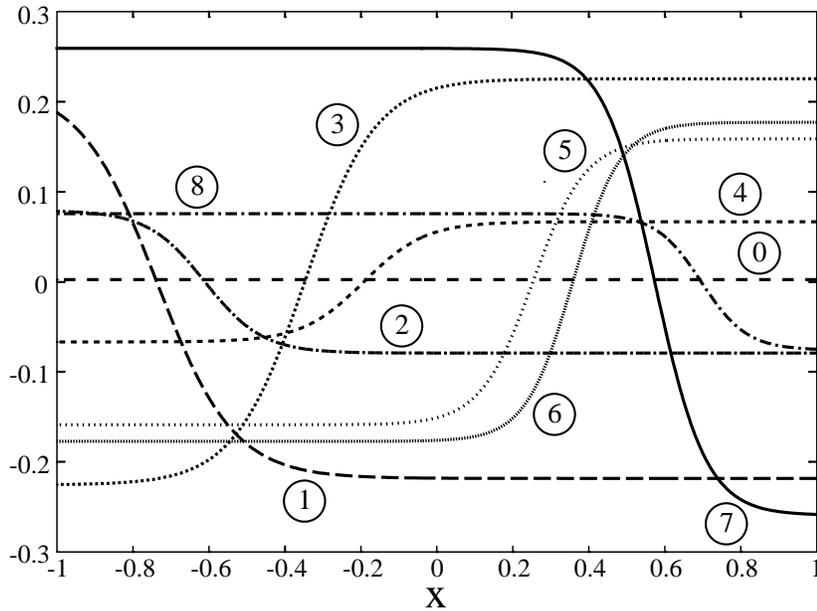


Figure 3.18: The contributions from the individual hidden neurons to the filter output. The numbers refer to the specific neuron. Comparing the figure with Fig. 3.17 we note that the left most descending flank in $g(x)$ is modeled by neuron #1 and #2. The neurons #3 and #4 provide the subsequent rising and simultaneously neuron #4 creates the shift to the new ascending phase around $x = 0.2$. Neuron #5 and #6 contribute to this ascending and finally neuron #7 and #8 give the final descending. Neuron #0 adjusts the overall mean level. In this case only a slight adjustment is needed due to the symmetries of $g(x)$.

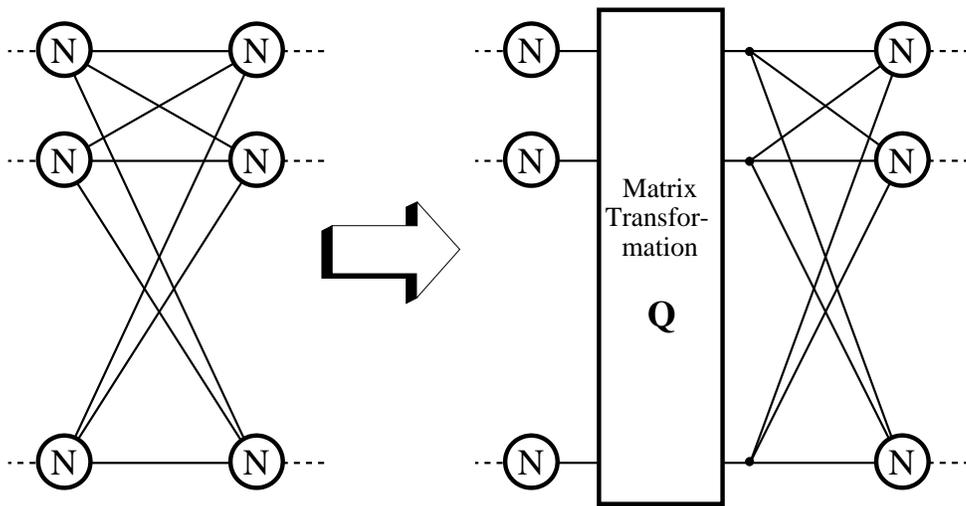


Figure 3.19: The Gram-Schmidt Neural Network is obtained by introducing a matrix transformation (or a layer of linear neurons) of the state vector in each layer so that the inputs to the succeeding layer become uncorrelated and normalized.

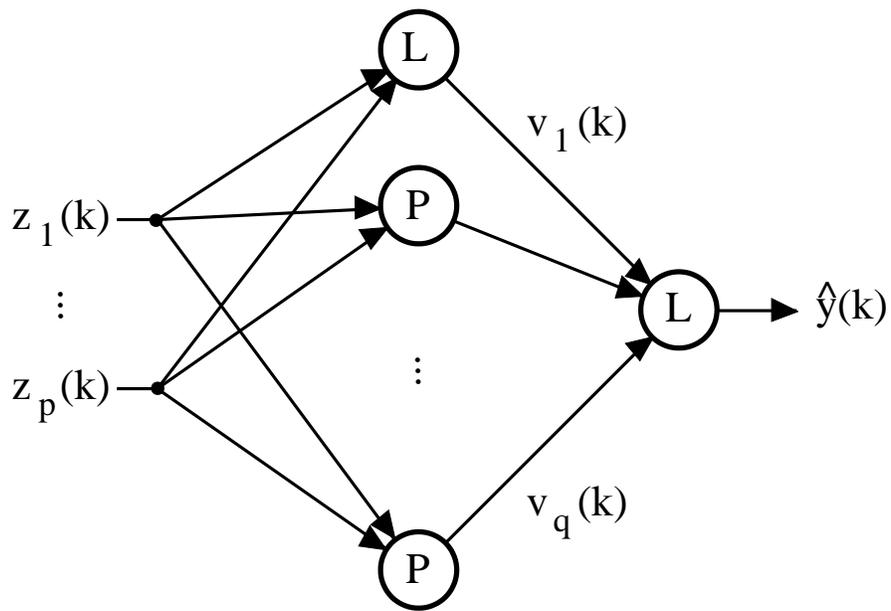


Figure 3.20: The PWLF regarded as 2-layer MFPNN.

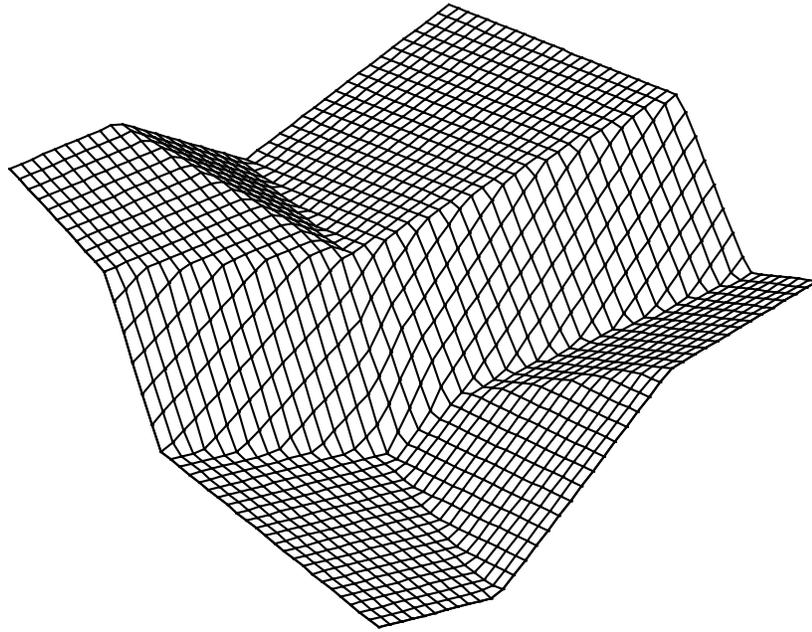


Figure 3.21: A typical input-output surface (in the case $p = \dim(\mathbf{z}) = 2$) produced by the PWLF. The input-output surface consists of adjoining p -dimensional hyperplanes which not necessarily are horizontal as is the case when using a 2-layer MFPNN.

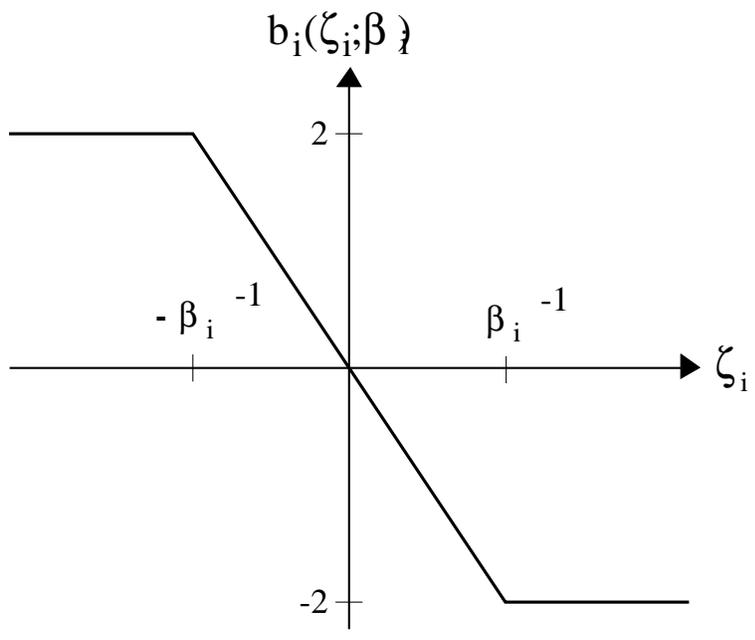


Figure 3.22: The sigmoidal shape of basis functions used in [Lin & Unbehauen 90].

CHAPTER 4

A GENERIC NONLINEAR FILTER ARCHITECTURE BASED ON NEURAL NETWORKS

In this chapter we present a generic nonlinear filter architecture (GNFA) based on neural networks which originally was proposed in [Hoffmann & Larsen 91], see also App. H. The generic filter architecture consists of a preprocessing unit succeeded by a (normally zero-memory) nonlinear filter given by the canonical filter representation. Various implementations of the nonlinear filter have already been discussed in Ch. 3; consequently, we principally focus on the construction of the preprocessing unit.

4.1 The Generic Neural Network Architecture

Recall the general form of the non-recursive model with additive error:

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \quad (4.1)$$

where

- $y(k)$ is the output signal.
- $f(\cdot)$ is the filtering function.
- $\mathbf{x}(k)$ is the input vector signal

$$\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-L+1)]^\top \quad (4.2)$$

where L specifies the memory length (window length).

- \mathbf{w} is the m -dimensional weight vector.
- $e(k; \mathbf{w})$ is the error signal.

Now consider *preprocessing* of the input signal vector:

$$\mathbf{z}(k) = \mathbf{f}_p(\mathbf{x}(k)) \quad (4.3)$$

where $\mathbf{z}(k)$ is the p -dimensional preprocessed vector

$$\mathbf{z}(k) = [z_1(k), z_2(k), \dots, z_p(k)]^\top \quad (4.4)$$

which – in contrast to $\mathbf{x}(k)$ – does not contain a time shift structure. Furthermore, $\mathbf{f}_p(\cdot)$ is the preprocessing vector function: $\mathbb{R}^L \mapsto \mathbb{R}^p$.

Accordingly, the filtering function obeys the decomposition:

$$\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}) \triangleq f_n(\mathbf{z}(k); \mathbf{w}) \quad (4.5)$$

where $f_n(\cdot)$ is denoted the *nonlinear filtering function*. Various implementations of $f_n(\cdot)$ are already discussed in Ch. 3, and – in particular – $f_n(\cdot)$ is given by the *canonical filter representation* (cf. Sec. 3.1.2.1):

$$\begin{aligned} f_n(\mathbf{z}(k); \mathbf{w}) &= \alpha_0 + \sum_{i=1}^q \alpha_i \cdot b_i(\mathbf{z}(k); \beta_i) \cdot D_i(\mathbf{z}(k)) \\ &= \alpha_0 + \sum_{i=1}^q \alpha_i v_i(k) \\ &= \boldsymbol{\alpha}^\top \mathbf{v}(k) \end{aligned} \quad (4.6)$$

where $b_i(\cdot)$ is the i 'th basis function parameterized by β_i , α_i weights the contribution of the i 'th basis functions, and $D_i(\cdot)$ denotes the i 'th domain function.

In Fig. 4.1 the proposed generic neural network architecture is depicted. The architecture may be viewed as a heterogeneous three-layer neural network: The first layer is the preprocessing unit, the second layer is the nonlinear filtering, and the third layer consists of the linear output neuron. Contrary to a conventional multi-layer neural network each unit is interpretative; thus it may be easier to incorporate a priori knowledge, and secondly ensuring an effective implementation of the filtering function.

4.2 Preprocessing Methods

Design of the preprocessing unit comprises the following issues:

- By ensuring a small p the “curse of dimensionality” may probably be circumvented. According to Ch. 3 even for moderate values of p the dimension of the weight vector m typically becomes large. Consequently, in order to ensure high quality of the model (see further Ch. 6), a huge number of training data has to be available.
- The mapping constituted by the preprocessing unit should extract the essential information contained in $x(k)$ w.r.t. the output $y(k)$. This includes estimation of p , L .
- Preprocessing includes a suitable phase compensation so that the optimal filtering (from $x(k)$ to $y(k)$) is ensured to be causal.

In general, the preprocessing function may be nonlinear; however, here we merely consider linear preprocessing, i.e.,

$$z_j(k) = h_j(k) * x(k), \quad j = 1, 2, \dots, p \quad (4.7)$$

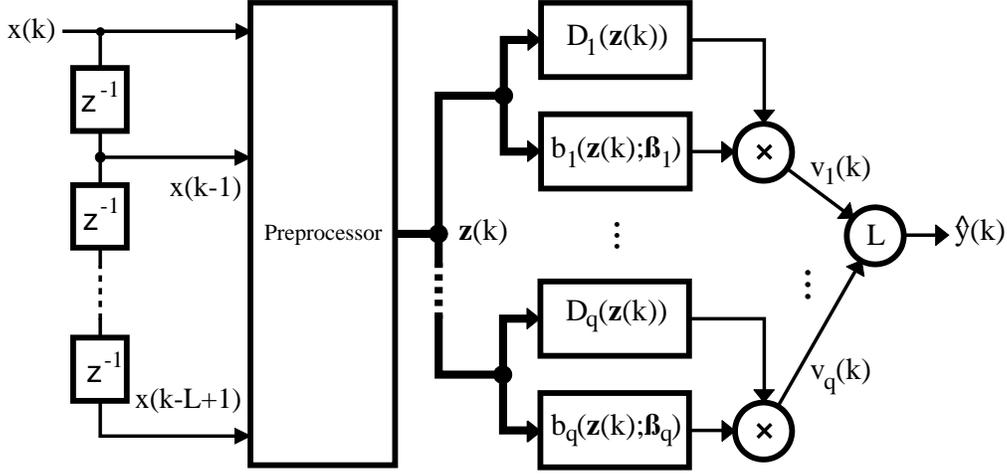


Figure 4.1: The generic neural network architecture

where $*$ denotes convolution and $h_j(k)$ is the impulse response of the j 'th preprocessing filter. The associated transfer function is denoted by, $H_j(z)$. Fig. 4.2 shows the linear preprocessor. In general we may write:

$$H_j(z) = \frac{\Xi_j(z)}{\Psi_j(z)} \quad (4.8)$$

where

$$\Xi_j(z) = 1 + \xi_{j,1} z^{-1} + \dots + \xi_{j,L-1} z^{-L+1}, \quad (4.9)$$

$$\Psi_j(z) = 1 + \psi_{j,1} z^{-1} + \dots + \psi_{j,P-1} z^{-P+1}. \quad (4.10)$$

If $\Psi_j(z) \equiv 1$ the filter is a FIR-filter; otherwise, an IIR-filter. When considering FIR-filters we also employ the matrix formulation:

$$\mathbf{z}(k) = \mathbf{C}\mathbf{x}(k) \quad (4.11)$$

where \mathbf{C} is the $p \times L$ dimensional *preprocessing matrix*

$$\mathbf{C} = \begin{bmatrix} h_1(1) & h_1(2) & \dots & h_1(L) \\ h_2(1) & h_2(2) & \dots & h_2(L) \\ \vdots & \vdots & \ddots & \vdots \\ h_p(1) & h_p(2) & \dots & h_p(L) \end{bmatrix}. \quad (4.12)$$

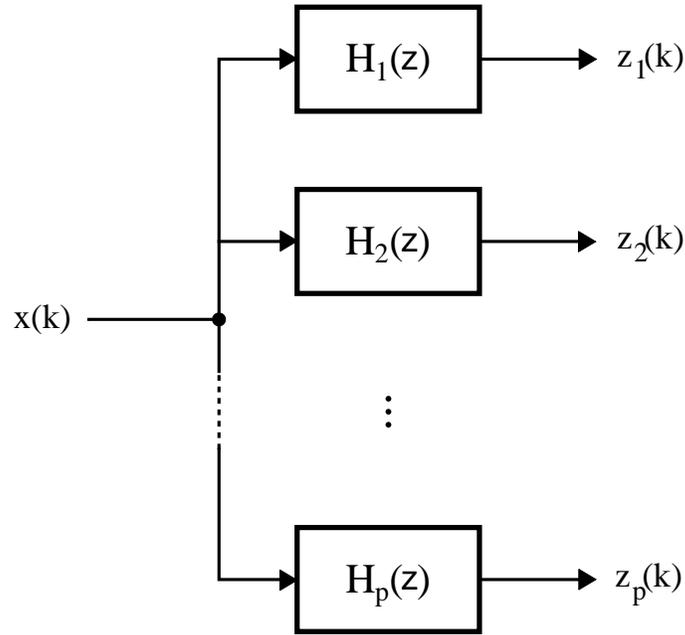


Figure 4.2: The linear preprocessor.

The filters, $h_j(k)$, act like a set of additional adjustable parameters which can be determined according to different strategies:

1. The parameters $\xi_{j,s}$, $\psi_{j,r}$, may be adjusted like the ordinary filter weight, \mathbf{w} , so that the overall cost function is minimized according to the set of related input-output data – the training set: $\mathcal{T} = \{\mathbf{x}(k); y(k)\}$, $k = 1, 2, \dots, N$. This strategy is referred to as *adaptive preprocessing*.

Often the required memory length of the filter involves relatively large L and P so that one faces the “curse of dimensionality”. However, it is still possible to impose some constraints on the preprocessing filters so that the total number of adjustable parameters is reduced. An example is given in Sec. 4.5.

2. The “curse of dimensionality” is better avoided by using a *fixed preprocessor*. That is, $\xi_{j,s}$, $\psi_{j,r}$ are predetermined¹ by means of general theoretical considerations or a priori knowledge. Obviously, the strategy involves a risk of degrading the overall performance; however, it should be emphasized that it is not possible to give any general theoretical statements which settles the optimal strategy since it definitely is highly dependent on the task under consideration. In Ch. 6 we elaborate on various matters concerning the search for an optimal architecture – which thus also comprises preprocessor design.

¹That is, the preprocessor is determined according to an unsupervised learning scheme.

4.2.1 Preprocessing Algorithm

In the subsequent sections various preprocessing methods for selecting $H_j(\mathbf{z})$ are discussed; however, let us first consider the problem of determining the memory length, L , and adjusting the phase between $x(k)$ and $y(k)$ when dealing with fixed FIR preprocessors. A suboptimal approach is given by the following assumptions:

- The phase is adjusted by time-shifting $x(k)$ and $y(k)$. Let D_x, D_y be the amount with which $x(k)$ and $y(k)$ are delayed, respectively. In particular, we define a common phase parameter, D , so that:

$$D_x = \begin{cases} -D & D \leq 0 \\ 0 & \text{otherwise} \end{cases}, \quad D_d = \begin{cases} D & D \geq 0 \\ 0 & \text{otherwise} \end{cases}. \quad (4.13)$$

- We consider the filtering function to be linear, i.e., $f(\mathbf{x}(k); \mathbf{w}) = \mathbf{w}^\top [1, \mathbf{x}^\top(k)]^\top$.

Initial estimates of L, D , can be obtained by using a nonparametric estimate of the linear impulse response, $h_{xy}(k)$, defined by:

$$y(k) = h_{xy}(k) * x(k). \quad (4.14)$$

The estimate can be obtained according to:

$$h_{xy}(k) = \text{Fou}^{-1} \{H_{xy}(f)\} \quad (4.15)$$

where $\text{Fou}^{-1}\{\cdot\}$ is the inverse Fourier transform and $H_{xy}(f)$ is the transfer function which according to [Bendat & Piersol 86, Sec. 9.2] obey the relation²:

$$H_{xy}(f) = \frac{P_{xy}(f)}{P_x(f)} \quad (4.16)$$

where $P_{xy}(f)$ is the cross-power spectrum, and $P_x(f)$ is the power spectrum given by:

$$P_{xy}(f) = \text{Fou} \{ \phi_{xy}(\tau) \} \quad (4.17)$$

$$P_x(f) = \text{Fou} \{ \phi_x(\tau) \} \quad (4.18)$$

where $\text{Fou}\{\cdot\}$ is the Fourier transform operator, and $\phi_{xy}(\tau), \phi_x(\tau)$ are the cross- and auto-correlation functions, respectively. The density spectra are estimated by using the method of Welch [Oppenheim & Schafer 89, Sec. 11.6]³ Fig. 4.3 shows an example of the impulse response produced by the mentioned procedure. A nonparametric estimate of the memory length, say L_{np} , is then given by the approximated length of the impulse response which e.g., is defined as the smallest time-interval containing $1 - \alpha$ per cent of the energy of $h_{xy}(k)$ where α is a small percentage. The nonparametric estimate of the phase parameter, D_{np} , is subsequently given as the time-shift which ensures an (essential) zero impulse response for $k < 0$ (see Fig. 4.3).

²Recall that we consider both $x(k)$ and $y(k)$ to be stochastic signals.

³Frequently one uses a Hann-weighted FFT, and 50% overlap among neighbor windows ensures low variance on the estimates. Furthermore, a number of zeros (equal to the desired length of the impulse response) is appended to the signals before performing FFT in order to avoid effects from circular convolution.

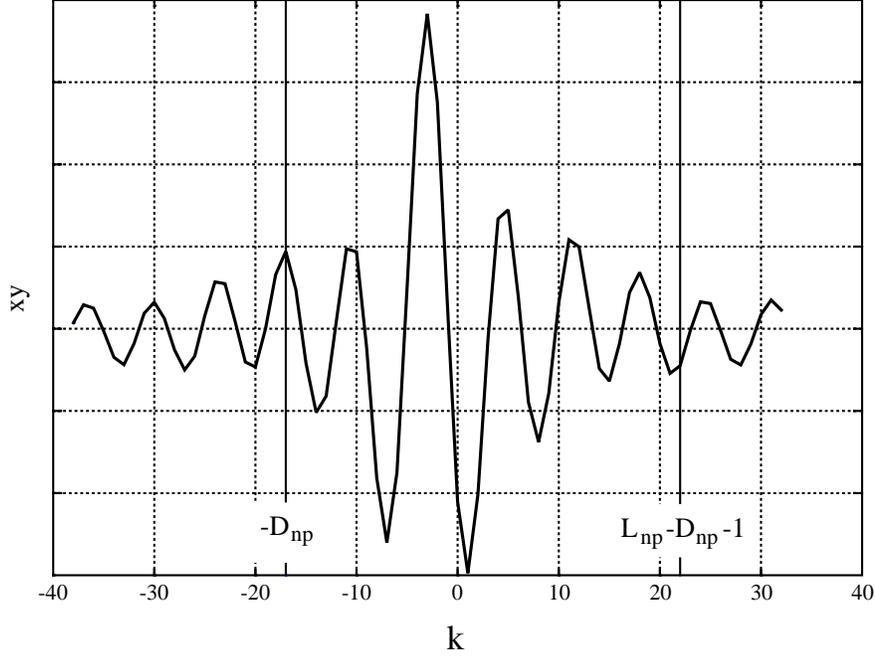


Figure 4.3: Nonparametric linear impulse response of the relation among $x(k)$ and $y(k)$. D_{np} , L_{np} are the nonparametric estimates of phase parameter and the memory length, respectively.

The estimates of L and D can be further refined by searching over possible sets of candidate values:

$$L \in \mathcal{L} = \{L_1, L_2, \dots, L_r\}, \quad (4.19)$$

$$D \in \mathcal{P} = \{D_1, D_2, \dots, D_s\} \quad (4.20)$$

$$(4.21)$$

where L_i , D_i are values in the vicinity of L_{np} , D_{np} . Again using the suboptimal approach that the filtering function is linear, Fig. 4.4 shows the configuration for performing the search over candidate values. Suppose that the preprocessing method has been selected and the filters $H_j(z)$ are determined. With specific values of L and D (and consequently D_x , D_y) the weights of the linear neuron⁴ are estimated so that the cost function is minimized. For instance, one may choose to minimize the least squares cost function,

$$S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N \left(y(k) - \mathbf{w}^\top [1, \mathbf{z}^\top(k)]^\top \right)^2 \quad (4.22)$$

according to the training set: $\mathcal{T} = \{\mathbf{x}(k); y(k)\}$, $k = 1, 2, \dots, N$. In Ch. 5 the topic of weight estimation is discussed in detail. The quality of the simple LL-model Fig. 4.4 is

⁴Note that a bias term is included in the linear neuron.

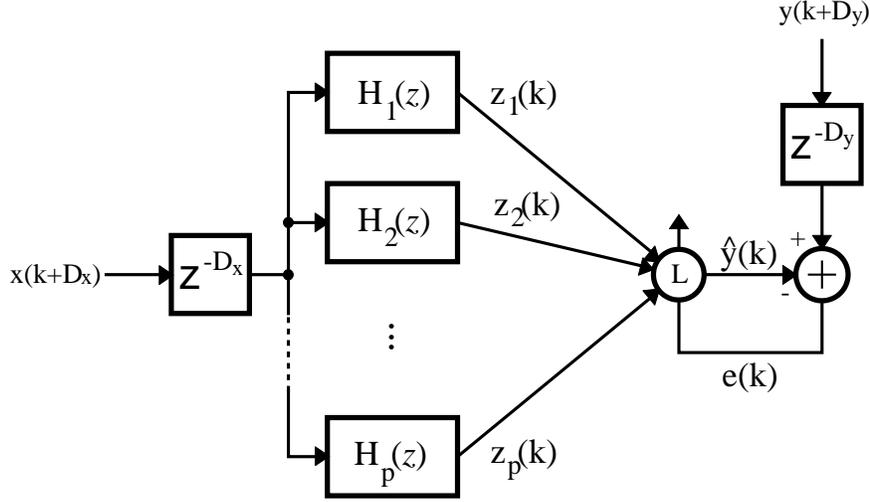


Figure 4.4: Configuration for determination of the memory length, L , and the phase parameter, D .

evaluated by considering the cross-validation estimate, C , (see further Sec. 6.5.3) which measures the average square error on a data set *independent* of the training set, i.e.,

$$C = \frac{1}{N_c} \sum_{k=1}^{N_c} \left(y(k) - \hat{\mathbf{w}}^\top [1, \mathbf{z}^\top(k)]^\top \right)^2 \quad (4.23)$$

where N_c is the size of the cross-validation set, and $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} S_N(\mathbf{w})$. Finally, L , D are estimated by minimizing C over the candidate sets, \mathcal{L} , \mathcal{P} , i.e.,

$$[\hat{L}, \hat{D}] = \arg \min_{\mathcal{L}, \mathcal{P}} C. \quad (4.24)$$

In summary, we formulate the Preprocessing Algorithm (PPA).

Preprocessing Algorithm Perform a nonparametric estimate of the memory length and the phase parameter, i.e., determine L_{np} , D_{np} .
Choose search sets,

$$\begin{aligned} L &\in \mathcal{L} = \{L_1, L_2, \dots, L_r\} \\ D &\in \mathcal{P} = \{D_1, D_2, \dots, D_s\} \end{aligned}$$

around the nonparametric estimates L_{np} , D_{np} . Select a preprocessing method, i.e., the structure of the preprocessing matrix \mathbf{C} . Initialize $C_{\min} = \infty$.

```

Step 4   for  $L \in \mathcal{L}$ 
           Estimate  $p$ 
           Estimate  $C$ 
           for  $D \in \mathcal{P}$ 
             Determine  $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} S_N(\mathbf{w})$ 
             Calculate the cross-validation estimate,  $C$ 
             if  $C < C_{\min}$ 
                $C_{\min} = C$ 
                $\hat{L} = L, \hat{D} = D$ 
             end
           end
         end

```

Elaborations:

- The assumption that the filtering function is linear can be relaxed to include any kind of nonlinear filter; however, this significantly increases the computational complexity. Simulation given in Ch. 8 indicates that the simple approach works satisfactory.
- The cross-validation estimate of the model quality can be replaced by other estimators considered in Ch. 6. Furthermore, one may also apply alternative strategies for architecture synthesis discussed in Ch. 6.
- If adaptive preprocessors are considered the cost function includes additional parameters specifying the preprocessor.
- Even though the final design includes an adaptive preprocessor, we still may employ a fixed preprocessor when optimizing L and D . Moreover, the fixed preprocessor may serve as an initialization of the adaptive preprocessor.
- If an IIR preprocessor is considered the preprocessing algorithm also includes a loop over possible orders, $P - 1$, of the denominator, $\Psi_j(\mathbf{z})$.

4.3 Principal Component Analysis

The well-known method *principal component analysis* (PCA) may be used to design a fixed FIR preprocessor. This idea is similar to what is known as *principal component regression* in the linear regression literature [Draper & Smith 81].

The origin is the covariance matrix of the input vector. Assume that $x(k)$ is a stationary signal and let $\bar{x}(k) = x(k) - E\{x(k)\}$, then the covariance matrix is the symmetric, positive

semi-definite, Toeplitz $L \times L$ matrix given by:

$$\mathbf{\Sigma}_x = E \left\{ \bar{\mathbf{x}}(k) \bar{\mathbf{x}}^\top(k) \right\} = \begin{bmatrix} \gamma_x(0) & \gamma_x(1) & \cdots & \gamma_x(L-1) \\ \gamma_x(1) & \gamma_x(0) & \cdots & \gamma_x(L-2) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_x(L-1) & \gamma_x(L-2) & \cdots & \gamma_x(0) \end{bmatrix} \quad (4.25)$$

where $\gamma_x(\tau) = E\{\bar{\mathbf{x}}(k)\bar{\mathbf{x}}(k+\tau)\}$ is the autocovariance function. Consider next the eigenvalue decomposition:

$$\mathbf{\Sigma}_x = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top \quad (4.26)$$

where $\mathbf{\Lambda} = \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_L\}$ is the $L \times L$ diagonal matrix of eigenvalues:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_L, \quad (4.27)$$

and \mathbf{Q} is the $L \times L$ matrix of eigenvectors

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_L] \quad (4.28)$$

where \mathbf{q}_s is the normalized eigenvector associated λ_s . If the components of $\mathbf{x}(k)$ are highly correlated then only a few, say p , eigenvalues are large and the rest close to zero. Consequently, most of the information contained in $\mathbf{x}(k)$ is – in terms of variance – explained by projecting onto the subspace spanned by the first $p \leq L$ eigenvectors, i.e.,

$$\mathbf{z}(k) = \mathbf{C}\bar{\mathbf{x}}(k) \quad (4.29)$$

where \mathbf{C} is the $p \times L$ matrix:

$$\mathbf{C} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_p]^\top. \quad (4.30)$$

$z_j(k)$ is denoted the j 'th principal component, and Fig. 4.5 shows the construction in a two-dimensional case. The principal components are characterized by the following properties [Anderson 84, Ch. 11]:

- The principal components are mutually uncorrelated with zero mean and variances $V\{z_j(k)\} = \lambda_j$.
- $z_j(k)$ constitutes the normalized linear combination of the components in $\bar{\mathbf{x}}(k)$ with maximum variance under the constraint that $E\{z_j(k)z_s(k)\} = 0, \forall s \in [1, j-1]$.
- Selecting \mathbf{C} according to `na:copt` ensures an optimal projection onto a p -dimensional subspace in mean square sense. That is, perform a linear reconstruction of $\bar{\mathbf{x}}(k)$ by

$$\bar{\mathbf{x}}_{\text{rc}}(k) = \mathbf{W}\mathbf{z}(k) \quad (4.31)$$

where \mathbf{W} is a $L \times p$ matrix. Minimizing $E\{\|\bar{\mathbf{x}}_{\text{rc}}(k) - \bar{\mathbf{x}}(k)\|^2\}$ w.r.t. \mathbf{W} the minimal value is obtained by the choice `na:copt`.

The amount of retained information (variance) by projection onto the p -dimensional subspace is given by:

$$\delta = \frac{\sum_{j=1}^p \lambda_j}{L} = \frac{1}{L\sigma_x^2} \cdot \sum_{j=1}^p \lambda_j \quad (4.32)$$

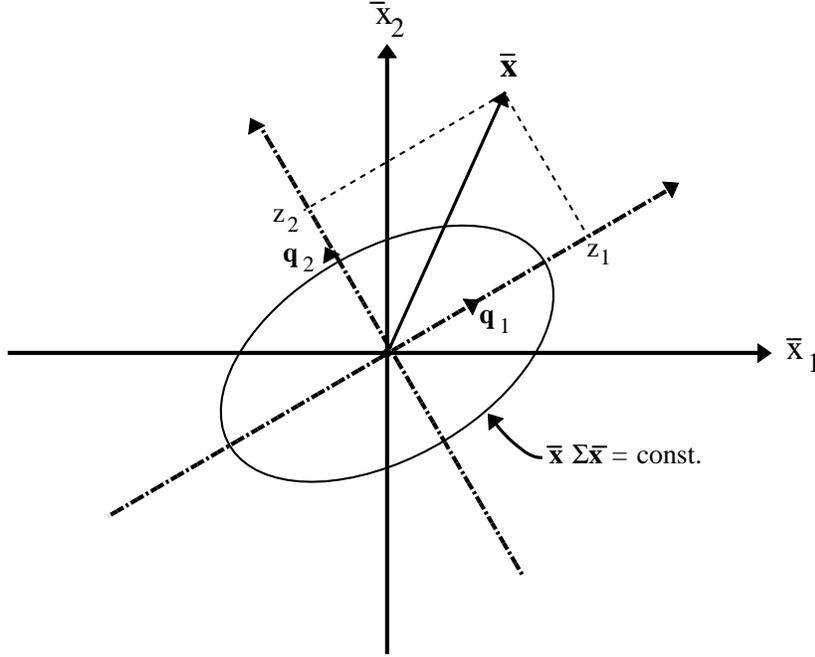


Figure 4.5: Estimating the preprocessed input vector $\mathbf{z}(k)$ by projecting the (zero-mean) input vector $\bar{\mathbf{x}}(k)$ onto the subspace spanned by the p normalized eigenvectors, \mathbf{q}_j corresponding to the largest eigenvalues.

where $\sigma_x^2 = \gamma_x(0)$ is the variance of $x(k)$. $0 \leq \delta \leq 1$ defines the total variance of the first p principal components relative to the total variance of the input signal vector. This enables an automatic determination of the dimension p . Specifying a minimal value of δ – e.g., $\delta_{\min} = 0.99$ – then p is estimated as the minimal integer satisfying: $\delta \geq \delta_{\min}$.

For practical purposes the PCA preprocessor is constructed by performing an eigenvalue decomposition of the estimated covariance matrix:

$$\hat{\Sigma}_x = \frac{1}{N} \sum_{k=1}^N \bar{\mathbf{x}}(k) \bar{\mathbf{x}}^\top(k) \quad (4.33)$$

where

$$\bar{\mathbf{x}}(k) = x(k) - \frac{1}{N} \sum_{k=1}^N x(k). \quad (4.34)$$

Assuming that $x(k)$ is an ergodic Gaussian distributed sequence then $\hat{\Sigma}_x$ is the maximum likelihood estimator of Σ_x . Furthermore, the eigenvalues and eigenvectors of $\hat{\Sigma}_x$ are maximum likelihood estimators of λ_j , \mathbf{q}_j , respectively, cf. [Anderson 84, Ch. 11]. The eigenvalue decomposition may be carried out by using standard numerical techniques (see e.g., [Madsen & Nielsen 75], [Press et al. 88]); however, the literature also provides methods for on-line estimation [Hertz et al. 91, Sec. 8.3], [Yu 91].

Let us summarize the PCA preprocessing by listing a number of advantages and drawbacks:

- The principal components, $z_j(k)$, are mutually uncorrelated. Furthermore, if we perform a variance normalization⁵ then the eigenvalue spread of $E\{\mathbf{z}(k)\mathbf{z}^\top(k)\}$ equals one. This possibly leads to convergence speed-up when applying first order algorithms for weight estimation, cf. Ch. 5.
- If the eigenvalue spread of Σ_x is large then using $p \ll L$ ensures that most information is retained. In consequence, the number of adjustable weights, \mathbf{w} , in the succeeding nonlinear filter is significantly reduced. On the other hand, if the eigenvalue spread is small, i.e. $x(k)$ is nearly a white signal, then PCA is not profitable.
- It is possible to obtain an estimate of a suitable dimension p by specifying the amount of variance to be explained.
- PCA is designed solely on knowledge of the input vector, i.e., even though it is possible to make a close reconstruction of $\bar{\mathbf{x}}(k)$ from $\mathbf{z}(k)$ there is no guarantee of retaining essential information concerning the output $y(k)$. For instance, if the overlap between the power spectra of $x(k)$ and $y(k)$ is small PCA is not reasonable. This is due to the fact that the spectrum of the first principal components mainly contains the dominating frequencies in $x(k)$. Then as memory normally is not present in the nonlinear filtering function, $f_n(\cdot)$, it becomes difficult to shift the spectra of z_j into the range which dominates the spectrum of $y(k)$. In Ch. 8 this matter is exemplified. However, simulations Ch. 8 indicate that when the input and output spectra resemble each other PCA seems practicable. In particular, this is the case when predicting time-series. In [Broomhead & King 87] the use of PCA is suggested in connection with time-series prediction.

4.4 Derivative Preprocessor

Another fixed FIR preprocessor is the *derivative preprocessor* (DPP) which is motivated by the fact that many physical systems are described by differential equations. Consider e.g., the dynamics of a pendulum: nonlinear differential equation:

$$\frac{d^2x(t)}{dt^2} + c\frac{dx(t)}{dt} + \left(\frac{2\pi}{P}\right)^2 \sin(x(t)) = y(t) \quad (4.35)$$

where $x(t)$ is the angle deflection, $y(t)$ is a driving force and c, P are constants. Let $D^i[x(k)]$ be a discrete i 'th derivative operator⁶, then by sampling na:pendul

$$D^2[x(k)] + cD[x(k)] + \left(\frac{2\pi}{P}\right)^2 \sin(x(k)) = y(k). \quad (4.36)$$

The output $y(k)$ is then expressed as a function of only $p = 3$ variables by defining:

$$\mathbf{z}(k) = [x(k), D[x(k)], D^2[x(k)]]^\top. \quad (4.37)$$

In general, the DPP is defined by:

$$\mathbf{z}(k) = [x(k), D[x(k)], \dots, D^{p-1}[x(k)]]^\top. \quad (4.38)$$

⁵Recall that $V\{\lambda_j^{-1}z_j(k)\} = 1$.

⁶That is, the approximation of $d^i x(t)/dt$.

To ensure that the discrete derivative operator $D[\cdot]$ accurately approximates the continuous time derivative operator over the frequency range of interest it must be implemented using a linear filter with high order. In the case of a FIR implementation this implies large L , and consequently we often face $p \ll L$.

Let $H_x(f) = \text{Fou}\{x(t)\}$ be the spectrum of $x(t)$ then the spectrum of the derivative is given by:

$$\text{Fou}\{x'(t)\} = 2\pi j f \cdot H_x(f) = H_{cd}(f)H_x(f) \quad (4.39)$$

where $H_{cd}(f) = 2\pi j f$ is the transfer function of the continuous derivative filter. That the phase is constant equal to $\pi/2$ and the magnitude proportional to the frequency f . Regarding a digital implementation we face the fundamental bandlimiting given by the Nyquist frequency, $f_s/2$, where f_s is the sampling frequency. That is the optimal digital filter is given by

$$H_d(f) = \begin{cases} 2\pi j f & f < f_s/2 \\ 0 & \text{otherwise} \end{cases} \quad (4.40)$$

with impulse response

$$h_d(k) = \begin{cases} 0 & k = 0 \\ \frac{(-1)^k f_s}{k} & k \neq 0 \end{cases} \quad (4.41)$$

Omitting a gain factor πf_s we choose instead to implement:

$$h_d(k) = \begin{cases} 0 & k = 0 \\ \frac{(-1)^k}{k\pi} & k \neq 0 \end{cases} \leftrightarrow H_d(f_n) = 2j f_n, \quad 0 \leq f_n < 0.5 \quad (4.42)$$

where f_n is the normalized frequency, i.e., $f_n = f/f_s$. In the rest of the chapter the we omit the index n for convenience.

The implementation of $H_d(f)$ is afflicted with two obstacles:

- Amplification of noise on the input since the signal-to-noise often is low at high frequencies. This indeed limits p to a low number.
- The optimal filter is non-causal and infinite.

The first obstacle may be dealt with performing a noise reducing low-pass filtering (see [Carlsson et al. 91] for an optimal approach). Below we shall employ a simple approach which consists in restricting the derivative filter to be a linear function of the frequency in a certain frequency interval of interest, say $[0; f_0]$. The second obstacle is circumvented by considering a truncated impulse response. Furthermore, causal filtering is obtained by delaying $y(k)$, i.e., estimating $y(k - D_y)$, $D_y > 0$ using the samples $x(k), x(k - 1), \dots$.

The derivative filter may e.g., be designed by the Parks-McClelland equiripple FIR filter design procedure [Oppenheim & Schaffer 89, Sec. 7.6–7]. A design example is shown in Fig. 4.6. Design parameters are:

- The filter order, L_d .
- The frequency range of interest, $[0; f_0]$.
- Maximum relative deviation, α , defined according to:

$$\frac{|\widehat{H}_d(f)| - |H_d(f)|}{|H_d(f)|} < \alpha, \quad \forall f \in [0; f_0] \quad (4.43)$$

where $H_d(f)$ is the desired response while $\widehat{H}_d(f)$ is the estimated response.

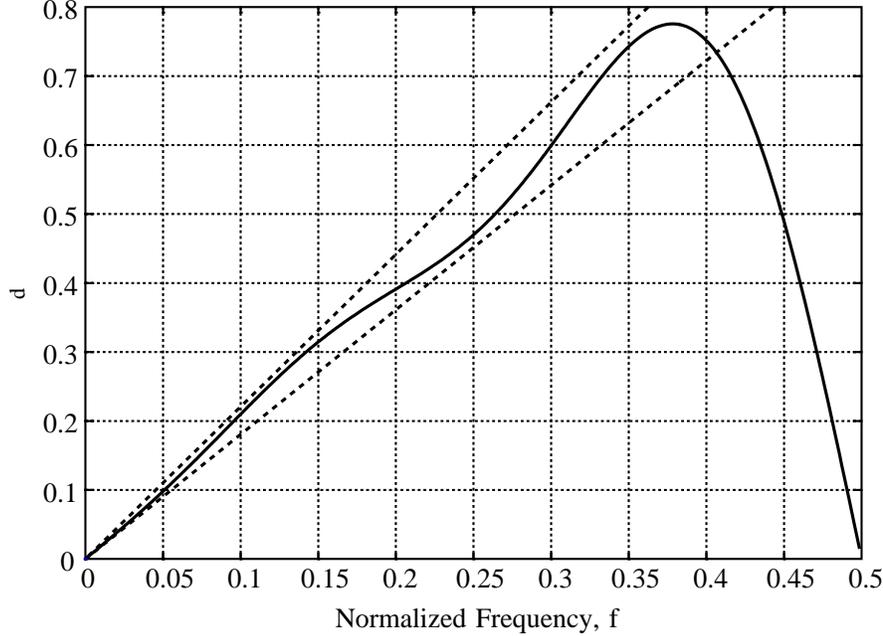


Figure 4.6: The magnitude of derivate filter estimated by using equiripple FIR filter design. The frequency range of interest is $[0; 0.4]$, the maximum relative deviation $\alpha = 0.1$ (shown by the dotted lines), and the filter order $L = 9$.

Higher order derivatives can be found by cascading first order derivative filters, i.e., $\forall i \in [1; p]$,

$$D^i[x(k)] = \underbrace{h_d(k) * h_d(k) * \dots * h_d(k)}_{i \text{ terms}} * x(k) \quad (4.44)$$

with corresponding transfer functions:

$$H_{d^i}(f) = \prod_{j=1}^i H_d(f). \quad (4.45)$$

According to [Hoffmann 92a] the relative deviation connected with $H_{d^i}(f)$ is approximately equal to $i\alpha$ if $\alpha \ll 2/(i-1)$. In Fig. 4.7 the final implementation of a second order DPP is shown. The shown delays serve a phase compensation among the individual derivative filters. Suppose that L_d is odd then the memory length equals the order of $h_{d^{p-1}}(k)$, i.e., cf. na:devcon: $L = (p-1)(L_d-1) + 1$. It turns out that the delay associated with $z_j(k)$ equals:

$$\frac{L - (j-1)(L_d-1) - 1}{2}.$$

4.5 Laguerre Function Preprocessor

In Ch. 3 as regards the discussion of the Wiener model we mentioned the use of Laguerre functions for a preprocessing of the input signal. Using the Laguerre functions as prepro-

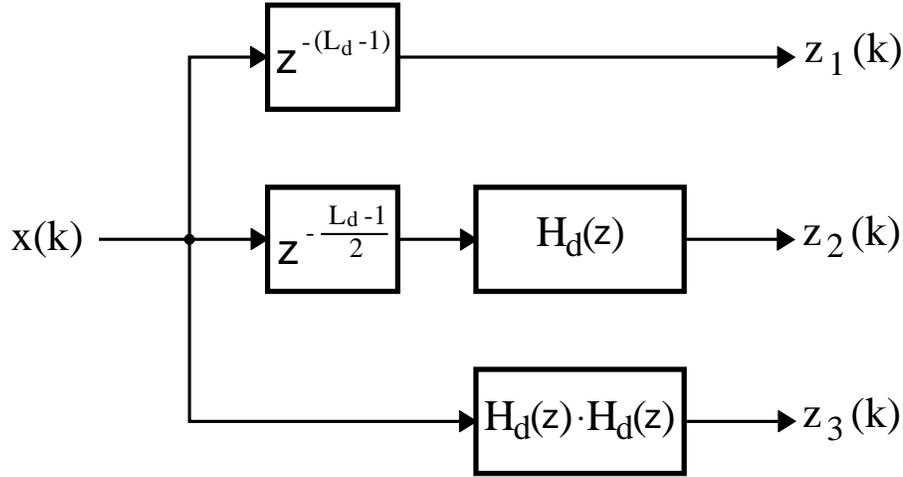


Figure 4.7: The second order derivative preprocessor.

cessing filters provides as a tool for handling infinite memory requirements, i.e., an IIR preprocessor. In Fig. 4.8 the construction of the $z_j(k)$ are shown. The transfer functions are given by⁷:

$$H_1(z) = \frac{\sqrt{1-\theta^2}}{1-\theta z^{-1}}, \quad (4.46)$$

$$H_j(z) = \frac{-\theta + z^{-1}}{1-\theta z^{-1}} H_{j-1}(z), \quad 2 \leq j \leq q \quad (4.47)$$

with $0 \leq \theta < 1$. $H_1(z)$ is a first order recursive low-pass filter, and the succeeding filters are obtained by constantly multiplying with the all-pass term: $-\theta + z^{-1}/1 - \theta z^{-1}$. The time-domain expression of $z_j(k)$ are given by:

$$z_1(k) = \theta z_1(k-1) + \sqrt{1-\theta^2} x(k), \quad (4.48)$$

$$z_j(k) = \theta z_j(k-1) - \theta z_{j-1}(k) + z_{j-1}(k-1), \quad 2 \leq j \leq q. \quad (4.49)$$

The choice of the parameter θ is not obvious even though it provides for a specification the memory length. Consequently, it may be regarded as an adjustable parameter which is determined according to the overall cost function. That is, the preprocessor becomes adaptive. In [Hoffmann 92a, Sec. 3.5.2] a scheme for adapting θ is considered.

The Gamma Delay Network suggested in [Principe et al. 92], [de Vries et al. 91] may be viewed as an IIR preprocessor with resembles the Laguerre function preprocessor. The

⁷Note that the Laguerre function preprocessor degenerates to a simple tapped delay line for $\theta = 0$.

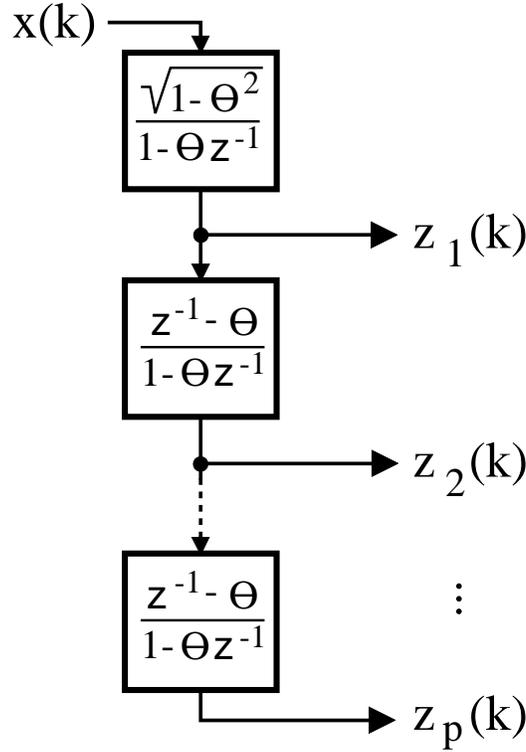


Figure 4.8: The Laguerre function preprocessor.

time-domain expression of $z_j(k)$ are given by⁸:

$$z_1(k) = x(k), \quad (4.50)$$

$$z_j(k) = \theta z_j(k-1) + (1-\theta)z_{j-1}(k-1), \quad 2 \leq j \leq q. \quad (4.51)$$

with $0 \leq \theta < 1$. That is, the transfer functions $H_j(z)$ appear by successive multiplication with a first order recursive low-pass filter, i.e.,

$$H_1(z) = 1, \quad (4.52)$$

$$H_j(z) = \frac{(1-\theta)z^{-1}}{1-\theta z^{-1}} H_{j-1}(z), \quad 2 \leq j \leq q. \quad (4.53)$$

4.6 Summary

In this chapter we presented the generic neural network architecture which may be viewed as a heterogeneous three-layer neural network: The first layer is a preprocessing unit, the second layer defines a nonlinear filtering, and the third layer consists of a linear output neuron. The second and third layer constitute a canonical filter representation. Various implementations of the canonical filter are already discussed in Ch. 3 so here we focus on the preprocessing unit which serves several purposes:

⁸Notice, when $\theta = 0$ the Gamma Delay preprocessor degenerates to a tapped delay line.

- Ensuring that the dimension of the preprocessed input vector $\mathbf{z}(k)$ is kept small so that the “curse of dimensionality” is circumvented. This is certainly one of the most important properties of the preprocessor.
- Extracting the essential information contained in $x(k)$ w.r.t. the output $y(k)$.
- Phase compensation so that the optimal filtering (from $x(k)$ to $y(k)$) is ensured to be causal.

We designed a preprocessing algorithm (PPA) aiming at implementing a linear, fixed FIR preprocessor. In addition several preprocessing methods were discussed. These comprise the principal component analysis (PCA), the derivative preprocessor (DPP), and the Laguerre function preprocessor.

CHAPTER 5

ALGORITHMS FOR FILTER PARAMETER ESTIMATION

The topic of this chapter is algorithms for estimation of filter parameters (weights). The weights are estimated by minimizing some cost function which measures the performance of the filter. The cost function depends on the current set of weights and the set of related input-output data – the so-called training set. A discussion of the choice of the cost function and aspects of the optimization task is provided. Principally, we focus on the Least Squares (LS) cost function with a weight decay regularizer.

Various first and second order parameter estimation algorithms are presented. In particular, we will derive algorithms for layered filter architectures such as the multi-layer feed-forward perceptron neural network (MFPNN). The crucial part is the calculation of the gradient of the filter output w.r.t. the weight vector which is done by using the *back-propagation* (BP) algorithm. The BP-algorithm is derived for general layered feed-forward filter architectures, and furthermore, a simple matrix formulation is provided in the case of the MFPNN architecture.

In addition, we elaborate on different matters concerning the algorithms. This includes: Initialization, convergence, and computational complexity. Especially, a weight initialization procedure for MFPNN's aiming at improving the speed of convergence is presented.

It should be mentioned that Ch. 8 provides some comparative numerical studies of the algorithms presented in this chapter.

5.1 Parameter Estimation

Define the data generating system by

$$y(k) = g(\mathbf{x}(k)) + \varepsilon(k) \quad (5.1)$$

where

- $y(k)$ is the output signal.
- $g(\cdot)$ constitutes a (nonlinear) unknown¹ mapping.

¹In certain cases some a priori knowledge concerning the structure and $g(\cdot)$ may be available; however, we will treat the case of “black-box” modeling.

- $\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-L+1)]^\top$ is the L -dimensional input vector signal.
- $\varepsilon(k)$ is the inherent zero-mean (non-observable) stochastic noise which explains lack of information about $y(k)$.

Recapitulate the non-recursive model with additive error:

$$\begin{aligned} y(k) &= f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \\ &= \hat{y}(k) + e(k; \mathbf{w}) \end{aligned} \quad (5.2)$$

where

- $\hat{y}(k)$ is the filter output.
- \mathbf{w} is the m -dimensional weight (parameter) vector.
- $e(k; \mathbf{w})$ is the error signal.

Suppose that a set of related input-output data has been collected, and define the *training set*:

$$\mathcal{T} = \{\mathbf{x}(k); y(k)\}, \quad k = 1, 2, \dots, N \quad (5.3)$$

where N is the size of the training set. The task is then to fit a model – i.e., estimate the weight vector – from the available training data. The literature provides various frameworks for weight estimation (see e.g., [Seber & Wild 89, Ch. 2]) such as Least Squares (LS) estimation, Maximum Likelihood estimation (ML), and Bayesian estimation. The weight estimation problem is generally formulated as a minimization task. Define the *cost function*

$$C_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N c(k; \mathbf{w}) \quad (5.4)$$

where $c(k; \mathbf{w}) = \text{dist}(y(k), \hat{y}(k)) \geq 0$ is denoted the instant cost. The instant cost measures the distance between the output $y(k)$ and the filter output $\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w})$. Hence, the weights are estimated by minimizing the cost function.

Let Ω be a closed, bounded set (i.e., a compact set) in weight space over which the cost, $C_N(\mathbf{w})$, is minimized. Define \mathcal{W} as the set of weight vectors which locally minimize $C_N(\mathbf{w})$, i.e., let $\dim(\mathcal{W}) = \dim(\mathbf{w})$ then

$$\mathcal{W} = \{\mathbf{w} \in \Omega \mid \exists \delta > 0, \forall \|\boldsymbol{\theta}\| < \delta, C_N(\mathbf{w} + \boldsymbol{\theta}) \geq C_N(\mathbf{w})\} \quad (5.5)$$

where $\|\cdot\|$ denotes a vector norm. A particular vector in the set is referred to as the *estimated weight vector* and denoted by $\hat{\mathbf{w}}$. In Fig. 5.1 the general shape of the cost function is shown. Usually it is difficult to characterize the cost function in detail since it depends both on the system and the model; however, notice some general facts:

1. \mathcal{W} contains in general local minima. If we deal with an LX-model normally only one minimum exist (see also Sec. 5.2). Recall that the parameterization within MFPNN (cf. Sec. 3.2.2) is ambiguous. This gives rise to several local minima with equal cost, i.e., they are really not intrinsic local minima. On the other hand, there may still be intrinsic local minima [Hect-Nielsen 89].

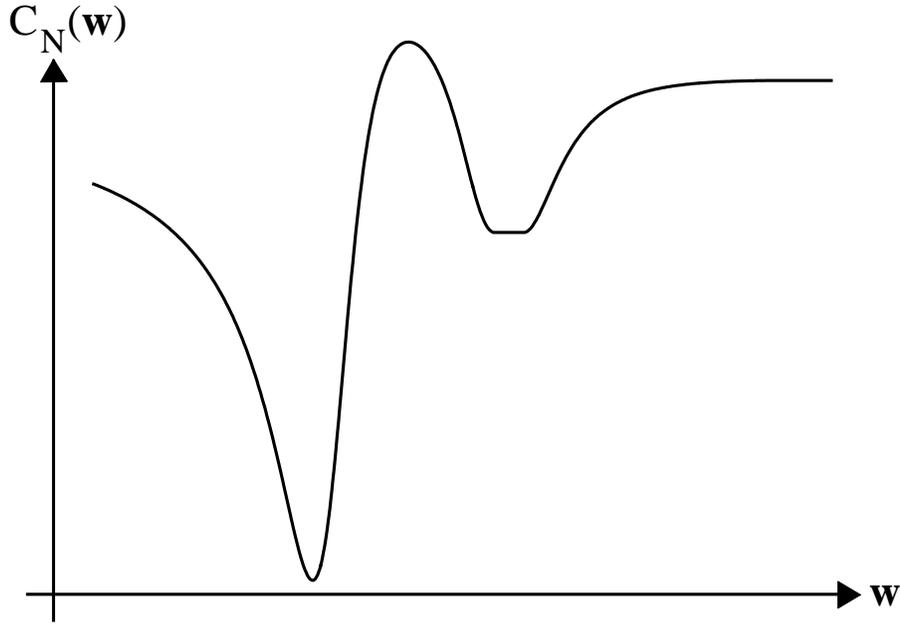


Figure 5.1: General shape of the cost function. Notice the existence of local minima and that a minimum can be non-unique.

2. The minima may be non-unique (i.e., the cost function is “flat”). In connection with classification problems this situation may occur. Consider e.g., using a single perceptron neural network (cf. Sec. 3.2.2) for classifying two distinct classes. Consequently, the decision boundary (the hyperplane) specified by the weights has no unique placement. Within filtering tasks the cost function may be almost “flat” in the minimum point in some directions of the weight space due to the fact that the filter is over-parameterized.

In addition, if the size of the training set, N , is small compared to the dimension, m , of the weight space then “flat” cost functions may occur. In particular, since each training sample specifies a restriction in the weight space then if $N < m$ then the minimum is definitely non-unique.

3. Dealing with a MFPNN the cost function contains “flat” parts due to degeneration as mentioned in Sec. 3.2.2. Degeneration occurs typically when the weights are small and large. A perceptron with small weights effectively acts like a linear neuron; consequently, the cost function becomes “flat” in a single direction (see Sec. 3.2.2). When the weights of a neuron are large then the perceptron saturates. That means, the filter output – and thereby the cost function – is almost independent of the weight values.
4. The curvature (the second order derivative w.r.t. \mathbf{w}) of the cost-function near a minimum may vanish, e.g., $C_N(w) \propto (w - \hat{w})^4$.

Also define the *expected cost function*

$$C(\mathbf{w}) = E_{\mathbf{x},\varepsilon}\{c(\mathbf{w})\} \quad (5.6)$$

where $c(\mathbf{w}) = \text{dist}(y, \hat{y})$ and $E_{\mathbf{x},\varepsilon}\{\cdot\}$ denote expectation w.r.t. the joint probability density function (p.d.f.) of $[\mathbf{x}, \varepsilon]^2$. The expected cost convey the average cost – or the cost when employing an infinite training set – since

$$\lim_{N \rightarrow \infty} \{C_N(\mathbf{w})\} = C(\mathbf{w}), \quad (5.7)$$

provided that $c(k; \mathbf{w})$ is a mean-ergodic sequence, i.e., cf. [Papoulis 84a, Ch. 9-5]

$$E_{\mathbf{x}(k),\varepsilon(k)}\{c^2(k; \mathbf{w})\} < \infty, \quad (5.8)$$

and

$$E_{\mathbf{x}(k),\varepsilon(k)}\{c(k; \mathbf{w})c(k + \tau; \mathbf{w})\} \rightarrow 0, \text{ as } \tau \rightarrow \infty. \quad (5.9)$$

The optimal weight vector set, \mathcal{W}^* , is the set of weight vectors which locally minimizes the expected cost function, i.e., let $\dim(\boldsymbol{\theta}) = \dim(\mathbf{w})$ then

$$\mathcal{W}^* = \{\mathbf{w} \in \Omega \mid \exists \delta > 0, \forall \|\boldsymbol{\theta}\| < \delta, C(\mathbf{w} + \boldsymbol{\theta}) \geq C(\mathbf{w})\} \quad (5.10)$$

where $\|\cdot\|$ denotes a vector norm. A particular member, \mathbf{w}^* , is denoted the *optimal weight vector* and reflects the best attainable model with the chosen filter architecture and distance measure, $\text{dist}(y, \hat{y})$.

5.1.1 Consistency

DEFINITION 5.1 *The estimator $\hat{\mathbf{w}}$ is a strongly consistent estimator of \mathbf{w}^* if*

$$\text{Prob}\{\hat{\mathbf{w}} \rightarrow \mathbf{w}^*\} = 1, \text{ as } N \rightarrow \infty \quad (5.11)$$

where $\text{Prob}\{\cdot\}$ denotes probability.

In [Seber & Wild 89, Ch. 12] various assumptions leading to strong consistency are given. However, we will apply the result of [White 81, Theorem 2.1] because it is applicable for incomplete models, i.e., cf. Def. 6.3, models which are not able to model the data perfectly, and secondly, for general type of cost functions. The necessary assumptions are:

ASSUMPTION 5.1 *The input $\mathbf{x}(k)$ and the inherent noise $\varepsilon(k)$ are assumed to be strictly stationary sequences.*

ASSUMPTION 5.2 *Assume that there exists a covering of Ω in compact subsets (in general different from that in sa:lssplit):*

$$\Omega = \bigcup_i \Omega^*(i) \quad (5.12)$$

such that $\mathbf{w}^(i) \in \Omega^*(i)$ uniquely minimizes $C(\mathbf{w})$ within the the partition $\Omega^*(i)$*

ASSUMPTION 5.3

²Note that the input vector and the inherent noise enters $c(\cdot)$ through pa:system, (5.2). Furthermore, the input vector signal is considered as a stochastic signal.

1. The instant cost is assumed to be a continuous function of \mathbf{w} on a compact set and a measurable function of \mathbf{x} and ε , $\forall \mathbf{w}$.
2. Suppose the existence of a function $\zeta(\mathbf{x}, \varepsilon)$ which complies with

$$E_{\mathbf{x}, \varepsilon} \{ \zeta(\mathbf{x}, \varepsilon) \} < \infty \quad (5.13)$$

so that $c(\mathbf{w}) \leq \zeta(\mathbf{x}, \varepsilon)$, $\forall \mathbf{x}, \varepsilon, \mathbf{w}$.

3. Every data sample in the training set is assumed to be a random sample of the joint distribution of $[\mathbf{x}(k), \varepsilon(k)]$.

THEOREM 5.1 *Suppose that As. 5.1, 5.2, 5.3 hold and recall that $\mathbf{w}^*(i)$ uniquely minimizes $C(\mathbf{w})$ within the compact subset $\Omega^*(i)$. If $\hat{\mathbf{w}}$ minimizes $C_N(\mathbf{w})$ within $\Omega^*(i)$ then $\hat{\mathbf{w}}$ is a strongly consistent estimator of $\mathbf{w}^*(i)$ as $N \rightarrow \infty$.*

PROOF See Theorem 2.1 of [White 81]. ■

It is possible to relax As. 5.2 in order to accomplish the case of non-unique minimizers. In that case the strong consistency is [Ljung 87, Sec. 8.3], [White 89a] formulated as:

$$\text{Prob} \left\{ \inf_{\mathbf{w}^* \in \mathcal{W}^*} \|\hat{\mathbf{w}} - \mathbf{w}^*\| \rightarrow 0 \right\} = 1, \text{ as } N \rightarrow \infty. \quad (5.14)$$

5.1.2 Least Squares Estimation

The most common cost function is the Least Squares (LS) cost function

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N e^2(k; \mathbf{w}) \quad (5.15)$$

which mainly is dealt with in this Thesis. However, several modifications may be useful in practical applications. This includes:

- Weighting.
- Robust Estimation.
- Regularization.

5.1.2.1 Weighted LS Estimation

Define the error vector

$$\mathbf{e} = [e(1; \mathbf{w}), e(2; \mathbf{w}), \dots, e(N; \mathbf{w})]^\top. \quad (5.16)$$

The weighted LS cost function is then given by:

$$C_N(\mathbf{w}) = \mathbf{e}^\top \mathbf{S} \mathbf{e} \quad (5.17)$$

where $\mathbf{S} = \{s_{ij}\}$ is the weighting matrix. Note that when $\mathbf{S} = \mathbf{I}$ the weighted LS cost function coincides with the usual LS cost function. In Sec. 5.6 a weighting is employed in order to ensure proper algorithm properties; however, also theoretical considerations may

lead to a preference of the weighted LS cost function. In general the optimal error signal, $e(k; \mathbf{w}^*)$ is a correlated sequence. This is seen by substituting pa:system into pa:model, i.e.,

$$e(k; \mathbf{w}^*) = \varepsilon(k) + g(\mathbf{x}(k)) - f(\mathbf{x}(k); \mathbf{w}^*). \quad (5.18)$$

The correlation thus stems from:

- Correlation in the inherent noise $\varepsilon(k)$.
- Employing an incomplete model which means that $g(\mathbf{x}(k)) \neq f(\mathbf{x}(k); \mathbf{w}^*)$, and consequently the correlation of input signal $\mathbf{x}(k)$ comes through.

Strong correlation in the error signal may result in very inefficient LS-estimates [Seber & Wild 89, Ch. 6]. A consequence is that the covariance matrix of (fluctuations in) the weight vector estimate becomes rather larger and thereby increasing the generalization error (see further Ch. 6). In order to remedy this inexpedience one may introduce the weighted LS estimator, cf. [Seber & Wild 89, Sec. 2.1.4]. The choice $\mathbf{S} = \boldsymbol{\Sigma}_e^{-1}$ where $\boldsymbol{\Sigma}_e = V\{\mathbf{e}\mathbf{e}^\top\}$ is the positive definite covariance matrix of the error is optimal with respect to minimizing the variance of the weight estimates [Ljung & Söderström 83, p. 87]. Obviously, if the error signal is white, i.e., $\boldsymbol{\Sigma}_e \propto \mathbf{I}$, then an optimal choice is the usual LS cost function. The correlation matrix is, in principle, required to be known in advance; however, this is rarely the case, and consequently one may rely on suboptimal methods such as iterated two-stage estimation [Seber & Wild 89, Sec. 6.2.4] which runs as follows:

1. Initially set $\boldsymbol{\Sigma}_e = \mathbf{I}$.
2. Perform a weighted LS estimation³.
3. Calculate the error signal with the current weight estimates. If the cost function has not decreased significantly then stop; otherwise, calculate the correlation matrix and go to step 2.

5.1.2.2 Robust Estimation

Robust estimation [Seber & Wild 89, Sec. 2.6] can be important in cases where outliers are present. That is, if atypical samples with very high squared error are present then the weights principally are adjusted in order to compensate these errors, and thus the error done on the typical samples is increased. Robust estimation is e.g., done by letting

$$c(k; \mathbf{w}) = e^2(k; \mathbf{w}) \cdot \varphi(e^2(k; \mathbf{w})) \quad (5.19)$$

where $\varphi(\cdot)$ is a function which penalizes large errors.

5.1.2.3 Regularization

The regularized LS cost function is given by

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa R_N(\mathbf{w}) \quad (5.20)$$

³The weighted LS estimation problem can be transformed into a standard LS problem [Seber & Wild 89, Sec. 2.1.4] thus the algorithms given in the sections below are applicable.

where

$$R_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N r(k; \mathbf{w}) \quad (5.21)$$

is the regularization term⁴ which generally depends on the training set, and $\kappa \geq 0$ is the regularization parameter which determines the trade off between the LS cost and the regularization term. The purpose of regularization is to impose some constraints on the minimization problem in order to accomplish:

- Numerical robustness. That is, a degenerated cost function which contains “flat” parts should be avoided.
- The quality of the filter architecture expressed by the generalization ability may be increased, see Sec. 6.9.

The regularization term typically conveys some a priori knowledge on the model e.g., the location of the optimal weight vector in the weight space. A common regularization term of this kind is the *weight decay regularizer*⁵ e.g., [Moody 91], [Krogh & Hertz 91]

$$R_N(\mathbf{w}) = r(\mathbf{w}) = \mathbf{w}^\top \mathbf{w} \quad (5.22)$$

which penalizes large weights. Different schemes have been suggested in e.g., [Nowland & Hinton 92], [Weigend et al. 90]. Another type of regularization which penalizes high input sensitivity [Drucker & Le Cun 91], [Geman et al. 92], [Moody 91] is e.g., given by:

$$R_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N \left| \frac{\partial f(\mathbf{x}(k); \mathbf{w})}{\partial \mathbf{x}(k)} \right|^2. \quad (5.23)$$

5.1.3 Maximum Likelihood Estimation

Within the ML approach the cost function is given by the negative log-likelihood function, i.e.,

$$C_N(\mathbf{w}) = -\frac{1}{N} \ln L_N(\mathbf{w}) \quad (5.24)$$

where \ln is the natural logarithm and $L_N(\cdot)$ is the likelihood function

$$L_N(\mathbf{w}) = p_y(y(1), \dots, y(N) | \mathbf{x}(1), \dots, \mathbf{x}(N); \mathbf{w}) \quad (5.25)$$

Here $p_y(\cdot | \cdot)$ denotes the joint conditional p.d.f. of all $y(k)$ in the training set conditioned on the inputs and the parameters. Using pa:model we get the rewriting

$$\begin{aligned} p_y(y | \mathbf{x}; \mathbf{w}) &= p_y(e + f(\mathbf{x}; \mathbf{w}) | \mathbf{x}; \mathbf{w}) \\ &= p_e(e | \mathbf{x}; \mathbf{w}). \end{aligned} \quad (5.26)$$

Consequently, the likelihood function becomes:

$$L_N(\mathbf{w}) = p_e(e(1), \dots, e(N) | \mathbf{x}(1), \dots, \mathbf{x}(N); \mathbf{w}). \quad (5.27)$$

⁴Normally we choose $R_N(\mathbf{w}) > 0$.

⁵This kind of regularization is in the statistical literature known as ridge regression.

The ML approach thus requires knowledge of the error distribution. A common assumption is that $e(k)$ is an i.i.d.⁶ Gaussian distributed sequence with zero mean and unknown variance, σ_e^2 . The variance thus acts like an extra parameter or weight. The likelihood function is in this case⁷:

$$\begin{aligned} L_N(\mathbf{w}; \sigma_e^2) &= \prod_{k=1}^N \frac{1}{\sqrt{2\pi\sigma_e}} \exp\left(-\frac{1}{2\sigma_e^2} e^2(k; \mathbf{w})\right) \\ &= (2\pi\sigma_e^2)^{-\frac{N}{2}} \exp\left(-\frac{1}{2\sigma_e^2} \sum_{k=1}^N e^2(k; \mathbf{w})\right). \end{aligned} \quad (5.28)$$

Now minimization of the cost function results in [Seber & Wild 89, Sec. 2.2]

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} S_N(\mathbf{w}), \quad (5.29)$$

$$\hat{\sigma}_e^2 = S_N(\hat{\mathbf{w}}). \quad (5.30)$$

That is, the ML-estimate of the weights equals the LS-estimate.

In practice, the the error signal is neither i.i.d. nor Gaussian distributed which makes the ML approach awkward to handle. Consequently, one may often rely on formulating a suitable cost function such as the LS cost – or a modified version.

5.2 Performing Least Squares Estimation

In the rest of this Thesis we will mainly focus on the the LS cost function with a weight decay regularizer, i.e.,

$$\begin{aligned} C_N(\mathbf{w}) &= S_N(\mathbf{w}) + \kappa \mathbf{w}^\top \mathbf{w} \\ &= \frac{1}{N} \sum_{k=1}^N e^2(k; \mathbf{w}) + \kappa \mathbf{w}^\top \mathbf{w}. \end{aligned} \quad (5.31)$$

The ultimate goal is to devise an algorithm which *globally* minimizes the cost function over the weight space Ω . However, usually global minimization involves a tremendous amount of computations; consequently, we are content with *local minimization* schemes. A local minimizer

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w}) \quad (5.32)$$

is found by solving the normal equations:

$$\nabla_N(\hat{\mathbf{w}}) = \mathbf{0} \quad (5.33)$$

where

$$\begin{aligned} \nabla_N(\mathbf{w}) &= \frac{1}{2} \cdot \frac{\partial C_N(\mathbf{w})}{\partial \mathbf{w}} \\ &= \frac{1}{2} \left[\frac{\partial C_N(\mathbf{w})}{\partial w_1}, \frac{\partial C_N(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial C_N(\mathbf{w})}{\partial w_m} \right]^\top \end{aligned} \quad (5.34)$$

is the *gradient of the cost function*.

⁶Independent identically distributed.

⁷Note that the dependence on σ_e^2 is explicitly emphasized.

All algorithms presented in this chapter are in principle local optimization algorithms although some of them have the ability to escape from bad local minima and thus displaying some global optimization like properties. The issues concerning global optimization will not be treated further in this Thesis, instead the reader is referred to [Törn & Žilinskas 88].

Introducing the instantaneous gradient vector of the filter output,

$$\boldsymbol{\psi}(k; \mathbf{w}) = \frac{\partial f(\mathbf{x}(k); \mathbf{w})}{\partial \mathbf{w}}, \quad (5.35)$$

the gradient yields:

$$\begin{aligned} \nabla(\mathbf{w}) &= \frac{1}{N} \sum_{k=1}^N \frac{\partial e(k; \mathbf{w})}{\partial \mathbf{w}} e(k; \mathbf{w}) + \kappa \mathbf{w} \\ &= -\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}) e(k; \mathbf{w}) + \kappa \mathbf{w}. \end{aligned} \quad (5.36)$$

Consequently, the normal equations obey:

$$-\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) + \kappa \hat{\mathbf{w}} = \mathbf{0}. \quad (5.37)$$

A property of the time-average of the error signal can be deduced from these equations. Suppose that the model in pa:model contains a bias term, say w_0 , i.e.,

$$f(\mathbf{x}(k); \mathbf{w}) = f_1(\mathbf{x}(k); \mathbf{w}_1) + w_0 \quad (5.38)$$

where $\mathbf{w} = [\mathbf{w}_1^\top, w_0]^\top$. That is, the last component of $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$ equals

$$\frac{\partial f(\mathbf{x}(k); \hat{\mathbf{w}})}{\partial w_0} = 1, \quad (5.39)$$

and consequently, the last normal equation becomes:

$$\frac{1}{N} \sum_{k=1}^N e(k; \hat{\mathbf{w}}) = \kappa \hat{w}_0. \quad (5.40)$$

As a consequence, if regularization is not employed then the time-average of the error signal equals zero.

In general, it is not possible to solve the normal equations analytically, and consequently we are content with various numerical methods which is the topic of the succeeding sections. However, when the model is linear in the weights, i.e., of type LX, an analytical solution is possible. Consider the LX-model

$$\begin{aligned} y(k) &= f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \\ &\triangleq \mathbf{w}^\top \mathbf{z}(k) + e(k; \mathbf{w}) \end{aligned} \quad (5.41)$$

where $\mathbf{z}(k) = \boldsymbol{\varphi}(\mathbf{x}(k))$ and $\boldsymbol{\varphi}(\cdot)$ is an arbitrary time invariant linear or nonlinear mapping. In this case

$$\boldsymbol{\psi}(k; \mathbf{w}) = \mathbf{z}(k) \quad (5.42)$$

which is independent on \mathbf{w} . The cost function is in this case given by:

$$C_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N \left(y(k) - \mathbf{w}^\top \mathbf{z}(k) \right)^2 + \kappa \mathbf{w}^\top \mathbf{w}. \quad (5.43)$$

Obviously, the cost function is *quadratic* in the weights. The normal equation becomes:

$$\begin{aligned} -\frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) e(k; \hat{\mathbf{w}}) + \kappa \hat{\mathbf{w}} &= \mathbf{0} \\ -\frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \left(y(k) - \mathbf{z}^\top(k) \hat{\mathbf{w}} \right) + \kappa \hat{\mathbf{w}} &= \mathbf{0} \\ \left(\frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) + \kappa \mathbf{I} \right) \hat{\mathbf{w}} &= \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) y(k). \end{aligned} \quad (5.44)$$

Now define the symmetric Hessian matrix of the cost function: by

$$\begin{aligned} \mathbf{J}_N &= \frac{1}{2} \cdot \frac{\partial^2 C_N(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\ &= \frac{1}{2} \cdot \begin{bmatrix} \frac{\partial^2 C_N(\mathbf{w})}{\partial w_1^2} & \cdots & \frac{\partial^2 C_N(\mathbf{w})}{\partial w_1 \partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 C_N(\mathbf{w})}{\partial w_m \partial w_1} & \cdots & \frac{\partial^2 C_N(\mathbf{w})}{\partial w_m^2} \end{bmatrix} \\ &= \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) + \kappa \mathbf{I} \\ &= \mathbf{H}_N + \kappa \mathbf{I} \end{aligned} \quad (5.45)$$

where \mathbf{H}_N is the Hessian matrix when employing $\kappa = 0$ corresponding to the usual LS cost function.

Assuming that the Hessian is positive definite⁸ the weight estimate cf. pa:normeqlin is *unique*, and given by:

$$\hat{\mathbf{w}} = \mathbf{J}_N^{-1} \cdot \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) y(k). \quad (5.46)$$

Often it is profitable to formulate an $N \times m$ input data matrix, \mathbf{Z} , and an $N \times 1$ output data vector \mathbf{y} as follows:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}^\top(1) \\ \vdots \\ \mathbf{z}^\top(N) \end{bmatrix}, \quad \mathbf{y} = [y(1), \dots, y(N)]^\top. \quad (5.47)$$

⁸This can – if necessary – be ensured by setting $\kappa > \lambda_{\min}$ where λ_{\min} is the smallest (possibly zero) eigenvalue of \mathbf{H}_N . Furthermore, note when $\kappa = 0$, the Hessian is certainly singular for $N < m$, $m = \dim\{\mathbf{w}\}$ since the rank equals N .

Thus the weight estimate `pa:wdestlin` reads:

$$\hat{\mathbf{w}} = \left(\mathbf{Z}^\top \mathbf{Z} + \kappa \mathbf{I} \right)^{-1} \mathbf{Z}^\top \mathbf{y}. \quad (5.48)$$

In connection with orthogonal architectures (see Ch. 3), which are linear in the weights, the Hessian possesses a block diagonal form, and consequently, subvectors of the weight vector are estimated independently of each other.

The expression for the cost function `pa:costlin` is – in terms of the estimated weight vector – rewritten as:

$$C_N(\mathbf{w}) = C_N(\hat{\mathbf{w}}) + \delta \mathbf{w}^\top \mathbf{J}_N \delta \mathbf{w} \quad (5.49)$$

where $\delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}$.

The optimal weight vector – i.e., $\mathbf{w}^* = \arg \min_{\mathbf{w}} C(\mathbf{w})$ – yields analogous to `pa:wdestlin`:

$$\mathbf{w}^* = \mathbf{J}^{-1} E \{ \mathbf{z} \mathbf{y} \} \quad (5.50)$$

where \mathbf{J} is the Hessian of $C(\mathbf{w})$, i.e.,

$$\mathbf{J} = E \left\{ \mathbf{z} \mathbf{z}^\top \right\} + \kappa \mathbf{I} = \mathbf{H} + \kappa \mathbf{I}. \quad (5.51)$$

The literature provides numerous numerical algorithms for solving the normal equations. The reader is referred to [Battiti 92], [Ljung 87], and [Seber & Wild 89] for excellent presentations. Below first and second order algorithms are presented, and – in particular – we focus on layered filter architectures such as the MFPNN. The algorithms can be divided into two classes: Off-line algorithms and recursive algorithms. Off-line algorithms – which also are denoted block or batch algorithms – have the following general iterative structure:

$$\mathbf{w}_{(i)} = \mathbf{w}_{(i-1)} + \Delta \mathbf{w}_{(i-1)} \quad (5.52)$$

where $\mathbf{w}_{(i)}$ is the weight estimate at the i 'th iteration and $\Delta \mathbf{w}_{(i-1)}$ the weight update – or search direction – given by:

$$\Delta \mathbf{w}_{(i-1)} = \varphi \left(\mathcal{T}; \mathbf{w}_{(i-1)} \right). \quad (5.53)$$

$\varphi(\cdot)$ is some function of the training set⁹, $\mathcal{T} = \{ \mathbf{x}(k); y(k) \}$, $k \in [1; N]$, and the previous weight estimate, $\mathbf{w}_{(i-1)}$.

On the other hand, recursive algorithms – also denoted on-line algorithms – have the general recursive structure:

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \Delta \mathbf{w}(k-1) \quad (5.54)$$

where $\mathbf{w}(k)$ is the weight estimate at the current time instant k and $\Delta \mathbf{w}(k-1)$ is the weight update given by:

$$\Delta \mathbf{w}(k-1) = \varphi \left(\mathbf{x}(k), y(k); \mathbf{w}(k-1) \right). \quad (5.55)$$

$\varphi(\cdot)$ is a function of the actual training sample $\{ \mathbf{x}(k); y(k) \}$ and the weight estimate at the previous time step. That is, the recursive algorithm performs one weight update per time step. Recursive algorithms can be operated in two different modes:

⁹Sometimes different subsets of the training set are used in the weight update.

1. The *on-line mode* in which a novel training sample is added each time step, i.e.,

$$\mathbf{w}(k) = \mathbf{w}(k-1)\varphi(\mathbf{x}(k), y(k); \mathbf{w}(k-1)), \quad k = 1, 2, \dots \quad (5.56)$$

Consequently, the size of the training set gradually increases. The on-line mode is used when training data are abundant and in connection with tracking of time-varying systems (non-stationary environments). Usually this mode is considered in the signal processing literature.

2. The *off-line mode* which deals with a limited number of (stationary) training data assembled in the training set $\mathcal{T} = \{\mathbf{x}(k); y(k)\}$, $k \in [1; N]$. In this case we may run through the training data several times. Define the number of iterations *itr* as the maximum number of training set replications. Furthermore, define a running time index ℓ ,

$$\ell = k + (i-1)N \quad (5.57)$$

where k refers to the actual training sample $\{\mathbf{x}(k); y(k)\}$ and $i = 1, 2, \dots, itr$ counts the number of training set replications. Now, the recursion becomes:

$$\mathbf{w}(\ell) = \mathbf{w}(\ell-1)\varphi(\mathbf{x}(k), y(k); \mathbf{w}(\ell-1)). \quad (5.58)$$

In principle, one may wonder why a recursive algorithm is used for the purpose of achieving an off-line estimate; however, recursive algorithms seem to be better than off-line algorithms in escaping from bad local minima of the cost function. This is further elaborated below.

5.3 Gradient Descent Algorithm

In the off-line method of gradient descent (GD) the weight update is given by:

$$\mathbf{w}_{(i)} = \mathbf{w}_{(i-1)} - \mu_{(i)} \nabla_N(\mathbf{w}_{(i-1)}) \quad (5.59)$$

where

- The subscript $_{(i)}$ denotes the iteration number.
- $\nabla_N(\mathbf{w})$ is the gradient of the cost function:

$$\nabla_N(\mathbf{w}) = \frac{1}{2} \cdot \frac{\partial C_N(\mathbf{w})}{\partial \mathbf{w}}. \quad (5.60)$$

- $\mu_{(i)} \geq 0$ is the *step-size*¹⁰.

Updating the weights according to these schemes ensures that the cost function declines if $\mu_{(i)}$ is suitable small. This is easily seen by performing the Taylor expansion:

$$C_N(\mathbf{w}_{(i-1)} - \mu_{(i)} \nabla_N(\mathbf{w}_{(i-1)})) = C_N(\mathbf{w}_{(i-1)}) - \mu_{(i)} \nabla_N^\top(\mathbf{w}_{(i-1)}) \nabla_N(\mathbf{w}_{(i-1)}) + o\left(\mu_{(i)}^2\right). \quad (5.61)$$

Define an arbitrary positive definite matrix, \mathbf{R} . Then modifying the search direction $\Delta \mathbf{w}_{(i-1)} = \mathbf{w}_{(i)} - \mathbf{w}_{(i-1)}$ according to

$$\Delta \mathbf{w}_{(i-1)} = -\mu_{(i)} \mathbf{R} \nabla_N(\mathbf{w}_{(i-1)}) \quad (5.62)$$

¹⁰In the neural network community $\mu_{(i)}$ is often denoted the learning rate or the convergence parameter.

ensures that the cost function still declines (see also [Seber & Wild 89, Theorem 13.1]). Generally, we denote the weight update in pa:descent a descent direction.

The literature provides various methods for selecting $\mu_{(i)}$. The method of exact line search consists in choosing:

$$\mu_{(i)} = \arg \min_{\mu} C_N(\mathbf{w}_{(i-1)} - \mu \nabla_N(\mathbf{w}_{(i-1)})). \quad (5.63)$$

An algorithm for performing this minimization is e.g., mentioned in [Seber & Wild 89, Sec. 13.2.3], [Press et al. 88, Ch. 10]; however, this topic is not further elaborated. A simple strategy which is not an exact line search consists in initially choosing a relatively large step-size, and then gradually reducing the step-size until

$$C_N(\mathbf{w}_{(i-1)} - \mu_{(i)} \nabla_N(\mathbf{w}_{(i-1)})) \leq C_N(\mathbf{w}_{(i-1)}) \quad (5.64)$$

is reached.

Now a possible GD-algorithm runs as follows:

Gradient Descent Algorithm Initialization:

- Step 1*
- a.* Choose an initial weight vector $\mathbf{w}_{(0)}$ and determine the cost, $C_N(\mathbf{w}_{(0)})$.
 - b.* Let the “old” cost $C_N(\mathbf{w}_{(i-1)}) = \infty$ ¹¹.
 - c.* Select a threshold, $\tau \ll 1$, specifying the minimum relative change in the cost function.
 - d.* Select the maximum number of iterations, *itr*.
 - e.* Select the maximum step-size, μ_{\max} .
 - f.* Select the regularization parameter, κ .
 - g.* Initialize the counter $i = 0$.

Step 2 **while** $(i < itr) \wedge \left(\frac{C_N(\mathbf{w}_{(i-1)}) - C_N(\mathbf{w}_{(i)})}{C_N(\mathbf{w}_{(i-1)})} \geq \tau \right)$

Increment the counter i

Calculate the gradient, $\nabla_N(\mathbf{w}_{(i-1)})$

Set $C_N(\mathbf{w}_{(i)}) = \infty$ and $\mu = \mu_{\max}$

while $C_N(\mathbf{w}_{(i)}) > C_N(\mathbf{w}_{(i-1)})$

$\mathbf{w}_{(i)} = \mathbf{w}_{(i-1)} - \mu \nabla_N(\mathbf{w}_{(i-1)})$

Calculate $C_N(\mathbf{w}_{(i)})$

$\mu \leftarrow \mu/2$

end

end

Notes:

¹¹That is, equal to a very large number.

- The initial weights, \mathbf{w}_0 are normally chosen as a random vector; however, in Sec. 5.3.2 a special initialization procedure for a 2-layer MFPNN is presented.
- The weight updating is terminated when $i = itr$ or when the stop criterion (see also [Seber & Wild 89, Sec. 15.2.4]):

$$\frac{C_N(\mathbf{w}_{(i-1)}) - C_N(\mathbf{w}_{(i)})}{C_N(\mathbf{w}_{(i-1)})} < \tau \quad (5.65)$$

is met. τ is a suitable threshold (e.g., 10^{-6}) which specifies the minimum significant relative change in the cost function.

- The calculation of the gradient, $\nabla_N(\mathbf{w})$ is done according to `pa:nablaexp`, i.e.,

$$\nabla_N(\mathbf{w}) = -\frac{1}{N} \sum_{k=1}^N \psi(k; \mathbf{w}) e(k; \mathbf{w}) + \kappa \mathbf{w}. \quad (5.66)$$

The crucial part is the calculation of the instantaneous gradient $\psi(k; \mathbf{w})$. Clearly, this is highly dependent on the chosen filter architecture. In Sec. 5.3.3 this is discussed for layered filter architectures, and – in particular – Sec. 5.3.4 deals with the MFPNN architecture.

Note that the designation “weight decay regularization” according to `pa:nab` and the weight update `pa:gdupdat` becomes clear. Suppose that the first term of `pa:nab` is omitted, i.e., $\nabla_N(\mathbf{w}) = \kappa \mathbf{w}$. Now, cf. `pa:gdupdat`

$$\mathbf{w}_{(i)} = (1 - \kappa \mu_{(i)}) \mathbf{w}_{(i-1)}. \quad (5.67)$$

That is, if $|1 - \kappa \mu_{(i)}| < 1$ then $\mathbf{w}_{(i)}$ decays to the zero vector.

5.3.1 Convergence

The convergence properties are generally difficult to clarify. However, approximate results are deducible by performing a second order Taylor series expansion of the cost function around the estimated weight vector, $\hat{\mathbf{w}}$, i.e.¹²,

$$C_N(\mathbf{w}) = C_N(\hat{\mathbf{w}}) + \delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \delta \mathbf{w} \quad (5.68)$$

where $\delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}$ and

$$\mathbf{J}_N(\hat{\mathbf{w}}) = \frac{1}{2} \cdot \frac{\partial^2 C_N(\hat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \quad (5.69)$$

is the Hessian matrix evaluated at the estimated weights.

If exact line search is employed then the GD-algorithm is *linearly convergent* [Battiti 92] which is expressed by:

$$\left| C_N(\mathbf{w}_{(i)}) - C_N(\hat{\mathbf{w}}) \right| \approx \left(\frac{\chi - 1}{\chi + 1} \right)^2 \left| C_N(\mathbf{w}_{(i-1)}) - C_N(\hat{\mathbf{w}}) \right| \quad (5.70)$$

where

¹²Note that the linear term of the expansion, $-2\nabla_N(\hat{\mathbf{w}})\delta\mathbf{w}$, vanishes since $\hat{\mathbf{w}}$ minimizes the cost function.

- $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w})$ is a particular estimated weight.
- χ is the eigenvalue spread

$$\chi = \frac{\lambda_{\max}}{\lambda_{\min}} \quad (5.71)$$

where λ_{\min} , λ_{\max} , are the minimum and maximum eigenvalues of $\mathbf{J}_N(\hat{\mathbf{w}})$, respectively.

If the eigenvalue spread χ (which equals the condition number of the Hessian matrix) is large then the factor in the parentheses equals one approximately. Consequently, the improvement done in the i 'th iteration becomes small. The condition number of the Hessian is in many cases large (see p. 101 and furthermore App. B) which thus results in slow convergence.

Note that pa:congnd is valid only when the second order cost function expansion holds, i.e., when $\mathbf{w}_{(i)}$ is close to $\hat{\mathbf{w}}$. This is accomplished in the limit $i \rightarrow \infty$.

5.3.2 Weight Initialization

The initial weights, \mathbf{w}_0 , are normally chosen as a small random vector or equal to zero. In the context of MFPNN these strategies often result in a very “flat” cost function, i.e., the gradient is small. Consequently, many iterations should be performed before any decrease in the cost function is achieved, q.e., slow convergence. The “flat” cost function is due to the fact that the neurons in a particular layer respond very similarly. In particular, this is the case if the weights are very small or large according to the discussion p. 101 (see also Sec. 3.2.2). Furthermore, the larger the number of neurons – in a specific layer – the larger probability that two neurons respond very similarly.

Consequently, it may be profitable to design a simple weight initialization procedure which ensures that the responses of the neurons in a particular layer become as different as possible.

For convenience we consider the initialization of a 2-layer MFPNN characterized by $\mathbf{m} = [m_0, m_1, m_2]^\top = [p, q, 1]^\top$. The filtering is cf. Sec. 3.2.2 given by:

$$\begin{aligned} \hat{y}(k) &= \mathbf{W}^{(2)} \mathbf{h} \left(\mathbf{W}^{(1)} \mathbf{s}^{(0)} \right) \\ &= w_{10}^{(2)} + \sum_{i=1}^q w_{1i}^{(2)} h \left(\left(\tilde{\mathbf{w}}_i^{(1)} \right)^\top \mathbf{z}(k) + w_{i0}^{(1)} \right) \end{aligned} \quad (5.72)$$

where

- $\mathbf{W}^{(r)}$, $r = 1, 2$, are the weight matrices:

$$\begin{aligned} \mathbf{W}^{(1)} &= \left[\mathbf{w}_1^{(1)}, \mathbf{w}_2^{(1)}, \dots, \mathbf{w}_q^{(1)} \right]^\top \\ \mathbf{W}^{(2)} &= \left(\mathbf{w}_1^{(2)} \right)^\top \end{aligned} \quad (5.73)$$

where

$$\mathbf{w}_i^{(r)} = \left[w_{i,0}^{(r)}, \left(\tilde{\mathbf{w}}_i^{(r)} \right)^\top \right]^\top = \left[w_{i,0}^{(r)}, w_{i,1}^{(r)}, \dots, w_{i,m_{r-1}}^{(r)} \right]^\top. \quad (5.74)$$

- $h(u)$ is the activation function which is chosen as the hyperbolic tangent, $\tanh(u)$.
- $\mathbf{z}(k) = [z_1(k), z_2(k), \dots, z_p(k)]^\top$ is the input vector signal and $\mathbf{s}^{(0)} = [1, \mathbf{z}^\top(k)]^\top$.

According to the study of the nonlinear perceptron Sec. 3.2.2 the individual neuron, say the i 'th neuron, divides the input space (i.e., the \mathbf{z} -space) into a positive and a negative region which are separated by the hyperplane:

$$\mathcal{H}_i : \left(\tilde{\mathbf{w}}_i^{(1)} \right)^\top \mathbf{z}(k) + w_{i,0}^{(1)} = 0. \quad (5.75)$$

The use of $\tanh(\cdot)$ -perceptrons results in a smooth transition between the positive and negative region.

These observations enable the formulation of a basic strategy for weight initialization which consists in the following items:

1. Selection of the directions of the normal vectors, $\tilde{\mathbf{w}}_i^{(1)}$, of the hyperplanes, \mathcal{H}_i .
2. Scaling of the weights so that the transition region of each neuron is related to the dynamic range of the input vector.
3. Location of the hyperplanes, i.e., specification of the thresholds $w_{i,0}^{(1)}$.
4. Selection of the weights in the second layer, i.e., $\mathbf{w}_1^{(2)}$.

The normal vectors are chosen parallel to the axes, z_j , i.e.,

$$\tilde{\mathbf{w}}_i^{(1)} = [0, \dots, 0, w_{i,j}^{(1)}, 0, \dots, 0]^\top, \quad j = 1, 2, \dots, p. \quad (5.76)$$

Since the number of hidden neurons, q , normally is greater than the number of input variables, $p = \dim(\mathbf{z})$ more normal vectors will be identical. Define n as the number of neurons in the hidden layer per input variable, i.e., $n = \lfloor q/p \rfloor$, and furthermore the number of surplus neurons, $s = q - np$. Now, pick n neurons with parallel normal vectors for each of $p - s$ input dimensions, and $n + 1$ neurons for the remaining s dimensions which are chosen randomly among the p possible. Let us illustrate this selection by the following example:

EXAMPLE 5.1

Let $p = 3$ and $q = 11$. That is, $n = \lfloor 11/3 \rfloor = 3$ and $r = 11 - 9 = 2$.

Neuron Number	Input Direction
1 – 3	z_1
4 – 6	z_2
7 – 9	z_3
10	z_3
11	z_1

The input directions of the neuron # 10, 11 are chosen randomly among the possible z_j -directions with the restriction that they are required to be different. \square

The next item concerns scaling of the weights. In that context define the *transition width*, b_o , of the activation function, $h(u)$, according to Fig. 5.2 The transition width is related to the specified dynamic range $[-h_{\max}; h_{\max}]$ and the shape of $h(u)$. With $h(u) = \tanh(u)$ we get:

$$b_o = \ln \left[\frac{1 + h_{\max}}{1 - h_{\max}} \right]. \quad (5.77)$$

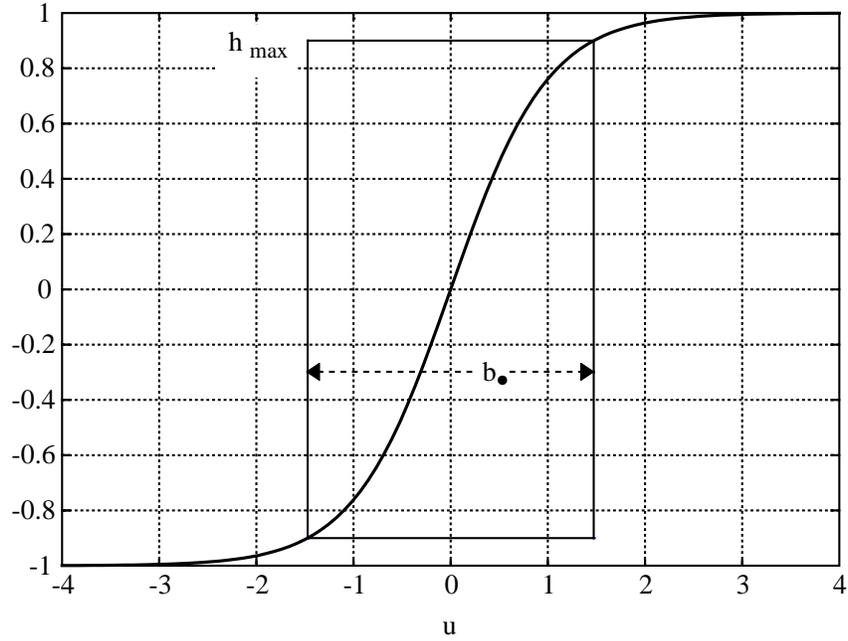


Figure 5.2: The transition width, b_o , of the activation function related to the dynamic range $[-h_{\max}; h_{\max}]$.

Now suppose that the normal vector of the i 'th neuron is parallel to the z_j direction, i.e., the response of the neurons given by:

$$s_i^{(1)} = h\left(w_{i,j}^{(1)} z_j(k) + w_{i,0}\right). \quad (5.78)$$

The desired transition width, b_j , of the neuron should then be small compared to the dynamic range of $z_j(k)$ for two reasons: First, in order to ensure reasonable dynamics of the neuron responses. Secondly, to ensure responses with low resemblance¹³. This is accomplished by setting:

$$b_j = \delta \cdot dr_{z_j} \quad (5.79)$$

where

- δ is a suitable percentage. For instance, $\delta = 10\%$; however, the larger n the smaller percentage, preferably.
- dr_{z_j} is the dynamic range of $z_j(k)$ given by:

$$dr_{z_j} = \max_k \{z_j(k)\} - \min_k \{z_j(k)\}. \quad (5.80)$$

If the $z_j(k)$ is spiky – i.e., the p.d.f. has long tails – we may rather prefer:

$$dr_{z_j} = \min \left\{ \max_k \{z_j(k)\} - \min_k \{z_j(k)\}, c \cdot \sigma_{z_j} \right\} \quad (5.81)$$

¹³Due to the fact that at least n neurons are placed along the z_j -direction, b_j is required to be small in order to produce responses with low resemblance.

where σ_{z_j} is the estimated standard deviation of $z_j(k)$, and c is a suitable constant, e.g., $c = 3$.

The desired transition width is cf. pa:neures obtained by letting:

$$w_{i,j}^{(1)} = \frac{b_o}{b_j}. \quad (5.82)$$

The third item concerns the location of, say n , individual neurons, i_1, i_2, \dots, i_n , which have identical normal vectors. The location on the z_j axis is cf. pa:hyp, (5.78) given by:

$$z_j = -\frac{w_{i,0}^{(1)}}{w_{i,j}^{(1)}}. \quad (5.83)$$

We suggest to locate the hyperplanes uniformly according to the fractiles of the estimated probability distribution function of $z_j(k)$ defined by:

$$P_{z_j}(z_j) = \text{Prob} \{z_j(k) < z_j\}. \quad (5.84)$$

The ϵ -fractile, denoted $z_{j,\epsilon}$, is accordingly given by:

$$P_{z_j}(z_{j,\epsilon}) = \epsilon. \quad (5.85)$$

The uniformly distributed fractiles, ϵ_ℓ , $\ell = 1, 2, \dots, n$ corresponding to the neurons, i_ℓ , are then:

$$\epsilon_\ell = \frac{\ell}{n+1}, \quad \ell = 1, 2, \dots, n. \quad (5.86)$$

For instance, if $n = 1$ then the neuron is located at the 50%-fractile – or equivalently – the mean value. According to pa:zjloc this defines the setting of the bias weights, i.e.,

$$w_{i_\ell,0}^{(1)} = -z_{j,\epsilon} \cdot w_{i_\ell,j}^{(1)}. \quad (5.87)$$

In Fig. 5.3 an example of this recipe is shown.

Finally, the fourth item is about the selection of the weights in the second layer. The bias weight is selected as the time-average of the output¹⁴, i.e.,

$$w_{10}^{(2)} = \langle y(k) \rangle = \frac{1}{N} \sum_{k=1}^N y(k). \quad (5.88)$$

Define the dynamic range of the output (see also pa:drzj2):

$$dr_y = \max_k \{y(k)\} - \min_k \{y(k)\}. \quad (5.89)$$

Now, the remaining weights, $w_{1i}^{(2)}$, $i \in [1; q]$, are chosen as independent uniformly distributed random variables over the interval:

$$\left[-\frac{dr_y}{2q}, \frac{dr_y}{2q} \right].$$

¹⁴This corresponds to the LS estimate, cf. Sec. 5.2, provided that the bias term is the only term entering the model.

The factor q^{-1} ensures that the dynamic range of the filter output, $\hat{y}(k)$ is of the same order of magnitude as dr_y .

To sum up, the weight initialization algorithm runs as follows:

Weight Initialization Algorithm Initialization:

- Step 1*
- a. Choose the dynamic range of the activation function by setting h_{\max} . Typically, $h_{\max} \in [0.8; 0.9]$.
 - b. Select δ which determines the desired transition width of the neurons as a percentage of the dynamic range of the input variables. Frequently, $\delta = 10\%$.
 - c. Specify the number of input variables, p , and the number of hidden neurons, q . It is assumed that $q \geq p$.

Step 2 Selection of normal vector directions:

- a. Calculate the number of neurons in the hidden layer per input variable, $n = \lfloor q/p \rfloor$, and the number of surplus neurons, $s = q - np$.
- b. If $s > 0$ then select s different input directions, $\mathcal{J} = \{j_1, j_2, \dots, j_s\}$ randomly among $[1; p]$.
- c. Define n_j as the number of neurons with normal vector direction parallel to the z_j -axis, i.e.,

$$n_j = \begin{cases} n + 1 & j \in \mathcal{J} \\ n & \text{otherwise} \end{cases} .$$

Step 3 Calculate the transition width of the activation function, i.e., determine

$$b_o = \ln \left[\frac{1 + h_{\max}}{1 - h_{\max}} \right] .$$

Step 4 Initialize $i = 1$

for $j = 1, 2, \dots, p$

Calculate dr_{z_j} , cf. pa:drzj1 or pa:drzj2.

$$b_j = \delta \cdot dr_{z_j}$$

Estimate $P_{z_j}(z_j)$

for $\ell = 1, 2, \dots, n_j$

$$\epsilon = \frac{\ell}{n_j + 1}$$

Determine the fractile $z_{j,\epsilon}$ according to $P_{z_j}(z_{j,\epsilon}) = \epsilon$

$$w_{i,j}^{(1)} = \frac{b_o}{b_j}$$

$$w_{i,0}^{(1)} = -z_{j,\epsilon} \cdot w_{i,j}^{(1)}$$

Increment i

end

end

Step 5 Weights in the output layer:

- a. $w_{10}^{(2)} = \frac{1}{N} \sum_{k=1}^N y(k)$
- b. Determine the dynamic range, dr_y , and draw $w_{1i}^{(2)}$, $i \in [1; q]$, independently from a uniform distribution over the interval: $[-dr_y(2q)^{-1}; dr_y(2q)^{-1}]$.

In Ch. 8 the suggested weight initialization algorithm is tested numerically, and the results demonstrate that convergence is speeded up.

5.3.3 The Back-Propagation Algorithm

The Back-Propagation Algorithm (BP) [McClelland & Rumelhart 86] was invented as an algorithm for estimating the weights of a multi-layer feed-forward perceptron neural network (MFPNN). The essential part of the BP-algorithm is the computation of the instantaneous gradient vector $\boldsymbol{\psi}(k; \mathbf{w})$ at some prescribed weights, \mathbf{w} . In this work we demonstrate that the BP Algorithm can be extended to deal with all layered filter architectures. Consider the layered filter structure depicted in Fig. 5.4.

The filter output obeys:

$$\begin{aligned} \hat{y}(k) &= f(\mathbf{x}(k); \mathbf{w}) \\ &= f^{(l)} \left(\mathbf{f}^{(l-1)} \left(\dots \mathbf{f}^{(1)} \left(\mathbf{x}(k); \mathbf{w}^{(1)} \right) \dots; \mathbf{w}^{(l-1)} \right); \mathbf{w}^{(l)} \right) \end{aligned} \quad (5.90)$$

where

- $\mathbf{f}^{(r)}(\cdot)$ is the vector function with a prescribed dimension m_r ¹⁵ which specifies the nonlinear mapping within the r 'th layer, $r \in [1; l]$.
- $\mathbf{w}^{(r)}$ is the weight vector associated with the r 'th layer, i.e., we assume the following partition:

$$\mathbf{w} = \left[\left(\mathbf{w}^{(1)} \right)^\top, \left(\mathbf{w}^{(2)} \right)^\top, \dots, \left(\mathbf{w}^{(l)} \right)^\top \right]^\top. \quad (5.91)$$

- $\tilde{\mathbf{s}}^{(r)}$ is the vector output of the r 'th layer with dimension m_r ¹⁶, i.e.,

$$\tilde{\mathbf{s}}^{(r)}(k) = \mathbf{f}^{(r)} \left(\mathbf{f}^{(r-1)} \left(\dots \mathbf{f}^{(1)}(\mathbf{x}(k); \mathbf{w}^{(1)}) \dots \right); \mathbf{w}^{(r)} \right). \quad (5.92)$$

Formally, $\tilde{\mathbf{s}}^{(l)}(k) = \hat{y}(k)$ and $\tilde{\mathbf{s}}^{(0)}(k) = \mathbf{x}(k)$.

With this notation the instantaneous gradient vector is expressed as (below time indices are omitted):

$$\begin{aligned} \boldsymbol{\psi}(\mathbf{w}) &= \frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \mathbf{w}} \\ &= \left[\frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \left(\mathbf{w}^{(1)} \right)^\top}, \frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \left(\mathbf{w}^{(2)} \right)^\top}, \dots, \frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \left(\mathbf{w}^{(l)} \right)^\top} \right]^\top. \end{aligned} \quad (5.93)$$

¹⁵In the output layer the $f^{(l)}$ is a scalar function since the filter output is a scalar.

¹⁶The \sim is used to adopt the notation within MFPNN, cf. Ch. 3.

That is the essential partial derivatives to calculate are:

$$\frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \mathbf{w}^{(r)}}, \quad r = 1, 2, \dots, l.$$

Applying the chain rule we get:

$$\begin{aligned} \frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \mathbf{w}^{(r)}} &= \frac{\partial \hat{y}}{\partial \mathbf{w}^{(r)}} \\ &= \begin{cases} \frac{\partial \hat{y}}{\partial \mathbf{w}^{(l)}} & , r = l \\ \frac{\partial (\tilde{\mathbf{s}}^{(r)})^\top}{\partial \mathbf{w}^{(r)}} \cdot \frac{\partial (\tilde{\mathbf{s}}^{(r+1)})^\top}{\partial \tilde{\mathbf{s}}^{(r)}} \cdots \frac{\partial (\tilde{\mathbf{s}}^{(l-1)})^\top}{\partial \tilde{\mathbf{s}}^{(l-2)}} \cdot \frac{\partial \hat{y}}{\partial \tilde{\mathbf{s}}^{(l-1)}} & , 1 \leq r < l \end{cases} \end{aligned} \quad (5.94)$$

Note that $\partial (\tilde{\mathbf{s}}^{(r+1)})^\top / \partial \tilde{\mathbf{s}}^{(r)}$ are $m_r \times m_{r+1}$ matrices.

pa:chain shows that the instantaneous gradient vectors can be calculated recursively. Define the vectors: $\boldsymbol{\delta}^{(r)}$, with dimension m_r , $r = 1, 2, \dots, l-1$, so that

$$\frac{\partial \hat{y}}{\partial \mathbf{w}^{(r)}} = \begin{cases} \frac{\partial \hat{y}}{\partial \mathbf{w}^{(l)}} & , r = l \\ \frac{\partial (\tilde{\mathbf{s}}^{(r)})^\top}{\partial \mathbf{w}^{(r)}} \boldsymbol{\delta}^{(r)} & , 1 \leq r < l \end{cases}. \quad (5.95)$$

That is, by comparison with pa:chain:

$$\boldsymbol{\delta}^{(r-1)} = \frac{\partial (\tilde{\mathbf{s}}^{(r)})^\top}{\partial \tilde{\mathbf{s}}^{(r-1)}} \boldsymbol{\delta}^{(r)}, \quad r = l-1, \dots, 2 \quad (5.96)$$

with

$$\boldsymbol{\delta}^{(l-1)} = \frac{\partial \hat{y}}{\partial \tilde{\mathbf{s}}^{(l-1)}}. \quad (5.97)$$

This is shown in Fig. 5.5. If only the product of the error signal and the instantaneous gradient vector is required (e.g., in the steepest descent algorithm) then $\boldsymbol{\delta}^{(l-1)}$ is redefined as:

$$\boldsymbol{\delta}^{(l-1)} = \frac{\partial \hat{y}}{\partial \tilde{\mathbf{s}}^{(l-1)}} \cdot e. \quad (5.98)$$

5.3.4 Back-Propagation in the MFPNN

In this subsection we present the detailed BP-algorithm when dealing with the MFPNN with linear output neuron [McClelland & Rumelhart 86]. According to nf:mlnn the processing within the network is given by:

$$\begin{aligned} \hat{y}(k) &= f_n(\mathbf{z}(k); \mathbf{w}) \\ &= \mathbf{W}^{(l)} \left(\mathbf{h} \left(\mathbf{W}^{(l-1)} \cdots \mathbf{h} \left(\mathbf{W}^{(1)} \mathbf{s}^{(0)}(k) \right) \cdots \right) \right) \end{aligned} \quad (5.99)$$

where

- $\mathbf{W}^{(r)}$ is the $m_r \times (m_r + 1)$ dimensional weight matrix containing the weights in the r 'th layer,

$$\mathbf{W}^{(r)} = \left[\mathbf{w}_1^{(r)}, \mathbf{w}_2^{(r)}, \dots, \mathbf{w}_{m_r}^{(r)} \right]^\top = \begin{bmatrix} w_{10}^{(r)} & w_{11}^{(r)} & \cdots & w_{1,m_r-1}^{(r)} \\ w_{20}^{(r)} & w_{21}^{(r)} & \cdots & w_{2,m_r-1}^{(r)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m_r,0}^{(r)} & w_{m_r,1}^{(r)} & \cdots & w_{m_r,m_r-1}^{(r)} \end{bmatrix}. \quad (5.100)$$

- m_r is the number of neurons in the r 'th layer, q.e., $\dim(\tilde{\mathbf{s}}^{(r)}) = m_r$.
- $\mathbf{h}(\mathbf{u}^{(r)}) = [h(u_0^{(r)}), h(u_1^{(r)}), \dots, h(u_{m_r}^{(r)})]^\top$ is the vector activation function¹⁷ and $h(\cdot)$ the activation function which is assumed to be differentiable with derivative $h'(\cdot)$. If $h(u) = \tanh(u)$ then the derivative becomes especially simple as $h'(u) = 1 - h^2(u)$.
- $\mathbf{s}^{(0)}(k)$ is the augmented vector signal $\mathbf{s}^{(0)}(k) = [1, \mathbf{z}^\top(k)]^\top$ where $\mathbf{z}(k)$ is the pre-processed input vector signal.

The vector output of the r 'th layer, $r < l$, thus obeys (omitting time index):

$$\begin{aligned} \tilde{\mathbf{s}}^{(r)} &= \mathbf{h} \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)} \right) \\ &= \mathbf{h} \left(\mathbf{W}^{(r)} \left[1, \left(\tilde{\mathbf{s}}^{(r-1)} \right)^\top \right]^\top \right), \end{aligned} \quad (5.101)$$

and the l 'th layer

$$\hat{\mathbf{y}} = \mathbf{W}^{(l)} \left[1, \left(\tilde{\mathbf{s}}^{(l-1)} \right)^\top \right]^\top. \quad (5.102)$$

With reference to pa:delrec

$$\begin{aligned} \frac{\partial \left(\tilde{\mathbf{s}}^{(r)} \right)^\top}{\partial \tilde{\mathbf{s}}^{(r-1)}} &= \left[h' \left(\left(\mathbf{w}_1^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \tilde{\mathbf{w}}_1^{(r)}, h' \left(\left(\mathbf{w}_2^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \tilde{\mathbf{w}}_2^{(r)}, \dots, \right. \\ &\quad \left. h' \left(\left(\mathbf{w}_{m_r}^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \tilde{\mathbf{w}}_{m_r}^{(r)} \right] \end{aligned} \quad (5.103)$$

where $\tilde{\mathbf{w}}_i^{(r)} = [w_{i1}^{(r)}, w_{i2}^{(r)}, \dots, w_{im_r-1}^{(r)}]^\top$ is the restricted weight vector, i.e., the bias weight, $w_{i0}^{(r)}$, is not included. Now, the δ -recursion pa:delrec for $r = l - 1, \dots, 2$ yields:

$$\begin{aligned} \delta^{(r-1)} &= \left[h' \left(\left(\mathbf{w}_1^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \tilde{\mathbf{w}}_1^{(r)}, h' \left(\left(\mathbf{w}_2^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \tilde{\mathbf{w}}_2^{(r)}, \dots, \right. \\ &\quad \left. h' \left(\left(\mathbf{w}_{m_r}^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \tilde{\mathbf{w}}_{m_r}^{(r)} \right] \delta^{(r)} \\ &= \left(\tilde{\mathbf{w}}^{(r)} \right)^\top \mathbf{h}' \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)} \right) \odot \delta^{(r)} \end{aligned} \quad (5.104)$$

where

¹⁷If the argument is column (row) vector $\mathbf{h}(\cdot)$ is a column (row) vector.

- $\tilde{\mathbf{w}}^{(r)}$ is the $m_r \times m_{r-1}$ restricted waight matrix,

$$\tilde{\mathbf{w}}^{(r)} = \begin{bmatrix} w_{11}^{(r)} & w_{12}^{(r)} & \cdots & w_{1,m_{r-1}}^{(r)} \\ w_{21}^{(r)} & w_{22}^{(r)} & \cdots & w_{2,m_{r-1}}^{(r)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m_r,1}^{(r)} & w_{m_r,2}^{(r)} & \cdots & w_{m_r,m_{r-1}}^{(r)} \end{bmatrix} \quad (5.105)$$

i.e., all bias weights in the first column of $\mathbf{W}^{(r)}$ are excluded.

- \odot is the element by element matrix (vector) product.
- $\mathbf{h}'(\mathbf{u}) = [h'(u_0^{(r)}), h'(u_1^{(r)}), \dots, h'(u_{m_r}^{(r)})]^\top$ is the derivative of the vector activation function.

The recursion is initiated by $\delta^{(l-1)}$ which according to pa:delfin is given by:

$$\begin{aligned} \delta^{(l-1)} &= \frac{\partial \hat{y}}{\partial \tilde{\mathbf{s}}^{(l-1)}} \\ &= \frac{\partial (\mathbf{W}^{(l)} \mathbf{s}^{(l-1)})}{\partial \tilde{\mathbf{s}}^{(l-1)}} \\ &= (\tilde{\mathbf{w}}^{(l)})^\top. \end{aligned} \quad (5.106)$$

What remains cf. pa:chain is to calculate the partial derivative of $\tilde{\mathbf{s}}^{(r)}$ w.r.t. the weights in that particular layer contained in a vector, $\mathbf{w}^{(r)}$, which is defined as:

$$\mathbf{w}^{(r)} = \left[(\mathbf{w}_1^{(r)})^\top, (\mathbf{w}_2^{(r)})^\top, \dots, (\mathbf{w}_{m_r}^{(r)})^\top \right]^\top. \quad (5.107)$$

Consequently, the partial derivative, $r = 1, 2, \dots, l-1$, yields:

$$\begin{aligned} \frac{\partial (\tilde{\mathbf{s}}^{(r)})^\top}{\partial \mathbf{w}^{(r)}} &= \frac{\partial \left[\mathbf{h} \left((\mathbf{s}^{(r-1)})^\top (\mathbf{W}^{(r)})^\top \right) \right]}{\partial \left[(\mathbf{w}_1^{(r)})^\top, (\mathbf{w}_2^{(r)})^\top, \dots, (\mathbf{w}_{m_r}^{(r)})^\top \right]^\top} \\ &= \frac{\partial \left[h \left((\mathbf{s}^{(r-1)})^\top \mathbf{w}_1^{(r)} \right), h \left((\mathbf{s}^{(r-1)})^\top \mathbf{w}_2^{(r)} \right), \dots, h \left((\mathbf{s}^{(r-1)})^\top \mathbf{w}_{m_r}^{(r)} \right) \right]}{\partial \left[(\mathbf{w}_1^{(r)})^\top, (\mathbf{w}_2^{(r)})^\top, \dots, (\mathbf{w}_{m_r}^{(r)})^\top \right]^\top} \\ &= \begin{bmatrix} h' \left((\mathbf{w}_1^{(r)})^\top \mathbf{s}^{(r-1)} \right) \mathbf{s}^{(r-1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & h' \left((\mathbf{w}_2^{(r)})^\top \mathbf{s}^{(r-1)} \right) \mathbf{s}^{(r-1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & h' \left((\mathbf{w}_{m_r}^{(r)})^\top \mathbf{s}^{(r-1)} \right) \mathbf{s}^{(r-1)} \end{bmatrix}. \end{aligned} \quad (5.108)$$

Now, according to pa:gradlay, for $1 \leq r < l$,

$$\begin{aligned} \frac{\partial \hat{y}}{\partial \mathbf{w}^{(r)}} &= \frac{\partial \left(\tilde{\mathbf{s}}^{(r)} \right)^\top}{\partial \mathbf{w}^{(r)}} \boldsymbol{\delta}^{(r)} \\ &= \begin{bmatrix} h' \left(\left(\mathbf{w}_1^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \mathbf{s}^{(r-1)} \delta_1^{(r)} \\ h' \left(\left(\mathbf{w}_2^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \mathbf{s}^{(r-1)} \delta_2^{(r)} \\ \vdots \\ h' \left(\left(\mathbf{w}_{m_r}^{(r)} \right)^\top \mathbf{s}^{(r-1)} \right) \mathbf{s}^{(r-1)} \delta_{m_r}^{(r)} \end{bmatrix}. \end{aligned} \quad (5.109)$$

Consequently, using the weight matrix representation

$$\begin{aligned} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(r)}} &= \begin{bmatrix} \frac{\partial \hat{y}}{\partial \mathbf{w}_1^{(r)}} \\ \frac{\partial \hat{y}}{\partial \mathbf{w}_2^{(r)}} \\ \vdots \\ \frac{\partial \hat{y}}{\partial \mathbf{w}_{m_r}^{(r)}} \end{bmatrix} \\ &= \left(\mathbf{s}^{(r-1)} \right)^\top \left(\mathbf{h}' \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)} \right) \odot \boldsymbol{\delta}^{(r)} \right). \end{aligned} \quad (5.110)$$

Within the last layer $r = l$ we get¹⁸:

$$\begin{aligned} \frac{\partial \hat{y}}{\partial \mathbf{W}^{(l)}} &= \frac{\partial \mathbf{W}^{(l)} \mathbf{s}^{(l-1)}}{\partial \mathbf{W}^{(l)}} \\ &= \left(\mathbf{s}^{(l-1)} \right)^\top. \end{aligned} \quad (5.111)$$

To sum up the BP-algorithm for the MFPNN is given by:

BP-Algorithm for the MFPNN The instantaneous gradient of the output layer, $r = l$:

$$\frac{\partial \hat{y}}{\partial \mathbf{W}^{(l)}} = \left(\mathbf{s}^{(l-1)} \right)^\top$$

and initialize,

$$\boldsymbol{\delta}^{(l-1)} = \left(\tilde{\mathbf{W}}^{(l)} \right)^\top.$$

Step 2 for $r = l - 1, l - 1, \dots, 2$

$$\boldsymbol{\nu} = \mathbf{h}' \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)} \right) \odot \boldsymbol{\delta}^{(r)}$$

$$\boldsymbol{\delta}^{(r-1)} = \left(\tilde{\mathbf{W}}^{(r)} \right)^\top \boldsymbol{\nu}$$

¹⁸Recall that $\mathbf{W}^{(l)}$ in fact is a row vector since we deal with a single output architecture.

$$\frac{\partial \hat{y}}{\partial \mathbf{W}^{(r)}} = (\mathbf{s}^{(r-1)})^\top \boldsymbol{\nu}$$

end

Step 3

$$\frac{\partial \hat{y}}{\partial \mathbf{W}^{(1)}} = (\mathbf{s}^{(0)})^\top (\mathbf{h}'(\mathbf{W}^{(1)} \mathbf{s}^{(0)}) \odot \boldsymbol{\delta}^{(1)}).$$

$\boldsymbol{\nu}$ is an auxiliary vector introduced in order to minimize the computational complexity. If only the product $\boldsymbol{\psi}e$ is required then use the initialization:

$$\boldsymbol{\delta}^{(l-1)} = (\widetilde{\mathbf{W}}^{(l)})^\top e \quad (5.112)$$

and

$$\frac{\partial \hat{y}}{\partial \mathbf{W}^{(l)}} e = (\mathbf{s}^{(l-1)})^\top e. \quad (5.113)$$

In Fig. 5.6 below the BP-algorithm dealing with the MFPNN architecture is depicted.

5.3.4.1 Complexity of Back-Propagation

In order to perform a fair comparison of various weight estimation algorithms both performance and computational complexity CP have to be considered. In that connection we evaluate the complexity of the BP-algorithm when dealing with the MFPNN. Consider:

- $CP_{\hat{y}}$, the complexity involved in calculating one sample of the filter output.
- $CP_{\boldsymbol{\psi}}$, the complexity involved in calculating one sample of the instantaneous gradient vector provided that the state vectors, $\mathbf{s}^{(r)}$, $r = 0, 1, \dots, l-1$, are determined in advance. This is actually a spinoff when calculating the filter output, \hat{y} .

The computational complexity is here defined as the total number of multiplications and divisions, i.e., additions, move operations etc. are neglected. The complexity of a division depends on the hardware platform. For instance using the Intel 486 microprocessor a floating point division requires around 7 times more clock cycles than a multiplication and dealing with a digital signal processor the factor is 16. As an approximate value we then consider the division to be 10 times more complex than the multiplication.

Filter Output According to pa:procray and (5.102) the required computations can be split into the following:

- For each layer, $1 \leq r \leq l$ the matrix product $\mathbf{W}^{(r)} \mathbf{s}^{(r-1)}$ which involves $m_r m_{r-1}$ multiplications since multiplications with the first element of $\mathbf{s}^{(r-1)}$, equal to unity, are not included.
- Within all layer but the output layer m_r evaluations of $h(\cdot)$ is required. The approximation capabilities of the neural network is not critically dependent on the exact shape of the activation function, $h(\cdot)$. Consequently, – regarding complexity – we

assume that $h(\cdot)$ is specified by a table of connected points: $\{u_n, h(u_n)\}$ and $h(u)$ is evaluated by linear interpolation, i.e.,

$$h(u) = \frac{h(u_n) - h(u_{n-1})}{u_n - u_{n-1}} (u - u_{n-1}) + h(u_{n-1}). \quad (5.114)$$

One evaluation of $h(u)$ thus requires one division and one multiplication – or equivalently – 11 multiplications. In total the complexity is: $11m_r$.

To sum up:

$$CP_{\hat{y}} = m_{l-1} + \sum_{r=1}^{l-1} m_r (m_{r-1} + 11). \quad (5.115)$$

In particular dealing with a 2-layer network with $m_0 = p$ and $m_1 = q$,

$$CP_{\hat{y}} = q(p + 12). \quad (5.116)$$

Now, the total number of weights, $m = q(p + 2) + 1$, q.e.,

$$CP_{\hat{y}} = (m - 1) \frac{p + 12}{p + 2}. \quad (5.117)$$

Now, usually $m \gg p$ and consequently we use the approximation:

$$CP_{\hat{y}} \approx m \frac{p + 12}{p + 2}. \quad (5.118)$$

Gradient Suppose that the output of the filter, \hat{y} , is calculated and all state vectors, $\mathbf{s}^{(r)}$, $r = 0, 1, \dots, l - 1$ are known. Now, consider the individual lines in the BP-algorithm p. 125:

Step 1. Does not involve any multiplications.

- Step 2.*
1. Focus on the calculation of $\boldsymbol{\nu} = \mathbf{h}' \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)} \right) \odot \boldsymbol{\delta}^{(r)}$. First consider the term: $\mathbf{h}' \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)} \right)$. If the hyperbolic tangent is used as the activation function the relation $h'(u) = 1 - h^2(u)$ could be used; however in general the activation function is implemented as table look up followed by an interpolation. The product $\mathbf{W}^{(r)} \mathbf{s}^{(r-1)}$ is already determined so what is left is $m_r h'(u)$ evaluations. If $h'(u)$ – like $h(u)$ – is implemented as a look up table, then the number of equivalent multiplications are: $11m_r$. Secondly, the element by element product requires m_r multiplications. In total: $12m_r$ multiplications.
 2. The product $\hat{\mathbf{w}}^{(r)} \boldsymbol{\nu}$ requires $m_r m_{r-1}$ multiplications.
 3. The product $\left(\mathbf{s}^{(r-1)} \right)^\top \boldsymbol{\nu}$ involves $m_r m_{r-1}$ multiplications since multiplications with one are neglected.

To sum up, the number of multiplications involved in *Step 2.* is:

$$\sum_{r=2}^{l-1} m_r (2m_{r-1} + 12).$$

Step 3. This step is equivalent to item 1 and 3 under *Step 2.*, i.e., $m_1 (m_0 + 12)$ multiplications.

The total complexity involved in the instantaneous gradient vector calculations is thus:

$$CP_{\psi} = m_1(m_0 + 12) + \sum_{r=2}^{l-1} m_r(2m_{r-1} + 12). \quad (5.119)$$

In particular dealing with a 2-layer network with $m_0 = p$ and $m_1 = q$,

$$CP_{\psi} = q(p + 12). \quad (5.120)$$

which is identical to $CP_{\hat{y}}$.

If only the product ψe is required then m_{l-1} extra multiplications are necessary when initializing $\delta^{(l-1)}$, cf. pa:delinie. In addition, pa:gradoute also involves m_{l-1} multiplications. Hence,

$$CP_{\psi e} = CP_{\psi} + 2m_{l-1}. \quad (5.121)$$

Dealing with a 2-layer network we get (see previous paragraph):

$$\begin{aligned} CP_{\psi e} &= (m-1)\frac{p+12}{p+2} + 2(m-1)\frac{1}{p+2} \\ &\approx m\frac{p+14}{p+2}. \end{aligned} \quad (5.122)$$

5.4 Stochastic Gradient Algorithm

The stochastic gradient (SG) algorithm is a recursive variant of the gradient descent algorithm which consists in replacing the gradient of the cost function, $\nabla_N(\cdot)$, in pa:gdupdat by the instantaneous value at time step k which cf. pa:nablaexp yields:

$$-\psi(k; \mathbf{w})e(k; \mathbf{w}) + \kappa\mathbf{w}.$$

The weight update consequently becomes

$$\begin{aligned} \mathbf{w}(k) &= \mathbf{w}(k-1) - \mu(k)(-\psi(k; \mathbf{w}(k-1))e(k; \mathbf{w}(k-1)) + \kappa\mathbf{w}(k-1)) \\ &= (1 - \kappa\mu(k))\mathbf{w}(k-1) + \mu(k)\psi(k; \mathbf{w}(k-1))e(k; \mathbf{w}(k-1)) \end{aligned} \quad (5.123)$$

where $\mathbf{w}(k)$, $\mu(k) \geq 0$ are the weight estimate and the step-size at time step k , respectively.

The term “stochastic gradient” (used e.g., by [Ljung & Söderström 83]) stems from the fact that the instantaneous value of the gradient may be viewed as a random sample of $\nabla_N(\cdot)$. This is also known as the method of “stochastic approximation” [Robbins & Monro 51]. In the neural network literature the SG-algorithm is known as the back-propagation algorithm [McClelland & Rumelhart 86] and in the linear signal processing literature one usually denotes the algorithm the Least Mean Squares (LMS) algorithm [Widrow & Hoff 60].

At first sight it may seem inefficient to approximate the gradient by its instant value; however, two advantages exist:

- The possibility of on-line tracking of time-varying systems. The step-size, $\mu(k)$, provides a trade off between the instant gradient and the old weight estimate. That is, if the environment changes rapidly then $\mu(k)$ should be large; otherwise, $\mu(k)$ should be reduced so that a more reliable estimate appears¹⁹. In the classical paper [Widrow et al. 76] the tracking properties of operating the SG-algorithm (actually the LMS-algorithm) in a non-stationary environment is discussed.

¹⁹The matters concerning setting of $\mu(k)$ is further elaborated below.

- Compared to the (off-line) GD-algorithm the SG-algorithm may escape from bad local minima. This is due to the fact that the weight update is a noisy variant of the gradient. That means, although the cost function declines in the direction of the negative gradient the “noise” may result in increasing cost; consequently, this makes it possible to break through a local maximum. It is possible to modify the weight update by adding additional noise in order to intensify the possibility of escaping from local minima, as mentioned in e.g., [Hertz et al. 91, Sec. 6.2], [White 89a].

5.4.1 Convergence and Step-Size Selection

An impending issue is the choice of the step-size in order to ensure convergence of the algorithm. Below we solely consider operating the SG-algorithm in the off-line mode, i.e., cf. Sec. 5.2 and `pa:sgupdat`, the recursion yields:

$$\mathbf{w}(\ell) = (1 - \kappa\mu(\ell)) \mathbf{w}(\ell - 1) + \mu(\ell)\boldsymbol{\psi}(k; \mathbf{w}(\ell - 1))e(k; \mathbf{w}(\ell - 1)) \quad (5.124)$$

where $\ell = k + (i - 1)N$, $k = 1, 2, \dots, N$ and $i = 1, 2, \dots, itr$ where itr is the number of iterations (number of training set replications). However, the provided statements are also applicable for operating in the on-line mode (with appropriate interpretation).

5.4.1.1 Fixed Step-Size

First, consider the case of a fixed step-size, i.e., $\mu(\ell) \equiv \mu$. General results are not accessible, instead – as in Sec. 5.3.1 – approximate results are available by considering a second order Taylor series approximation of the cost function around the estimated weight vector, i.e.,

$$C_N(\mathbf{w}) = C_N(\hat{\mathbf{w}}) + \delta\mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}})\delta\mathbf{w} \quad (5.125)$$

where

- $\delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}$.
- $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w})$.
- $\mathbf{J}(\hat{\mathbf{w}})$ is the Hessian matrix evaluated at $\hat{\mathbf{w}}$.

Essentially, this corresponds to considering the filter as linear (in the weights) in the vicinity of $\hat{\mathbf{w}}$. Hence, the convergence results of the LMS-algorithm, which applies to linear filters, are applicable. According to e.g., [Widrow & Stearns 85, Ch. 6], [Haykin 91, Sec. 5.9]²⁰: If

$$0 < \mu < \frac{2}{\lambda_{\max}} \quad (5.126)$$

where λ_{\max} is the largest eigenvalue of $\mathbf{J}_N(\hat{\mathbf{w}})$ then the algorithm is convergent on the average, q.e.²¹,

$$\langle \mathbf{w}(\ell) \rangle \rightarrow \hat{\mathbf{w}}, \quad \ell \rightarrow \infty \quad (5.127)$$

²⁰It should be emphasized that the results provided in the mentioned references concerns operating the algorithm in the on-line mode; however, the results are equivalent in the respect that ensemble averages is replaced with time-averages.

²¹If $\mathbf{J}_N(\mathbf{w}^*)$, is singular then a formally correct formulation is given by:

$$\inf_{\hat{\mathbf{w}} \in \mathcal{W}} \|\langle \mathbf{w}(\ell) \rangle - \hat{\mathbf{w}}\| \rightarrow 0, \quad \ell \rightarrow \infty$$

where \mathcal{W} is the set of weights which locally minimize $C_N(\mathbf{w})$.

where $\langle \cdot \rangle$ denotes the time-average²². The relaxation of $\langle \mathbf{w}(\ell) \rangle$ is exponential²³ and the relaxation of the individual modes, $i = 1, 2, \dots, m$ where $m = \dim(\mathbf{w})$, are expressed by the time constants:

$$\tau_i = \frac{-1}{\ln(1 - \mu\lambda_i)} \approx \frac{1}{\mu\lambda_i}, \quad \mu \ll 1 \quad (5.128)$$

where λ_i is the i 'th eigenvalue of $\mathbf{J}_N(\hat{\mathbf{w}})$. Notice that small eigenvalues correspond to modes with large time constants, i.e., slow convergence, and the minimal time constants are achieved when μ is large, i.e., $\mu = 2/\lambda_{\max}$. The mode which relaxes most slowly – i.e., the mode corresponding to λ_{\min} – achieves a minimal time constant approximately equal to 2χ , where $\chi = \lambda_{\max}/\lambda_{\min}$ is the eigenvalue spread. Consequently, if χ is large the SG-algorithm will display slow convergence. This is in keeping with the convergence properties of the GD-algorithm according to Sec. 5.3.1.

The fluctuations in $\mathbf{w}(\ell)$ around $\hat{\mathbf{w}}$ imply that on the average the cost at $\mathbf{w}(\ell)$ is larger than $C_N(\hat{\mathbf{w}})$. This is explicitly expressed by the dimensionless ratio, \mathcal{M} the *misadjustment* [Haykin 91, Sec. 5.12], given by:

$$\mathcal{M} = \frac{\langle C_N(\mathbf{w}(\ell)) \rangle - C_N(\hat{\mathbf{w}})}{C_N(\hat{\mathbf{w}})} = \frac{\mu \text{tr} \mathbf{J}_N(\hat{\mathbf{w}})}{2 - \mu \text{tr} \mathbf{J}_N(\hat{\mathbf{w}})} \quad (5.129)$$

where tr denotes the trace operator. The misadjustment is defined in the interval:

$$0 < \mu < \frac{2}{\text{tr} \mathbf{J}_N(\hat{\mathbf{w}})} \leq \frac{2}{\lambda_{\max}} \quad (5.130)$$

which cf. [Haykin 91, Sec. 5.12] also ensures convergence of $\langle C_N(\mathbf{w}(\ell)) \rangle$ ²⁴. Obviously, \mathcal{M} increases with μ and consequently, there will be a trade off between misadjustment and speed of convergence.

5.4.1.2 Time-Varying Step-Size

Using a time-varying step-size it seems natural to employ a large μ at start so that fast convergence is achieved. Consequently, μ should be reduced in order to eliminate misadjustment. The optimal reduction of μ is provided by the following theorem [Ljung & Söderström 83, Theorem 4.3., p. 182]²⁵: $\langle \mathbf{w}(\ell) \rangle$ is a consistent estimator of the estimated weights \mathcal{W} defined by pa:calw , i.e.,

$$\text{Prob} \left\{ \inf_{\hat{\mathbf{w}} \in \mathcal{W}} \|\langle \mathbf{w}(k) \rangle - \hat{\mathbf{w}}\| \rightarrow 0 \right\} = 1, \quad \text{as } \ell \rightarrow \infty \quad (5.131)$$

provided that the step-size complies with:

$$\ell \cdot \mu(\ell) \rightarrow \alpha, \quad \ell \rightarrow \infty \quad (5.132)$$

²²That is,

$$\langle \varphi(k) \rangle = \frac{1}{N} \sum_{k=1}^N \varphi(k).$$

²³If $1/\lambda_{\max} < \mu < 2/\lambda_{\max}$ an oscillating exponential relaxation appears.

²⁴Note that the convergence condition w.r.t. $\langle C_N(\mathbf{w}(\ell)) \rangle$ also ensures that $\langle \mathbf{w}(\ell) \rangle$ converges.

²⁵It should be noted that [Ljung & Söderström 83] consider the on-line operation mode of recursive algorithms. This involves a number of additional regularity conditions which is not required when considering the off-line mode. This is considered in Sec. 5.6. For further reference, the reader should consult the consistency results given in, [Battiti 92], [White 87], [White 89a].

where $\alpha > 0$.

Although the consistency result provides the asymptotic behavior of $\mu(\ell)$ it does not state anything about the magnitude. Moreover, the convergence results pa:condw, (5.130) merely provide the magnitude of $\mu(\ell)$ close to the optimum $\hat{\mathbf{w}}$, i.e., in the limit of large ℓ . Consequently – in outline – a proper selection of $\mu(\ell)$ still remains an open question. The literature provides a variety of heuristic strategies concerning step-size adjustment, e.g., [Hertz et al. 91, Sec. 6.2], [Silva & Almeida 90]; however, it is beyond the scope of the Thesis to discuss those subjects. Instead, we suggest a normalization which ensures that the step-size has a suitable order of magnitude. The approach is a generalization of the so-called α - LMS algorithm [Widrow et al. 88] which we shall denote the *normalized* SG-algorithm (see also e.g., [Dahl 87]). Consider the normalization,

$$\mu(\ell) = \frac{\alpha(\ell)}{|\boldsymbol{\psi}(k; \mathbf{w}(\ell - 1))|^2 + \frac{\kappa m}{N}} \quad (5.133)$$

where

- $\alpha(\ell) \geq 0$ is the *normalized step-size*. In order to ensure convergence: $0 \leq \alpha < 2$. This is due to the fact that the normalization in pa:asg forms an estimate of the trace of the Hessian matrix which enters the bounds given by pa:condst.
- $\boldsymbol{\psi}(k; \mathbf{w}(\ell - 1))$ is the instantaneous gradient vector.
- $m = \dim(\mathbf{w})$.
- κ is the regularization parameter.
- N is the number of training examples.

The normalization is motivated by the convergence condition pa:condst which states that the step-size should be inversely proportional to the trace of the Hessian. The Hessian matrix is here taken as the current Hessian, i.e., $\mathbf{J}_N(\mathbf{w}(\ell - 1))$ which yields²⁶:

$$\begin{aligned} \mathbf{J}_N(\mathbf{w}(\ell - 1)) &= \frac{1}{2} \frac{\partial^2 C_N(\mathbf{w}(\ell - 1))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\ &= \frac{\partial \nabla_N(\mathbf{w}(\ell - 1))}{\partial \mathbf{w}^\top} \\ &= \frac{\partial \left[-\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}(\ell - 1)) e(k; \mathbf{w}(\ell - 1)) + \kappa \mathbf{w} \right]}{\partial \mathbf{w}^\top} \\ &= \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}(\ell - 1)) \boldsymbol{\psi}^\top(k; \mathbf{w}(\ell - 1)) - \boldsymbol{\Psi}(k; \mathbf{w}(\ell - 1)) e(k; \mathbf{w}(\ell - 1)) \\ &\quad + \kappa \mathbf{I} \end{aligned} \quad (5.134)$$

where

$$\boldsymbol{\Psi}(k; \mathbf{w}) = \frac{\partial \boldsymbol{\psi}(k; \mathbf{w})}{\partial \mathbf{w}^\top} \quad (5.135)$$

²⁶Here we use the definition of the gradient pa:nablaexp and the fact: $\partial e / \partial \mathbf{w} = -\partial \hat{y} / \partial \mathbf{w}$.

As elaborated in Sec. 5.5, we usually neglect the sum of the terms, $\Psi(k; \mathbf{w})e(k; \mathbf{w})$, i.e.,

$$\tilde{\mathbf{J}}_N(\mathbf{w}(\ell-1)) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}(\ell-1))\boldsymbol{\psi}^\top(k; \mathbf{w}(\ell-1)) + \kappa \mathbf{I}. \quad (5.136)$$

This approximation is denoted the *pseudo Hessian*. Now, the trace becomes:

$$\begin{aligned} \text{tr } \tilde{\mathbf{J}}_N(\mathbf{w}(\ell-1)) &= \frac{1}{N} \sum_{k=1}^N \text{tr} \left[\boldsymbol{\psi}(k; \mathbf{w}(\ell-1))\boldsymbol{\psi}^\top(k; \mathbf{w}(\ell-1)) + \frac{\kappa}{N} \cdot \mathbf{I} \right] \\ &= \frac{1}{N} \sum_{k=1}^N \left[|\boldsymbol{\psi}(k; \mathbf{w}(\ell-1))|^2 + \frac{\kappa m}{N} \right]. \end{aligned} \quad (5.137)$$

Since we prefer a fully recursive algorithm the trace of the pseudo Hessian is replaced by the instant value which precisely is the denominator of the expression in pa:asg²⁷.

5.4.2 The SG-algorithm and Computational Complexity

To sum up we give accurate formulations of the SG-algorithm and the Normalized SG-algorithm (the NSG-algorithm).

Stochastic Gradient Algorithm Initialization:

- Step 1*
- a. Choose an initial weight vector $\mathbf{w}(0)$, cf. Sec. 5.3.2.
 - b. Specify the number of iterations, *itr*.
 - c. Choose the step-size sequence, $\mu(\ell)$, $\ell \in [1; N \text{ itr}]$.
 - c. Select the regularization parameter, κ and calculate the parameter $\nu = \kappa m/N$.
 - d. Initialize the counter $\ell = 0$.

Step 2 **for** $i = 0, 1, \dots, \text{itr} - 1$

for $k = 1, 2, \dots, N$

$\ell \leftarrow \ell + 1$

$\hat{\mathbf{y}}(\ell) = f(\mathbf{x}(k); \mathbf{w}(\ell))$ $CP_{\hat{\mathbf{y}}}$

$e(\ell) = y(k) - \hat{\mathbf{y}}(\ell)$

$e(\ell) \leftarrow \mu(\ell)e(\ell)$ 1

Compute the product $\boldsymbol{\psi}(\ell)e(\ell)$ $CP_{\boldsymbol{\psi}e}$

$\mathbf{w}(\ell) = (1 - \kappa\mu(\ell)) \mathbf{w}(\ell-1) + \boldsymbol{\psi}(\ell)e(\ell)$ $m + 1$

end

end

Notes:

²⁷An alternative is to perform a recursive update of this trace, see e.g., [Ljung & Söderström 83, Sec. 3.5.3].

- The number of iterations specifies the number of times the training set is replicated during training²⁸.
- The computation of $\boldsymbol{\psi}(\ell)e(\ell) \equiv \boldsymbol{\psi}(\ell; \mathbf{w}(\ell))e(\ell; \mathbf{w}(\ell))$ is in layered architectures performed by back-propagation cf. Sec. 5.3.3 and Sec. 5.3.4.
- The indices ℓ on all variables are included for the sake of clarity only; consequently, they can be omitted.

In the right column the computational complexity²⁹ CP involved in the actual line is listed. The computational complexity of the SG-algorithm is then

$$CP_{\text{SG}} = \begin{cases} N \text{ itr} \left(CP_{\hat{y}} + CP_{\boldsymbol{\psi}e} + 1 \right) & \kappa = 0 \\ N \text{ itr} \left(CP_{\hat{y}} + CP_{\boldsymbol{\psi}e} + m + 2 \right) & \kappa \neq 0 \end{cases} \quad (5.138)$$

where $CP_{\hat{y}}$ and $CP_{\boldsymbol{\psi}e}$ are the complexities involved in calculating one sample of the filter output and of the product of the instantaneous gradient vector and the error, respectively. These complexities are evidently highly dependent on the chosen filter architecture; however, recall the expressions listed in Sec. 5.3.4.1 when dealing with an MFPNN. In particular, cf. pa:ycom2, (5.122), for a 2-layer MFPNN architecture the complexity reads:

$$CP_{\text{SG}} \approx \begin{cases} N \text{ itr} \left(m \frac{2p+26}{p+2} + 1 \right) & \kappa = 0 \\ N \text{ itr} \left(m \frac{3p+28}{p+2} + 2 \right) & \kappa \neq 0 \end{cases} . \quad (5.139)$$

Normalized Stochastic Gradient (NSG) AlgorithmInitialization:

- Step 1*
- Choose an initial weight vector $\mathbf{w}(0)$, cf. Sec. 5.3.2.
 - Specify the number of iterations, itr .
 - Choose the normalized step-size sequence, $\alpha(\ell)$, $\ell \in [1; N \text{ itr}]$.
 - Select the regularization parameter, κ and calculate the parameter $\nu = \kappa m / N$.
 - Initialize the counter $\ell = 0$.

Step 2 **for** $i = 0, 1, \dots, \text{itr} - 1$

for $k = 1, 2, \dots, N$

$\ell \leftarrow \ell + 1$

$\hat{y}(\ell) = f(\mathbf{x}(k); \mathbf{w}(\ell))$ $CP_{\hat{y}}$

$e(\ell) = y(k) - \hat{y}(\ell)$

 Compute the gradient vector $\boldsymbol{\psi}(\ell)$ $CP_{\boldsymbol{\psi}}$

$\mu(\ell) = \frac{\alpha(\ell)}{|\boldsymbol{\psi}(k; \mathbf{w}(\ell - 1))|^2 + \nu}$ $m + 10$

$e(\ell) \leftarrow \mu(\ell)e(\ell)$ 1

²⁸In the literature iterations also are denoted “passes” or “epochs”.

²⁹Measured as the total number of multiplications and divisions.

$$\mathbf{w}(\ell) = (1 - \kappa\mu(\ell)) \mathbf{w}(\ell - 1) + \boldsymbol{\psi}(\ell)e(\ell) \quad 2m + 1$$

end

end

The computational complexity of the NSG-algorithm is then

$$CP_{\text{NSG}} = \begin{cases} N \text{ itr} \left(CP_{\hat{y}} + CP_{\boldsymbol{\psi}} + 2m + 11 \right) & \kappa = 0 \\ N \text{ itr} \left(CP_{\hat{y}} + CP_{\boldsymbol{\psi}} + 3m + 12 \right) & \kappa \neq 0 \end{cases}. \quad (5.140)$$

In the case of a 2-layer MFPNN architecture then cf. pa:ycom2, (5.120)

$$CP_{\text{NSG}} \approx \begin{cases} N \text{ itr} \left(m \frac{4p+28}{p+2} + 11 \right) & \kappa = 0 \\ N \text{ itr} \left(m \frac{5p+30}{p+2} + 12 \right) & \kappa \neq 0 \end{cases}. \quad (5.141)$$

5.5 The Modified Gauss-Newton Algorithm

The off-line Newton-Raphson approach is based on successive second order Taylor series expansions of the cost function. Let $\mathbf{w}_{(i-1)}$ be the previous weight estimate, i.e., the weight estimate at iteration $i - 1$. The cost function is approximated by:

$$C_N(\mathbf{w}) \approx C_N(\mathbf{w}_{(i-1)}) + 2\nabla_N(\mathbf{w}_{(i-1)})\delta\mathbf{w} + \delta\mathbf{w}^\top \mathbf{J}_N(\mathbf{w}_{(i-1)})\delta\mathbf{w} \quad (5.142)$$

where

- $\delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{(i-1)}$.
- $\nabla_N(\cdot)$ is the gradient of the cost function which cf. pa:gradient, (5.36) is given by:

$$\begin{aligned} \nabla_N(\mathbf{w}_{(i-1)}) &= \frac{1}{2} \cdot \frac{\partial C_N(\hat{\mathbf{w}}_{(i-1)})}{\partial \mathbf{w}} \\ &= -\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}_{(i-1)})e(k; \mathbf{w}_{(i-1)}) + \kappa\mathbf{w}_{(i-1)} \end{aligned} \quad (5.143)$$

with

$$\boldsymbol{\psi}(k; \mathbf{w}_{(i-1)}) = \frac{\partial f(\mathbf{x}(k); \mathbf{w}_{(i-1)})}{\partial \mathbf{w}} \quad (5.144)$$

$$e(k; \mathbf{w}_{(i-1)}) = y(k) - f(\mathbf{x}(k); \mathbf{w}_{(i-1)}). \quad (5.145)$$

- $\mathbf{J}_N(\cdot)$ is the Hessian matrix which cf. pa:fulhes

$$\begin{aligned} \mathbf{J}_N(\mathbf{w}_{(i-1)}) &= \frac{1}{2} \frac{\partial^2 C_N(\mathbf{w}_{(i-1)})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\ &= \frac{\partial \nabla_N(\mathbf{w}_{(i-1)})}{\partial \mathbf{w}^\top} \\ &= \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}_{(i-1)})\boldsymbol{\psi}^\top(k; \mathbf{w}_{(i-1)}) - \boldsymbol{\Psi}(k; \mathbf{w}_{(i-1)})e(k; \mathbf{w}_{(i-1)}) \\ &\quad + \kappa \mathbf{I} \end{aligned} \quad (5.146)$$

with

$$\Psi(k; \mathbf{w}) = \frac{\partial \psi(k; \mathbf{w})}{\partial \mathbf{w}^\top} = \frac{\partial^2 f(\mathbf{x}(k); \mathbf{w})}{\partial \partial \mathbf{w}} \mathbf{w}^\top. \quad (5.147)$$

Requiring that the weights of the current iteration, $\mathbf{w}_{(i)}$, minimize the cost function – i.e., $\partial C_N(\mathbf{w}_{(i)})/\partial \mathbf{w} = \mathbf{0}$ – leads to the weight update

$$\Delta \mathbf{w}_{(i-1)} = \mathbf{w}_{(i)} - \mathbf{w}_{(i-1)} = -\mathbf{J}_N^{-1}(\mathbf{w}_{(i-1)}) \nabla N(\mathbf{w}_{(i-1)}). \quad (5.148)$$

Obviously, the invertibility of the Hessian is presupposed. This update is known as a Newton-Raphson iteration [Seber & Wild 89, Sec. 13.3]. If the model is an LX-model – i.e., linear in the weights – then the cost function is indeed quadratic (see e.g., pa:costlin in Sec. 5.2), q.e., pa:mngntayl is not an approximation. Consequently, the cost function is minimized by performing *one* Newton-Raphson iteration irrespective of how $\mathbf{w}_{(i-1)}$ is chosen. On the other hand, dealing with general NN-models with non-quadratic cost functions usually several iterations are required in order to obtain a reasonable solution. Generally, we use a modification of the Newton-Raphson iteration. The reader is e.g., referred to [Seber & Wild 89, Sec. 13.3 & 14.1–14.3] for a review of various modifications. Here we merely mention the so-called *modified Gauss-Newton* (MGN)³⁰ approach which comprises the following modifications:

- The Hessian is in general not restricted to be positive semidefinite. Suppose for instance that $\mathbf{w}_{(i-1)}$ is close to a local maximum then the Hessian is negative definite, and consequently, the weight update does not become a descent direction cf. pa:descent. A possible modification of the Hessian so that positive semidefiniteness is ensured consists in neglecting the term

$$-\frac{1}{N} \sum_{k=1}^N \Psi(k; \mathbf{w}_{(i-1)}) e(k; \mathbf{w}_{(i-1)})$$

according to pa:mgnhes. The modified Hessian is denoted the *pseudo Hessian* defined by:

$$\tilde{\mathbf{J}}_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N \psi(k; \mathbf{w}) \psi^\top(k; \mathbf{w}) + \kappa \mathbf{I}. \quad (5.149)$$

Obviously, the pseudo Hessian is positive semidefinite³¹ and the direction $-\tilde{\mathbf{J}}_N^{-1}(\mathbf{w}_{(i-1)}) \nabla(\mathbf{w}_{(i-1)})$ thus becomes a descent direction cf. pa:descent.

An additional benefit of using the pseudo Hessian approach is a – normally – tremendous reduction in the computational complexity. This is due to the fact that the first term of the pseudo Hessian is an outer product of the instantaneous gradient vector, ψ , which already has been calculated in order to determine the gradient ∇ cf. pa:mgnnab.

³⁰Also known as the damped Gauss-Newton and the “method of scoring”.

³¹This is seen by expressing the pseudo Hessian as: $\mathbf{J}_N = \mathbf{B}^\top \mathbf{B} + \kappa \mathbf{I}$ where \mathbf{B} is the $N \times m$ matrix:

$$\mathbf{B} = \begin{bmatrix} \psi^\top(1; \mathbf{w}) \\ \vdots \\ \psi^\top(N; \mathbf{w}) \end{bmatrix}.$$

\mathbf{J} is positive semidefinite if $\forall \mathbf{v} \neq \mathbf{0}: \mathbf{v}^\top \mathbf{J}_N \mathbf{v} \geq 0$. That is, let $\mathbf{v}_1 = \mathbf{B} \mathbf{v}$ then $\mathbf{v}^\top \mathbf{J}_N \mathbf{v} = |\mathbf{v}_1|^2 + \kappa |\mathbf{v}|^2 \geq 0$ as $\kappa \geq 0$.

The validity of the pseudo Hessian approach may be supported by the following facts:

- First notice that the discussion is relevant when $\mathbf{w}_{(i)}$ is close to $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w})$ only.
- If the error, $e(k; \hat{\mathbf{w}})$, is small then one would expect that the neglected term is small.
- If the model is nearly linear in the weights then $\Psi(k; \mathbf{w})$ is close to zero³², and consequently, the neglected term becomes small.
- When the model is capable of modeling the system³³ then the error, $e(k; \hat{\mathbf{w}})$, tends to become independent of the input, and thereby $\Psi(k; \hat{\mathbf{w}})$. That means

$$-\frac{1}{N} \sum_{k=1}^N \Psi(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \approx - \left[\frac{1}{N} \sum_{k=1}^N \Psi(k; \hat{\mathbf{w}}) \right] \cdot \left[\frac{1}{N} \sum_{k=1}^N e(k; \hat{\mathbf{w}}) \right]. \quad (5.150)$$

When κ is small the time-average of the error equals zero approximately cf. pa:errav, i.e., the neglected term is small.

- The quadratic cost function approximation in pa:mgntayl is valid only when $\mathbf{w}_{(i)}$ is close to $\mathbf{w}_{(i-1)}$. However, the update $\mathbf{w}_{(i)} - \mathbf{w}_{(i-1)} = -\tilde{\mathbf{J}}_N^{-1}(\mathbf{w}_{(i-1)}) \nabla_N(\mathbf{w}_{(i-1)})$ may possibly result in that the quadratic approximation is violated. Hence, it may be profitable to introduce a step-size, $\mu_{(i)} \geq 0$ which modifies the length of the weight update so that the cost is decreased, i.e., $C_N(\mathbf{w}_{(i)}) < C_N(\mathbf{w}_{(i-1)})$.

The modified Gauss-Newton iteration is then given by:

$$\mathbf{w}_{(i)} = \mathbf{w}_{(i-1)} - \mu_{(i)} \tilde{\mathbf{J}}_N^{-1}(\mathbf{w}_{(i-1)}) \nabla_N(\mathbf{w}_{(i-1)}). \quad (5.151)$$

In Fig. 5.7 an example of one MGN iteration is shown. When using the MGN approach the convergence is quadratic [Seber & Wild 89, Sec. 13.2.3], i.e.,

$$\left| C_N(\mathbf{w}_{(i)}) - C_N(\hat{\mathbf{w}}) \right| = \text{const} \cdot \left| C_N(\mathbf{w}_{(i-1)}) - C_N(\hat{\mathbf{w}}) \right|^2. \quad (5.152)$$

This is indeed much faster than the linear convergence of the GD-algorithm. However, if the pseudo Hessian is a poor approximation to the Hessian in the vicinity of $\hat{\mathbf{w}}$ the RGN approach may result in slow linear convergence [Seber & Wild 89, Sec. 14.1]. Another advantage of the MGN is that it is almost independent of the eigenvalue spread of the Hessian contrary to the GD-algorithm. This is due to the fact that the inverse Hessian enters the weight update. Contemplate for simplicity the case where the Hessian is diagonal, i.e., $\tilde{\mathbf{J}}_N^{-1}(\mathbf{w}_{i-1}) = \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_m\}$. In that case, the weights are updated individually as

$$\Delta w_{(i)} = \mu_{(i)} \lambda_i^{-1} \cdot \nabla_{N,i} \quad (5.153)$$

where $\nabla_{N,i}$ is the the i 'th component of $\nabla_N(\mathbf{w}_{(i-1)})$. Each weight thus has its own effective step-size $\mu_{(i)} \lambda_i^{-1}$ which is large when the eigenvalue, λ_i , is small and vice versa. This is in keeping with the fact that speed of convergence is proportional to the eigenvalues – or

³²Note that $\Psi(k; \mathbf{w}) \equiv \mathbf{0}$ for LX-models.

³³A more accurate definition is given in Ch. 6 where we shall denote such a model a complete model. Furthermore, the transition of the Hessian into the pseudo Hessian is discussed in various contexts.

curvature of the cost function (see e.g., Sec. 5.4). Consequently, all weights have equal speed of convergence irrespective of the eigenvalue spread.

In summary, the MGN-algorithm becomes:

Modified Gauss-Newton Algorithm Initialization:

- Step 1*
- a. Choose an initial weight vector $\mathbf{w}_{(0)}$ cf. Sec. 5.3.2 and determine the cost, $C_N(\mathbf{w}_{(0)})$.
 - b. Let the “old” cost $C_N(\mathbf{w}_{(i-1)}) = \infty$ ³⁴.
 - c. Select a threshold, $\tau \ll 1$, specifying the minimum relative change in the cost function.
 - d. Select the maximum number of iterations, itr .
 - e. Select the maximum step-size, μ_{\max} .
 - f. Select the regularization parameter, κ .
 - g. Initialize the counter $i = 0$.

Step 2 **while** $(i < itr) \wedge \left(\frac{C_N(\mathbf{w}_{(i-1)}) - C_N(\mathbf{w}_{(i)})}{C_N(\mathbf{w}_{(i-1)})} \geq \tau \right)$

Increment the counter i

Calculate the gradient, $\nabla_N(\mathbf{w}_{(i-1)})$, and
the pseudo Hessian, $\tilde{\mathbf{J}}_N(\mathbf{w}_{(i-1)})$

Determine the update: $\mathbf{u} = -\tilde{\mathbf{J}}_N^{-1}(\mathbf{w}_{(i-1)})\nabla_N(\mathbf{w}_{(i-1)})$

Set $C_N(\mathbf{w}_{(i)}) = \infty$ and $\mu = \mu_{\max}$

while $C_N(\mathbf{w}_{(i)}) > C_N(\mathbf{w}_{(i-1)})$

$\mathbf{w}_{(i)} = \mathbf{w}_{(i-1)} + \mu\mathbf{u}$

Calculate $C_N(\mathbf{w}_{(i)})$

$\mu \leftarrow \mu/2$

end

end

Note:

- The weight updating is terminated when $i = itr$ or when the stop criterion (see also [Seber & Wild 89, Sec. 15.2.4]):

$$\frac{C_N(\mathbf{w}_{(i-1)}) - C_N(\mathbf{w}_{(i)})}{C_N(\mathbf{w}_{(i-1)})} < \tau \quad (5.154)$$

is met. τ is a suitable threshold (e.g., 10^{-6}) which specifies the minimum significant relative change in the cost function.

³⁴That is, equal to a very large number.

5.6 Recursive Gauss-Newton Algorithm

In this section we consider a modification of the LS cost function with weight decay regularizer pa:costfun. Define the cost function:

$$C_k(\mathbf{w}) = \mu(k) \sum_{n=1}^k \beta(k, n) e^2(n; \mathbf{w}) + \gamma \mu(k) \beta(k, 0) \|\mathbf{w} - \mathbf{w}(0)\|^2, \quad k = 1, 2, \dots, N \quad (5.155)$$

where

- $\beta(k, n)$ is the weighting function (see also Sec. 5.1.2.1) which in accordance with [Ljung 87, Ch. 11.2] is assumed to obey the recursive property:

$$\beta(k, n) = \begin{cases} \lambda(k) \beta(k-1, n) & 0 \leq n \leq k-1 \\ 1 & n = k \end{cases} \quad (5.156)$$

where $0 < \lambda(k) \leq 1$ is the instantaneous forgetting factor. Writing out pa:betarek we get

$$\beta(k, n) = \begin{cases} \prod_{j=n+1}^k \lambda(j) & 0 \leq n \leq k-1 \\ 1 & n = k \end{cases}. \quad (5.157)$$

A common choice is $\lambda(k) = \lambda$ which results in

$$\beta(k, n) = \lambda^{k-n}. \quad (5.158)$$

That is, exponential weighting. $0 < \lambda \leq 1$ is consequently often called the exponential forgetting factor. That is the most recent samples of the error are weighted more than old samples. This is in particular important when dealing with time-varying systems (non-stationary environments) in order to track changes. In addition, also in the case of fixed systems (stationary signals) it may be useful to employ weighting. This is elaborated below.

- $\mu(k)$ is a normalization constant

$$\mu(k) = \begin{cases} 1 & k = 0 \\ \left[\sum_{n=1}^k \beta(k, n) \right]^{-1} & k \geq 1 \end{cases}. \quad (5.159)$$

$\mu^{-1}(k)$ may be interpreted as the effective memory length, i.e., the effective number of samples entering the cost function. In addition, it will be demonstrated below that $\mu(k)$ in fact also defines a time-varying step-size.

In the case $\lambda(k) = \lambda$, we get:

$$\mu(k) = \begin{cases} \frac{1-\lambda}{1-\lambda^k} & \lambda < 1, k \geq 1 \\ k^{-1} & \lambda = 1, k \geq 1 \end{cases}. \quad (5.160)$$

Furthermore, using pa:betarek the following recursion for $\mu(k)$ is attainable:

$$\frac{1}{\mu(k)} = \frac{\lambda(k)}{\mu(k-1)} + 1, \quad k = 2, 3, \dots, N \quad (5.161)$$

with $\mu(1) = 1$. Note that $0 < \mu(k) \leq 1$ as $0 < \lambda \leq 1$.

- The regularization parameter is defined by:

$$\kappa = \gamma\mu(k)\beta(k, 0) \quad (5.162)$$

where γ is the algorithm parameter adjusted so that the desired κ is achieved. Below it is demonstrated that γ acts as an initialization parameter of the Hessian.

- The weights are regularized against the initial weight vector, $\mathbf{w}(0)$, which not necessarily equals zero, cf. Sec. 5.3.2, as in traditional weight decay regularization.

In Fig. 5.8 a typical progress of the parameters is depicted. A recursive algorithm based on the Gauss-Newton approach, cf. Sec. 5.5, consists in performing one Gauss-Newton iteration each time a data sample is collected³⁵. The presented derivation is based on [Ljung 87, Sec. 11.4] which denotes the algorithm: The “recursive prediction error algorithm”.

Let $\mathbf{w}(k)$ be the weight estimate at time k , i.e., the approximation of $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_k(\mathbf{w})$. Now, according to pa:gcnitr with $\mathbf{w}_{(i)} = \mathbf{w}(k)$, $\mathbf{w}_{(i-1)} = \mathbf{w}(k-1)$ and the step-size $\mu_{(i)} = 1$:

$$\mathbf{w}(k) = \mathbf{w}(k-1) - \tilde{\mathbf{J}}_k^{-1}(\mathbf{w}(k-1))\nabla_k(\mathbf{w}(k-1)). \quad (5.163)$$

The gradient is cf. pa:cstrgn, (5.34)

$$\begin{aligned} \nabla_k(\mathbf{w}) &= \frac{1}{2} \cdot \frac{\partial C_k(\mathbf{w})}{\partial \mathbf{w}} \\ &= -\mu(k) \sum_{n=1}^k \beta(k, n) \boldsymbol{\psi}(n; \mathbf{w}) e(n; \mathbf{w}) + \gamma\mu(k)\beta(k, 0) (\mathbf{w} - \mathbf{w}(0)). \end{aligned} \quad (5.164)$$

Note initially, i.e., $k = 0$, the normal equations yields $\nabla_0(\mathbf{w}) = \mathbf{0}$ which results in $\hat{\mathbf{w}} = \mathbf{w}(0)$.

Using pa:betarek, (5.161) then:

$$\begin{aligned} \nabla_k(\mathbf{w}(k-1)) &= -\mu(k) \sum_{n=1}^k \beta(k, n) \boldsymbol{\psi}(n; \mathbf{w}(k-1)) e(n; \mathbf{w}(k-1)) \\ &\quad + \gamma\mu(k)\beta(k, 0) (\mathbf{w}(k-1) - \mathbf{w}(0)) \\ &= -\mu(k)\lambda(k) \sum_{n=1}^{k-1} \beta(k-1, n) \boldsymbol{\psi}(n; \mathbf{w}(k-1)) e(n; \mathbf{w}(k-1)) \\ &\quad + \gamma\mu(k)\lambda(k)\beta(k-1, 0) (\mathbf{w}(k-1) - \mathbf{w}(0)) \\ &\quad - \mu(k)\beta(k, k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) e(k; \mathbf{w}(k-1)) \\ &= \frac{\mu(k)\lambda(k)}{\mu(k-1)} \nabla_{k-1}(\mathbf{w}(k-1)) - \mu(k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) e(k; \mathbf{w}(k-1)) \\ &= (1 - \mu(k)) \nabla_{k-1}(\mathbf{w}(k-1)) - \mu(k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) e(k; \mathbf{w}(k-1)). \end{aligned} \quad (5.165)$$

³⁵The derivation presented here deals with the on-line mode recursive Gauss-Newton algorithm; however, by minor modifications the off-line mode version appears without further ado.

Assuming that $\mathbf{w}(k-1)$ actually minimizes $C_{k-1}(\mathbf{w})$ then $\nabla_{k-1}(\mathbf{w}(k-1)) = \mathbf{0}$, i.e.,

$$\nabla_k(\mathbf{w}(k-1)) = -\mu(k)\boldsymbol{\psi}(k; \mathbf{w}(k-1))e(k; \mathbf{w}(k-1)). \quad (5.166)$$

In general this is only an approximation. However, dealing with LX-models (see pa:lxmodel) then the weight vector converge in only one Gauss-Newton iteration cf. Sec. 5.5; consequently, the assumption is satisfied.

The pseudo Hessian is cf. pa:cstrgn, (5.149)

$$\begin{aligned} \tilde{\mathbf{J}}_k(\mathbf{w}) &= \frac{1}{2} \frac{\partial^2 C_k(\mathbf{w})}{\partial \boldsymbol{\theta} \partial \mathbf{w}} \mathbf{w}^\top \\ &= \mu(k) \sum_{n=1}^k \beta(k, n) \boldsymbol{\psi}(n; \mathbf{w}) \boldsymbol{\psi}^\top(n; \mathbf{w}) + \gamma \mu(k) \beta(k, 0) \mathbf{I} \end{aligned} \quad (5.167)$$

and the initial value, i.e., $k=0$, yields $\tilde{\mathbf{J}}_0(\mathbf{w}) = \gamma \mathbf{I}$.

Now, cf. pa:betarek, (5.161):

$$\begin{aligned} \tilde{\mathbf{J}}_k(\mathbf{w}(k-1)) &= \mu(k) \sum_{n=1}^k \beta(k, n) \boldsymbol{\psi}(n; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(n; \mathbf{w}(k-1)) + \gamma \mu(k) \beta(k, 0) \mathbf{I} \\ &= \mu(k) \lambda(k) \sum_{n=1}^{k-1} \beta(k-1, n) \boldsymbol{\psi}(n; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(n; \mathbf{w}(k-1)) \\ &\quad + \gamma \mu(k) \lambda \beta(k-1, 0) \mathbf{I} + \mu(k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(k; \mathbf{w}(k-1)) \\ &= (1 - \mu(k)) \tilde{\mathbf{J}}_{k-1}(\mathbf{w}(k-1)) \\ &\quad + \mu(k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(k; \mathbf{w}(k-1)). \end{aligned} \quad (5.168)$$

$\tilde{\mathbf{J}}_{k-1}(\mathbf{w}(k-1))$ is not accessible and we are therefore content with the approximation $\tilde{\mathbf{J}}_{k-1}(\mathbf{w}(k-2))$ provided in the previous time step, i.e., the recursion becomes:

$$\tilde{\mathbf{J}}_k(\mathbf{w}(k-1)) = (1 - \mu(k)) \tilde{\mathbf{J}}_{k-1}(\mathbf{w}(k-2)) + \mu(k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(k; \mathbf{w}(k-1)). \quad (5.169)$$

If we employ an LX-model the (pseudo) Hessian does not depend on the weight vector; consequently, the recursion pa:hesupdat is exact. In general, the presence $\mu(k)$ in pa:hesupdat enables a trade off between the old estimate, $\tilde{\mathbf{J}}_{k-1}(\mathbf{w}(k-2))$, and the recent update, $\boldsymbol{\psi}(k; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(k; \mathbf{w}(k-1))$.

Combining pa:wupdat, (5.166), (5.169) then the recursive algorithm yields:

$$\begin{aligned} \tilde{\mathbf{J}}_k(\mathbf{w}(k-1)) &= (1 - \mu(k)) \tilde{\mathbf{J}}_{k-1}(\mathbf{w}(k-2)) + \mu(k) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) \boldsymbol{\psi}^\top(k; \mathbf{w}(k-1)), \\ \mathbf{w}(k) &= \mathbf{w}(k-1) + \mu(k) \tilde{\mathbf{J}}_k^{-1}(\mathbf{w}(k-1)) \boldsymbol{\psi}(k; \mathbf{w}(k-1)) e(k; \mathbf{w}(k-1)). \end{aligned} \quad (5.170)$$

When dealing with a nonlinear minimization task the Hessian truly varies much at first since it is evaluated at different estimates $\mathbf{w}(k)$; subsequently, it should be profitable to have $\mu(k)$ large in order to track this change. However, when the weights are converged then the Hessian is almost constant and $\mu(k)$ can be reduced (see also Fig. 5.8). Furthermore, note that $\mu(k)$ in fact acts like a time-varying step-size.

In [Ljung & Söderström 83, Theorem 4.3, p. 182] it is shown that $\mathbf{w}(k)$ is a consistent estimator³⁶ of the optimal weights \mathcal{W}^* defined by pa:calwast, i.e.,

$$\text{Prob} \left\{ \inf_{\mathbf{w}^* \in \mathcal{W}^*} \|\mathbf{w}(k) - \mathbf{w}^*\| \rightarrow 0 \right\} = 1, \quad \text{as } k \rightarrow \infty. \quad (5.171)$$

The necessary prerequisites are:

- The step-size complies with:

$$k \cdot \mu(k) \rightarrow \alpha, \quad k \rightarrow \infty \quad (5.172)$$

where $\alpha > 0$. This is e.g., accomplished with $\alpha = 1$ by choosing the forgetting factor scheme:

$$\lambda(k) = 1 - \eta \cdot a^k, \quad k = 0, 1, \dots \quad (5.173)$$

where $\eta = 1 - \lambda(0)$ and a specifies the rate with which $\lambda(k)$ approaches 1. In practice pa:lamsch is implemented by the first-order recursive filter:

$$\lambda(k) = a\lambda(k-1) + (1-a) \quad (5.174)$$

with $\lambda(0) = 1 - \eta$. A brief proof which shows the claimed property proceeds as follows: Substituting pa:betarek into pa:myrek, using pa:lamsch, and finally expanding yields:

$$\begin{aligned} \mu^{-1}(k) &= \sum_{n=1}^k \prod_{j=n+1}^k \lambda(j) \\ &= k + \sum_{i \in \mathcal{I}} b_i \cdot \exp \left[\ln a \left(\frac{k(k+1)}{2} - i \right) \right] \end{aligned} \quad (5.175)$$

where b_i are constants and $\mathcal{I} = \{i_1, i_2, \dots, i_{\max}\}$ with

$$i_1 = 1, \quad i_{\max} = \frac{k(k+1) - 1}{2}, \quad |\mathcal{I}| < \frac{k(k+1)}{2} - 1. \quad (5.176)$$

In consequence,

$$k^{-1} \mu^{-1}(k) = 1 + \sum_{i \in \mathcal{I}} b_i \cdot \frac{\exp \left[\ln a \left(\frac{k(k+1)}{2} - i \right) \right]}{k} \rightarrow 1, \quad k \rightarrow \infty. \quad (5.177)$$

- The data in the training set are stationary processes, and

$$\begin{aligned} &\boldsymbol{\psi}(k)e(k), \\ &\boldsymbol{\psi}(k)\boldsymbol{\psi}^\top(k) - \widetilde{\mathbf{H}}_k \end{aligned}$$

are mean-ergodic sequences (see e.g., p. 103)³⁷.

³⁶Here the consistency result is formulated w.r.t. operating the algorithm in the on-line mode; however, a similar result is obtained in the off-line mode case, see e.g., Sec. 5.4.1.2.

³⁷This is fulfilled if the input vector is a strictly stationary strongly mixing sequence (see e.g., p. 180) and the system pa:system is time-invariant.

- The pseudo Hessian $\tilde{\mathbf{J}}_k(\mathbf{w}(k-1))$ is positive definite $\forall k$.

The weight update `pa:wupdat1` requires the inverse pseudo Hessian thus involving a computational expensive matrix inversion each time step. Instead, one may apply the matrix inversion lemma (see e.g., App. B) in order to obtain a recursion for the inverse pseudo Hessian. Define the non-normalized inverse pseudo Hessian:

$$\mathbf{P}(k) = \mu(k) \tilde{\mathbf{H}}_k^{-1}. \quad (5.178)$$

According to [Ljung 87, Sec. 11.7] then the recursion yields:

$$\mathbf{P}(k) = \frac{\mathbf{P}(k-1)}{\lambda(k)} - \frac{\frac{\mathbf{P}(k-1)}{\lambda(k)} \boldsymbol{\psi}(k) \boldsymbol{\psi}^\top(k) \frac{\mathbf{P}(k-1)}{\lambda(k)}}{1 + \boldsymbol{\psi}^\top(k) \frac{\mathbf{P}(k-1)}{\lambda(k)} \boldsymbol{\psi}(k)}}. \quad (5.179)$$

This recursion is, however, known to have poor numerical properties [Ljung 87, Sec. 11.7] since round-off errors accumulate and make $\mathbf{P}(k)$ indefinite. This inexpediency can be circumvented by performing a factorization of $\mathbf{P}(k)$ which is discussed in Sec. 5.7.

To summarize, the recursive Gauss-Newton (RGN) algorithm³⁸ yields:

Recursive Gauss-Newton Algorithm Initialization:

- Step 1*
- a. Choose an initial weight vector $\mathbf{w}(0)$, cf. Sec. 5.3.2.
 - b. Select the regularization parameter, κ .
 - c. Specify the number of iterations, *itr*, i.e., the number of times the training set is replicated during training.
 - d. Choose a scheme – e.g., `pa:lamreki` – for updating the instantaneous forgetting factor, $\lambda(\ell)$, $\ell \in [1; N \text{ itr}]$.
 - e. Initialize the inverse pseudo Hessian $\mathbf{P}(0) = \gamma^{-1} \mathbf{I}$. γ is cf. `pa:kappaeqn` given by:

$$\gamma = \frac{\kappa}{\mu(N \text{ itr}) \beta(N \text{ itr}, 0)}. \quad (5.180)$$

- f. If μ is updated cf. the line denoted † in *Step 2.* below then initialize: $\mu^{-1}(0) = 1$.
- g. Initialize the counter, $\ell = 0$.

Step 2 **for** $i = 0, 1, \dots, \text{itr} - 1$

for $k = 1, 2, \dots, N$

$\ell \leftarrow \ell + 1$

$$\lambda^{-1} = \frac{1}{\lambda(\ell)} \quad 10$$

$$\mu^{-1}(\ell) = \lambda(\ell) \mu^{-1}(\ell - 1) + 1 \quad 1 \quad \dagger$$

$$\hat{\mathbf{y}}(\ell) = f(\mathbf{x}(k); \mathbf{w}(\ell)) \quad CP_{\hat{\mathbf{y}}}$$

³⁸Note that the RGN-algorithm is formulated in the off-line mode; however, the on-line mode algorithm simply results by replacing ℓ with k and skipping the i loop.

$$\begin{array}{ll}
e(\ell) = y(k) - \hat{y}(\ell) & \\
\mathbf{P}(\ell - 1) \leftarrow \lambda^{-1} \mathbf{P}(\ell - 1) & \frac{m^2 + m}{2} \\
\text{Compute } \boldsymbol{\psi}(\ell) & CP_{\boldsymbol{\psi}} \\
\boldsymbol{\nu}(\ell) = \mathbf{P}(\ell - 1) \boldsymbol{\psi}(\ell) & m^2 \\
\boldsymbol{\kappa}(\ell) = \frac{\boldsymbol{\nu}(\ell)}{1 + \boldsymbol{\psi}^\top(\ell) \boldsymbol{\nu}(\ell)} & 2m + 10 \\
\mathbf{w}(\ell) = \mathbf{w}(\ell - 1) + \boldsymbol{\kappa}(\ell) e(\ell) & m \\
\mathbf{P}(\ell) = \mathbf{P}(\ell - 1) - \boldsymbol{\kappa}(\ell) \boldsymbol{\nu}^\top(\ell) & \frac{m^2 + m}{2} \\
\text{end} & \\
\text{end} &
\end{array}$$

Notes:

- The algorithm passes through the training set several times specified by the number of iterations *itr*, i.e., the number of times the training set is replicated.
- The computation of the instantaneous gradient $\boldsymbol{\psi}(\ell) \equiv \boldsymbol{\psi}(\ell; \mathbf{w}(\ell))$ is in layered architectures performed by back-propagation cf. Sec. 5.3.3 and Sec. 5.3.4.
- The index ℓ on all variables is included for the sake of clarity only and can consequently be omitted.
- The update of $\mu(\ell)$ is only necessary when a final estimate of the inverse pseudo Hessian is required subsequently. That is, for $\ell = N \text{ itr}$ the estimated weights yield $\hat{\mathbf{w}} = \mathbf{w}(\ell)$ and according to pa:iphdef we set:

$$\widetilde{\mathbf{H}}_N^{-1}(\hat{\mathbf{w}}) = \mu^{-1}(\ell) \mathbf{P}(\ell). \quad (5.181)$$

- $\boldsymbol{\nu}$ and $\boldsymbol{\kappa}$ (often called the Kalman gain) are m -dimensional auxiliary variables which serve the purpose of reducing the computational burden.
- In every line of the algorithm the computational complexity is listed³⁹. Since \mathbf{P} is a symmetric matrix, i.e., only $(m^2 + m)/2$ terms have to be calculated.

Neglecting the complexity of the line indicated by †, and adding up the individual complexity components, the computational complexity of the RGN-algorithm yields:

$$CP_{\text{RGN}} = N \text{ itr} \left(CP_{\hat{y}} + CP_{\boldsymbol{\psi}} + 2m^2 + 4m + 20 \right). \quad (5.182)$$

Considering a 2-layer MFPNN using the approximate result cf. pa:ycom2, and pa:cpsi2l:

$$CP_{\text{RGN}} = N \text{ itr} \left(2m^2 + m \frac{6p + 22}{p + 2} + 20 \right). \quad (5.183)$$

as $p = \dim(\mathbf{z})$ denotes the number of input neurons.

In the neural network literature similar algorithms have been proposed [Chen et al. 90a], [Chen et al. 90b], [Kollias & Anastassiou 89]. In that context note the following matters:

³⁹Recall that this measured as the number of multiplications and divisions and one division is assumed to be equivalent to 10 multiplications

- The algorithms presented in [Chen et al. 90a] and [Chen et al. 90b] do not emphasize that the algorithms indeed minimize the modified cost defined in pa:cstrgn.
- In [Chen et al. 90a] it is noted that dealing with layered architectures we need to back-propagate the $\delta^{(r)}$ vector (cf. Sec. 5.3.3) only in order to find the instantaneous gradient vector $\psi(k)$. On the other hand, in [Kollias & Anastassiou 89], is suggested to back-propagate two terms. First the $\delta^{(r)}$ vector (with dimension m_r) in order to determine the product $\psi(k)e(k)$. Secondly, a $m_r \times m_r$ matrix $\beta^{(r)}$ in order to update the inverse pseudo Hessian. This approach thus seems inefficient.
- Both [Chen et al. 90b] and [Kollias & Anastassiou 89] suggest – in connection with the MFPNN architecture – to use block-diagonal pseudo Hessian approach. This consists in neglecting terms which correspond to weights of different neurons. In other words, we deal with one pseudo Hessian per neuron, and the Hessians are independent. This naturally diminish the performance of the algorithm; however, the complexity is reduced significantly. In addition, it enables a parallel implementation since each neuron is updated individually.

5.7 Recursive Gauss-Newton Algorithm with Bierman Factorization

The aim of the Recursive Gauss-Newton Algorithm with Bierman Factorization (RGNB) is to remedy the numerical stability problems with the RGN-algorithm. This is done by factorizing (or decomposing) \mathbf{P} according to the U-D factorization [Bierman 75]:

$$\mathbf{P} = \mathbf{U}\mathbf{D}\mathbf{U}^\top \quad (5.184)$$

where $\mathbf{U} = \{u_{ij}\}$ is a unit upper triangular matrix (i.e., the diagonal consists of ones solely) and $\mathbf{D} = \text{diag}\{\mathbf{d}\}$, $\mathbf{d} = [d_1, d_2, \dots, d_m]$, is a diagonal matrix. A number of alternative factorizations is possible, see e.g., [Bierman 75], [Ljung 87, Ch. 11.7]. When $d_i > 0$, $\forall i \in [1; m]$ then obviously \mathbf{P} is positive definite. That is, if \mathbf{U} and \mathbf{D} are updated separately then \mathbf{P} can be ensured to be positive definite. The algorithm is shown to be numerically stable when the forgetting factor $\lambda = 1$ cf. [Bierman 75]. In [Bierman 75] the algorithm is derived in the case $\lambda = 1$ only. However, it is straight forward to incorporate the forgetting factor, $\lambda(k)$. This is due to the fact that the update $\mathbf{P}(k)$ depends on $\mathbf{P}(k-1)/\lambda(k)$ only, cf. pa:pupdat. Suppose that $\mathbf{P}(k-1)/\lambda(k)$ is factorized as:

$$\frac{\mathbf{P}(k-1)}{\lambda(k)} = \mathbf{U}\mathbf{D}\mathbf{U}^\top, \quad (5.185)$$

and $\mathbf{P}(k)$ as:

$$\mathbf{P}(k) = \widehat{\mathbf{U}}\widehat{\mathbf{D}}\widehat{\mathbf{U}}^\top. \quad (5.186)$$

Furthermore define the following quantities:

$$\alpha = 1 + \psi^\top(k) \frac{\mathbf{P}(k-1)}{\lambda(k)} \psi(k), \quad (5.187)$$

$$\mathbf{f} = [f_1, f_2, \dots, f_m] = \mathbf{U}^\top \psi(k), \quad (5.188)$$

$$\mathbf{v} = [v_1, v_2, \dots, v_m] = \mathbf{D}\mathbf{f} = \mathbf{D}\mathbf{U}^\top \psi(k). \quad (5.189)$$

Substituting the factorizations pa:fac1, (5.186) into pa:pupdat and using the above quantities gives:

$$\widehat{\mathbf{U}}\widehat{\mathbf{D}}\widehat{\mathbf{U}}^\top = \left[\mathbf{U}\mathbf{D}\mathbf{U}^\top - \frac{1}{\alpha} \left(\mathbf{U}\mathbf{D}\mathbf{U}^\top \boldsymbol{\psi}(k) \boldsymbol{\psi}^\top(k) \mathbf{U}\mathbf{D}\mathbf{U}^\top \right) \right] \quad (5.190)$$

$$= \mathbf{U} \left[\mathbf{D} - \frac{1}{\alpha} \left(\mathbf{D}\mathbf{U}^\top \boldsymbol{\psi}(k) \left(\mathbf{D}\mathbf{U}^\top \boldsymbol{\psi}(k) \right)^\top \right) \right] \mathbf{U}^\top \quad (5.191)$$

$$= \mathbf{U} \left[\mathbf{D} - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha} \right] \mathbf{U}^\top. \quad (5.192)$$

The U-D factorization of the term in the square brackets can be done explicitly cf. [Bierman 75]. Suppose that the factorization is given by:

$$\widetilde{\mathbf{U}}\widetilde{\mathbf{D}}\widetilde{\mathbf{U}}^\top = \mathbf{D} - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha}. \quad (5.193)$$

Now pa:ud1 yields:

$$\widehat{\mathbf{U}}\widehat{\mathbf{D}}\widehat{\mathbf{U}}^\top = \mathbf{U}\widetilde{\mathbf{U}}\widetilde{\mathbf{D}}\widetilde{\mathbf{U}}^\top \mathbf{U}^\top = \mathbf{U}\widetilde{\mathbf{U}}\widetilde{\mathbf{D}} \left(\mathbf{U}\widetilde{\mathbf{U}} \right)^\top. \quad (5.194)$$

Since both \mathbf{U} and $\widetilde{\mathbf{U}}$ are upper triangular matrices then the product $\mathbf{U}\widetilde{\mathbf{U}}$ is an upper triangular matrix. Consequently, the following identities result:

$$\widehat{\mathbf{D}} = \widetilde{\mathbf{D}}, \quad (5.195)$$

$$\widehat{\mathbf{U}} = \mathbf{U}\widetilde{\mathbf{U}}. \quad (5.196)$$

At the succeeding time step, $k + 1$, \mathbf{U} and \mathbf{D} is the U-D factors of $\mathbf{P}(k)/\lambda(k + 1)$. This is obtained by letting: $\mathbf{D} = \widehat{\mathbf{D}}/\lambda(k + 1)$ and $\mathbf{U} = \widehat{\mathbf{U}}$.

This concludes the updating of the U-D factors and finally the RGNB-algorithm cf. [Bierman 75] becomes⁴⁰:

RGNB-AlgorithmInitialization:

- Step 1*
- a.* Choose an initial weight vector \mathbf{w} , cf. Sec. 5.3.2.
 - b.* Select the regularization parameter, κ .
 - c.* Specify the number of iterations, *itr*, i.e., the number of times the training set is replicated during training.
 - d.* Choose a scheme – e.g., pa:lamreki – for updating the instantaneous forgetting factor, $\lambda(\ell)$, $\ell \in [1; N \text{ itr}]$.
 - e.* Initialize the U-D factors, i.e., $\mathbf{U} = \mathbf{I}$, and $\mathbf{D} = \gamma^{-1} \mathbf{I}$. γ is cf. pa:kappaeqn given by:

$$\gamma = \frac{\kappa}{\mu(N \text{ itr})\beta(N \text{ itr}, 0)}. \quad (5.197)$$

- f.* If μ is updated cf. the line denoted † in *Step 2*. then initialize: $\mu^{-1}(0) = 1$.
- g.* Initialize the counter, $\ell = 0$.

⁴⁰The iteration variable ℓ used elsewhere is omitted for simplicity.

```

Step 2   for  $i = 0, 1, \dots, itr - 1$ 
          for  $k = 1, 2, \dots, N$ 
             $\ell \leftarrow \ell + 1$ 
             $\lambda^{-1} = \frac{1}{\lambda(\ell)}$                                 10
             $\mu^{-1} \leftarrow \lambda(\ell)\mu^{-1} + 1$                 1           †
             $\hat{y} = f(\mathbf{x}(k); \mathbf{w}(\ell))$                          $CP_{\hat{y}}$ 
             $e = y(k) - \hat{y}$ 
            Compute  $\boldsymbol{\psi}$                                         $CP_{\boldsymbol{\psi}}$ 
             $\mathbf{f} = \mathbf{U}\boldsymbol{\psi}$                                 $\frac{m^2 - m}{2}$ 
             $\mathbf{v} = \mathbf{D}\mathbf{f}$                                        m
             $\alpha = 1 + v_1 f_1$                                        1
             $\gamma = \frac{1}{\alpha}$                                        10
             $d_1 \leftarrow \lambda^{-1} \gamma d_1$                         2
             $b_1 = v_1$ 
            for  $j = 2, 3, \dots, m$ 
               $\beta = \alpha$ 
               $\alpha \leftarrow \alpha + f_j v_j$                             1
               $p_j = -f_j \gamma$                                           1
               $\gamma = \frac{1}{\alpha}$                                        10
               $d_j \leftarrow \lambda^{-1} \beta \gamma d_j$                     3
               $b_j = v_j$ 
              for  $i = 1, 2, \dots, j - 1$ 
                 $\beta = u_{i,j}$ 
                 $u_{i,j} = \beta + b_i p_j$                                     1
                 $b_i \leftarrow b_i + \beta v_j$                             1
              end
            end
             $e \leftarrow \frac{e}{\alpha}$                                        10
             $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{b}e$                                m
          end
        end
      end

```

Notes:

- The computation of the instantaneous gradient $\boldsymbol{\psi}(\ell) \equiv \boldsymbol{\psi}(\ell; \mathbf{w}(\ell))$ is in layered ar-

chitectures performed by back-propagation cf. Sec. 5.3.3 and Sec. 5.3.4.

- \mathbf{b} , \mathbf{p} are some auxiliary m -dimensional vectors and β , γ are auxiliary scalars.
- As in the RGN-algorithm the updating of μ^{-1} is omitted unless the inverse pseudo Hessian is required subsequently. In the latter case then with $\ell = N \text{ itr}$ the estimated weights yield $\hat{\mathbf{w}} = \mathbf{w}$ and the inverse pseudo Hessian is cf. pa:iphdef, (5.185) we set⁴¹:

$$\widetilde{\mathbf{H}}_N^{-1}(\hat{\mathbf{w}}) = \mu^{-1} \mathbf{P} = \mu^{-1} \lambda(\ell) \mathbf{U} \mathbf{D} \mathbf{U}^\top. \quad (5.198)$$

- In right column the computational complexity of the actual line is shown. Notice:
 - A division is assumed to be equal to 10 multiplications.
 - The \mathbf{U} matrix contains $(m^2 - m)/2$ elements which are different from zero and one.
 - The \mathbf{D} matrix contains m non-zero elements.
 - The complexity of the i -loop is equal to $2(j - 1)$ and since $j \in [2; m]$ then the total complexity is found by multiplying with $2 \sum_{j=2}^m (j - 1) = m^2 - m$.
 - The total complexity of the lines which are in the j -loop only is found by multiplying with $15(m - 1)$.
- Neglecting the complexity of the line indicated by †, and adding up the individual complexity components yields:

$$CP_{\text{RGNB}} = N \text{ itr} \left(CP_{\hat{\mathbf{y}}} + CP_{\psi} + \frac{3}{2}m^2 + \frac{31}{2}m + 18 \right). \quad (5.199)$$

Considering a 2-layer MFPNN using the approximate result cf. pa:ycom2, and pa:cpsi2l:

$$CP_{\text{RGNB}} = N \text{ itr} \left(\frac{3}{2}m^2 + m \frac{35p + 110}{2p + 4} + 18 \right). \quad (5.200)$$

Another advantage of the RGNB-algorithm appears by comparing pa:rgnbcom and pa:rgncom. The result is⁴²:

$$CP_{\text{RGNB}} < CP_{\text{RGN}} \text{ as } m \geq 23. \quad (5.201)$$

⁴¹In order to avoid a multiplication with $\lambda(\ell)$ in pa:iphrgnb the multiplications with λ^{-1} in the updating for d_j , $j \in [1; m]$ should be omitted in the last iteration.

⁴²This is seen by solving the inequality

$$\begin{aligned} \frac{3}{2}m^2 + \frac{31}{2}m + 18 &< 2m^2 + 4m + 20 \\ \Downarrow & \\ \frac{1}{2}m^2 - \frac{23}{2}m + 2 &> 0 \end{aligned}$$

which has the solution $m \geq \left\lceil \frac{23 + \sqrt{513}}{2} \right\rceil = 23$.

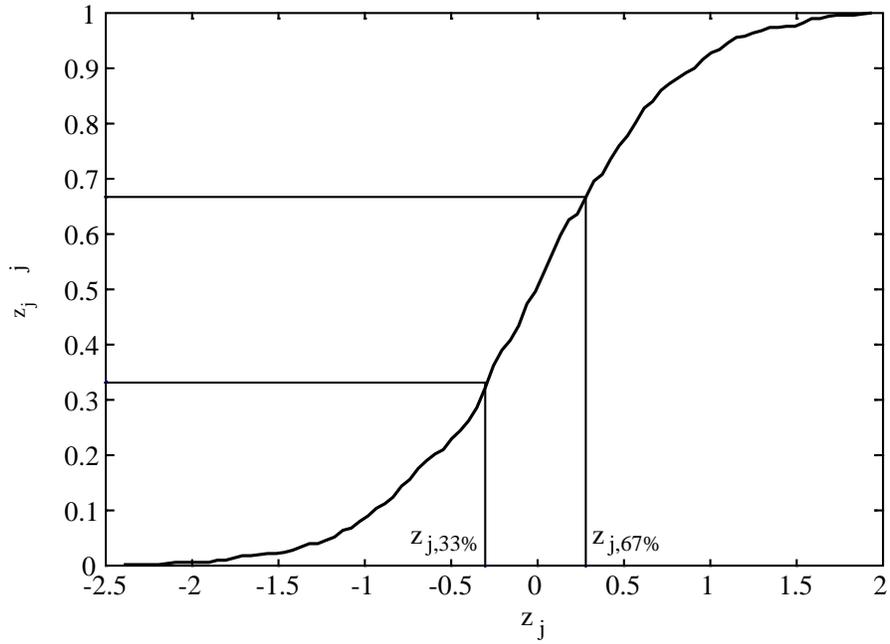
5.8 Summary

The subject of this chapter was algorithms for estimating the filter weights - or parameters. The weights are estimated by minimizing some cost function on the basis of collected input-output data, called the training set. The choice of cost function and various aspects of the minimization task were discussed. In particular, we focused on the Least Squares (LS) cost function with a weight decay regularizer.

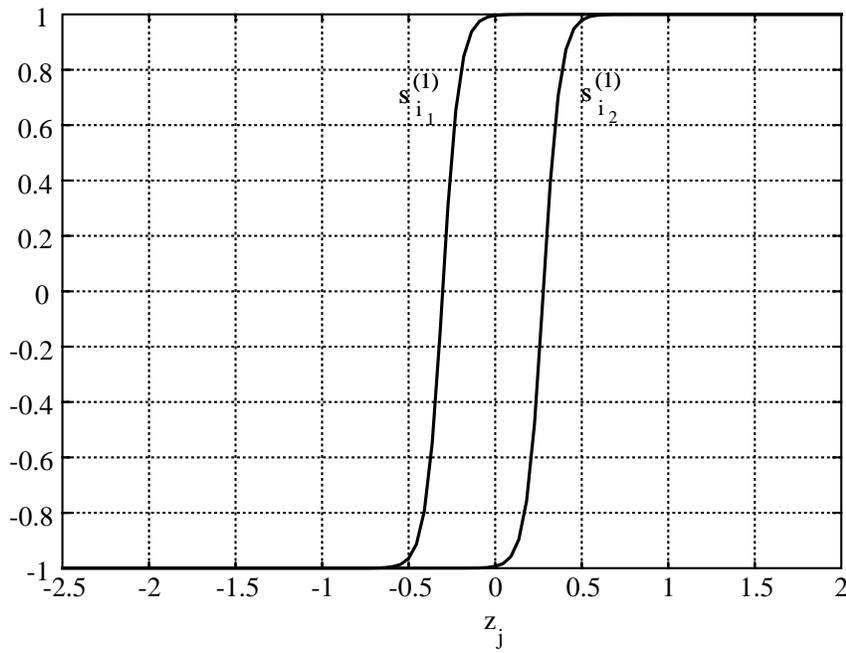
The minimization task can be solved using either off-line or recursive algorithms. An off-line algorithm iteratively updates the weight estimates based on all data in the training set, while a recursive algorithm uses the most recent data sample only. Recursive algorithms are therefore suitable for on-line tracking of time varying systems; however, they may also be used in an off-line mode. The algorithms fall into two classes: Gradient Descent (first order algorithm) and Newton-Raphson (second order algorithm) based algorithms. Algorithm performance was discussed in terms of convergence properties and computational complexity. Generally, first and second order algorithms provide a trade off between convergence and computational complexity. Second order algorithms converge much faster at the expense of increased computational complexity. In Ch. 8 we present some comparative studies which indicate the potential of second order algorithms.

We focused mainly on layered feed-forward filter architectures, including the multi-layer feed-forward perceptron neural network (MFPNN), even though the algorithms are general purpose. The chosen architecture enters the algorithms through the determination of the filter output, $\hat{y}(k)$, and the instantaneous gradient vector, $\boldsymbol{\psi}(k; \mathbf{w}) = \partial \hat{y}(k) / \partial \mathbf{w}$, i.e., the derivative of the filter output w.r.t. the weights. The crucial matter is the determination of $\boldsymbol{\psi}$. We demonstrated how the back-propagation algorithm can be generalized to deal with general layered filter architectures w.r.t. determination of $\boldsymbol{\psi}$. Moreover, a compact matrix formulation of the back-propagation within MFPNN was provided.

In addition, we suggested a procedure for proper initialization of the weights in an MFPNN in order to speed up the algorithm convergence.



(a)



(b)

Figure 5.3: Example of the location of two neurons, i.e., $n = 2$, i_1 , i_2 according to the probability distribution function, $P_{z_j}(z_j)$, of z_j . The lower plot shows the responses of the neurons.

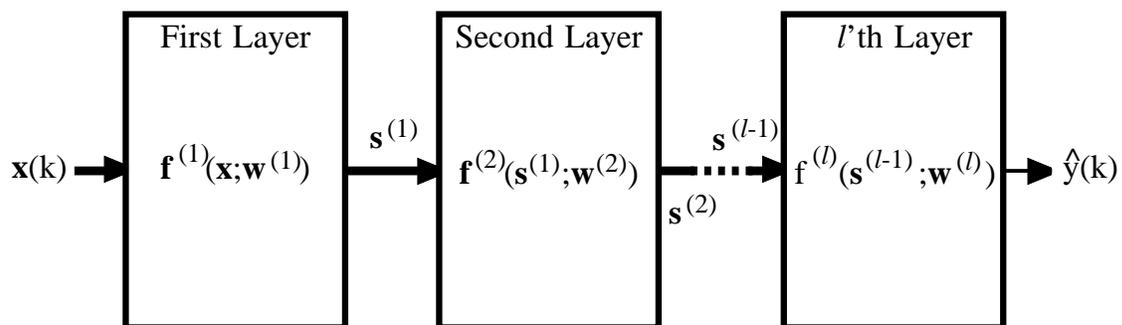


Figure 5.4: A layered filter architecture. $\tilde{\mathbf{s}}^{(r)}$, $\mathbf{w}^{(r)}$ are the output and the weight vector of the r 'th layer, respectively. $\mathbf{f}^{(r)}(\cdot)$ is the vector function which specifies the nonlinear mapping of the r 'th layer.

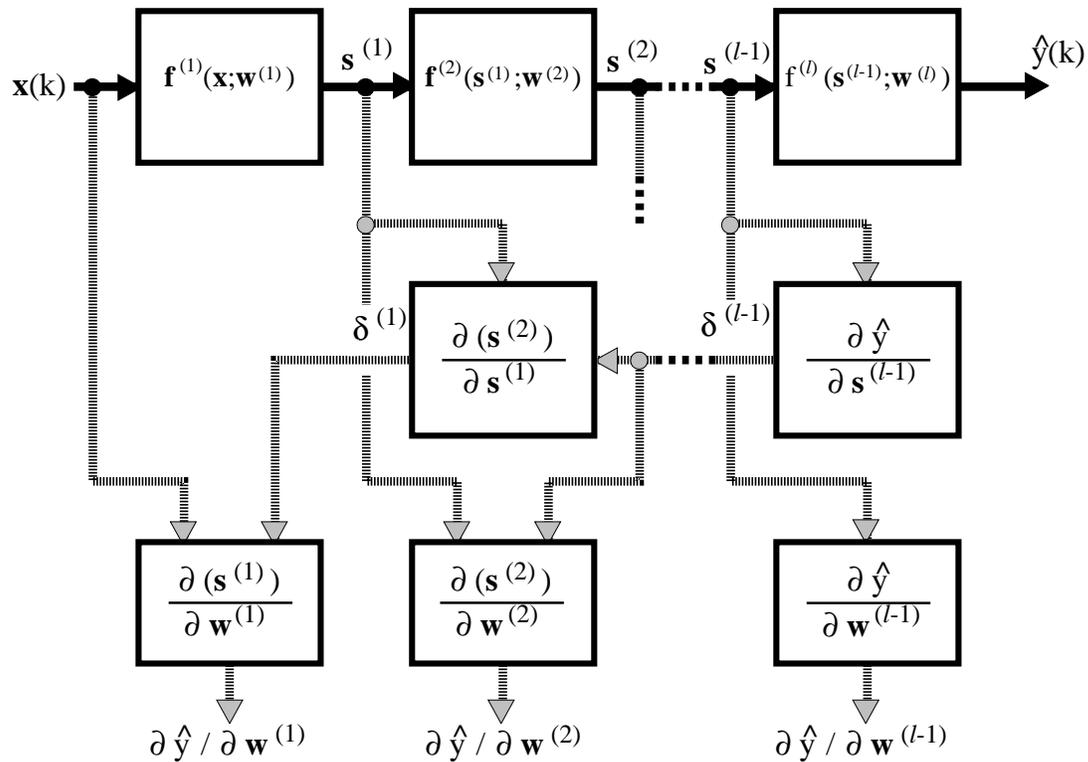


Figure 5.5: The general back-propagation algorithm for calculating gradient vectors.

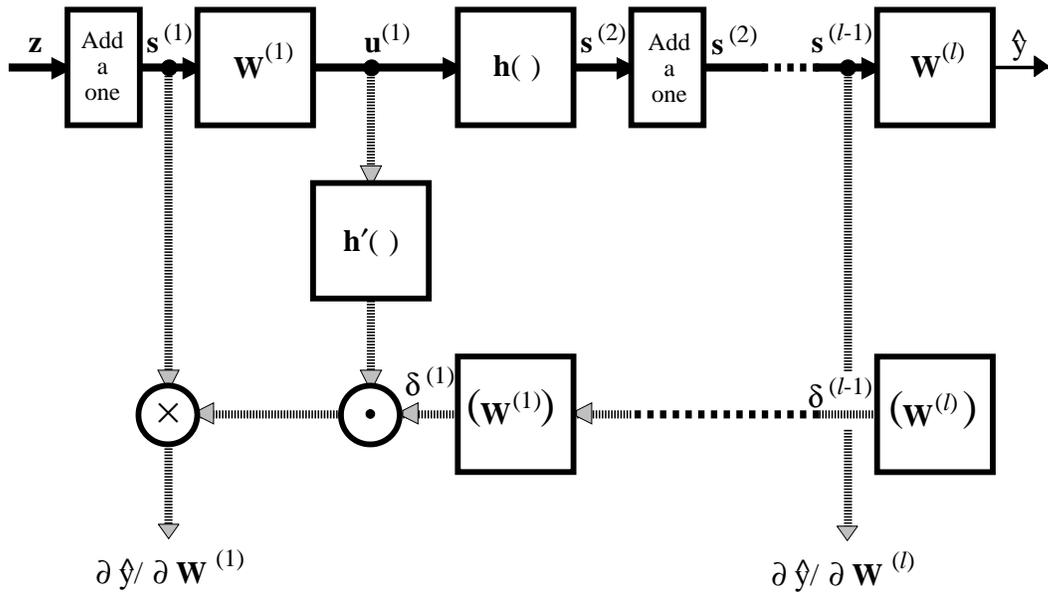


Figure 5.6: The BP algorithm dealing with the MFPNN architecture. \otimes and \odot denote matrix and element by element multiplication, respectively.

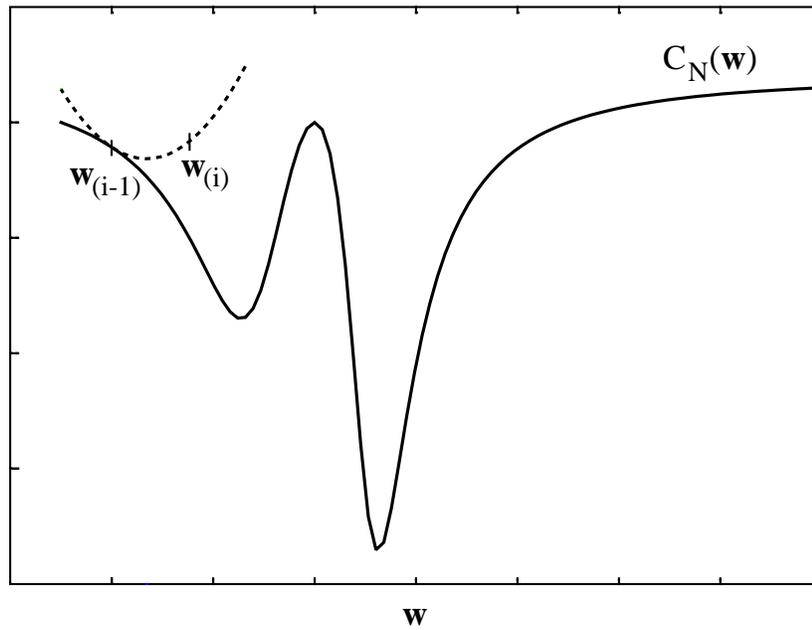
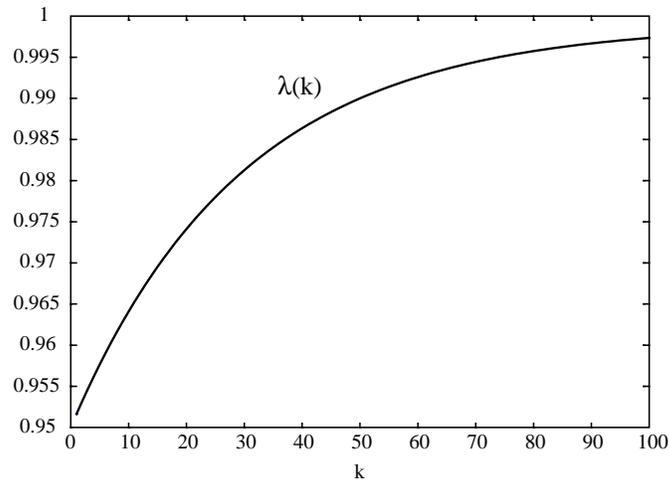
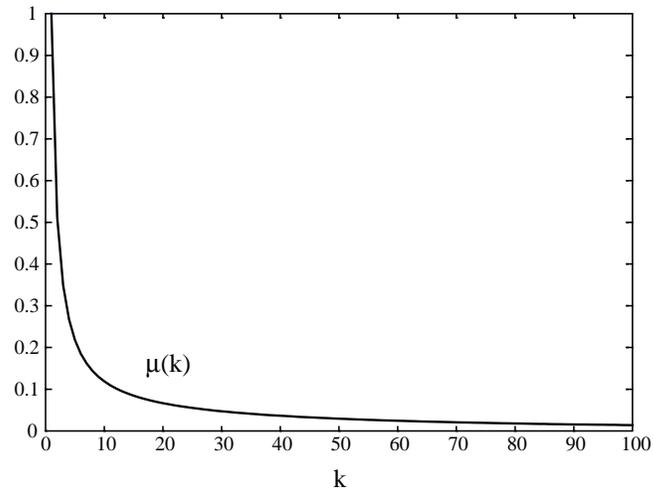


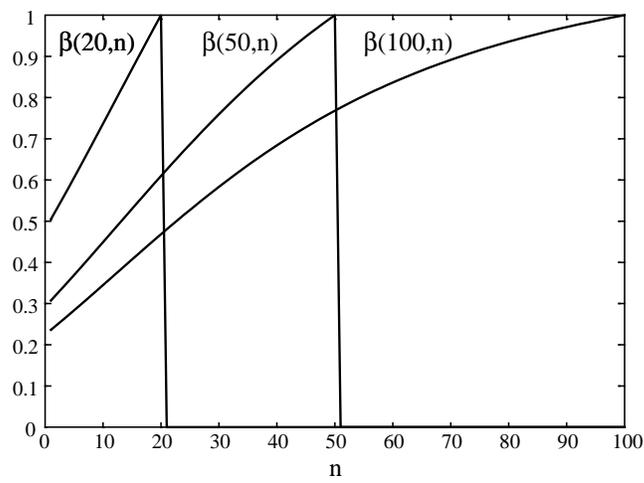
Figure 5.7: An example of a modified Gauss-Newton iteration with $\mu > 1$.



(a)



(b)



(c)

Figure 5.8: Example of $\lambda(k)$, $\mu(k)$, $\beta(k, n)$.

CHAPTER 6

FILTER ARCHITECTURE SYNTHESIS

The subject of this chapter is the discussion of filter architectures.

First we briefly discuss the possibility of using a priori knowledge in order to determine a proper filter architecture.

Next the concept of generalization ability and generalization error is introduced. From a theoretical point of view the ultimate goal is to determine the architecture which minimizes the generalization error; however practical considerations such as computational complexity, numerical precision, hardware platforms etc. may lead to a filter architecture which is not necessarily optimal w.r.t. generalization error. We introduce the *model error decomposition* which enables the discussion and interpretation of various factors which influence the generalization error. The aim is to clarify possibilities and limitations in finding algorithms which automatically synthesize the filter architecture.

The rest of the chapter is devoted to practical filter architecture synthesis algorithms, called ASA's. First a basic algorithm for architecture synthesis based on generalization error estimates is introduced. A number of commonly used generalization error estimates are mentioned and their limitations are discussed. Furthermore, we present a novel generalization error estimate which is developed with the aim of handling NN-models which are incomplete, i.e., not able to model data perfectly. Next we focus on algorithms for expansion and reduction of the filter architecture aiming at improving the generalization ability. The ideal ASA gradually expands and reduces the architecture in order to accomplish changing environments and to ensure that the ASA does not get stuck in a suboptimal architecture. In particular, algorithms which optimize the filter architecture by eliminating superfluous weights are presented.

Finally we demonstrate that regularization can be viewed as a tool for partial optimization of the filter architecture since it has an influence on the generalization error. It is shown that the use of regularization may reduce the generalization and thereby improve the filter architecture.

6.1 System and Model

Let us recapitulate the definitions of the system and the model. Suppose that the training set, $\mathcal{T} = \{\mathbf{x}(k); y(k)\}$, $k = 1, 2, \dots, N$ of connected input-output data is generated by the

nonlinear system:

$$y(k) = g(\mathbf{x}(k)) + \varepsilon(k) \quad (6.1)$$

where $\mathbf{x}(k)$ denotes the (L -dimensional, stochastic) input vector signal, $y(k)$ is the scalar output signal, $g(\cdot)$ constitutes a time-invariant mapping and $\varepsilon(k)$ is the inherent stochastic noise source with zero mean and finite variance, σ_ε^2 . Furthermore, suppose that the model of the nonlinear system obeys:

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \quad (6.2)$$

where $e(k; \mathbf{w})$ is the error signal.

6.2 A Priori Knowledge

As mentioned in Ch. 1 a priori knowledge may be significant when guiding the choice of a proper filter architecture. Two extreme situations exist:

1. The structure of the system which generated the data is known in detail from physical experimental analysis. The only missing information is the specific values of some characteristic system parameters under the actual circumstances.
2. No information of the data generating system is available at all. That is, we are obliged to use “black box” modeling.

In practice none of the situations occur. If much a priori knowledge is available a highly specialized model structure will emerge. It is hard to believe that the neural network architectures presented in Ch. 3 will resemble this specialized structure. Consequently, it should be emphasized that neural network architectures have the greatest potential in cases where sparse a priori knowledge is available or too expensive to collect. On the other hand, neglecting structural knowledge, thus ending with black box modeling, may result in poor performance. Efforts on incorporating the knowledge into the architecture is consequently important; however, in the general case it is troublesome to provide any guidelines.

A source of structural knowledge may stem from estimating some relevant features from the present data, i.e., the training set. In Ch. 4 during the discussion of preprocessing methods simple algorithms for determining architecture parameters such as window length, L , and delay, D , is suggested. The choice of preprocessing method may be regarded as structural knowledge thus reducing the task of the succeeding black box filter. The literature also provides methods which decide whether the system is basically linear or nonlinear from estimating various features, see e.g., [Priestley 88, Sec. 3.4], [Billings & Voon 83]. However, the computational complexity and assumptions involved in these approaches give rise to the question whether it is simpler just fitting an LL- and an XN-model. If the XN-model performs better than the LL-model then this indicates that the system indeed is nonlinear of nature.

In [Ljung 87, Ch. 16] further elaborations on a priori considerations are treated.

6.3 Generalization Ability

In the context of searching for an optimal filter architecture a model quality, or performance measure, is required. The quality of an estimated model is judged as good if the

model has the ability to generalize, that is the model is capable of predicting the system properly using data *which are not present* in the training set. Formally we make the following definition which also is suggested elsewhere, e.g., [Krogh & Hertz 91], [Ljung 87], and [Moody 91]:

DEFINITION 6.1 *The generalization error¹, G , is defined as the expected, squared error on a test sample, $\{\mathbf{x}_t; y_t\}$, which is **independent** of the training set but originates from the same distribution.*

$$\begin{aligned} G(\mathbf{w}) &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ [y_t - f(\mathbf{x}_t; \mathbf{w})]^2 \right\} \\ &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ e_t^2(\mathbf{w}) \right\}. \end{aligned} \quad (6.3)$$

$E_{\mathbf{x}_t, \varepsilon_t} \{ \cdot \}$ denotes expectation with respect to the joint p.d.f. of $[\mathbf{x}_t, \varepsilon_t]$. Note that G depends both on the model, $f(\cdot)$, and the weights, \mathbf{w} . Clearly the model has a better generalization ability the smaller the generalization error is. Note that according to the definition of the expected cost function, $C(\mathbf{w})$ pa:expcst, $G(\mathbf{w}) = C(\mathbf{w})$ if no regularization is employed.

6.3.1 Alternative Definitions of Generalization Ability

The above definition of G is a common choice and closely connected with the employed cost function. However, other types of cost functions naturally lead to a redefinition of the generalization ability.

If the distribution of the output $y(k)$ is known the weights may be estimated by a maximum likelihood (ML) procedure. That is, let $\hat{\mathbf{w}}$ be a particular ML-estimator then

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w}) \quad (6.4)$$

where the cost function C_N is defined by the negative log-likelihood function

$$C_N(\mathbf{w}) = -\frac{1}{N} \ln L_N(\mathbf{w}) \quad (6.5)$$

where $\ln[\cdot]$ is the natural logarithm and $L_N(\cdot)$ is the likelihood function

$$L_N(\mathbf{w}) = p(y(1), \dots, y(N) | \mathbf{x}(1), \dots, \mathbf{x}(N); \mathbf{w}) \quad (6.6)$$

Here $p(\cdot | \cdot)$ denotes the joint conditional p.d.f. of all $y(k)$ in the training set conditioned on the inputs and the parameters.

In this context a natural measure of the generalization error, say G , is the expected cost function, i.e., the expected negative log-likelihood on an independent sample $\{\mathbf{x}_t; y_t\}$, i.e.,

$$G(\mathbf{w}) = E_{\mathbf{x}_t, \varepsilon_t} \{ C_N(\mathbf{w}) \} = E_{\mathbf{x}_t, \varepsilon_t} \{ -\ln[p(y_t | \mathbf{x}_t; \mathbf{w})] \} \quad (6.7)$$

Note, that this definition is in keeping with the philosophy which leads to the *AIC* generalization error estimator [Akaike 74] (see Sec. 6.5).

¹The term “generalization error” may seem misleading since it in fact is the expectation of the *squared* error signal. However, the nomenclature is in keeping with literature on this subject and furthermore, the attached substantive “generalization” should prevent any misunderstanding.

EXAMPLE 6.1

Assume that $e \in \mathcal{N}(0, \sigma_e^2)$ with unknown variance σ_e^2 , i.e., the variance acts like an extra parameter. According to sa:model

$$p(y_t | \mathbf{x}_t; \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma_e} \exp\left(-\frac{1}{2\sigma_e^2} e_t^2(\mathbf{w})\right) \quad (6.8)$$

↓

$$-\ln p(y_t | \mathbf{x}_t; \mathbf{w}) = \ln(\sqrt{2\pi}\sigma_e) + \frac{e_t^2(\mathbf{w})}{2\sigma_e^2}. \quad (6.9)$$

The generalization error is then according to sa:explike

$$G(\mathbf{w}; \sigma_e^2) = \ln(\sqrt{2\pi}\sigma_e) + \frac{1}{2\sigma_e^2} E_{\mathbf{x}_t, \varepsilon_t} \{e_t^2(\mathbf{w})\}. \quad (6.10)$$

Note the dependence on the parameter σ_e^2 is explicitly emphasized. □

Consider another case where the input $\mathbf{x}(k)$ – in contrast to being stochastic – is a deterministic periodic sequence with period T . A straight forward definition is thus to alter the expectation in sa:gentrue w.r.t. to \mathbf{x} with a time-averaging, i.e.,

$$G(\mathbf{w}) = \frac{1}{T} \sum_{k=1}^T E_{\varepsilon(k)} \{e^2(k; \mathbf{w})\} \quad (6.11)$$

where $\varepsilon(k)$ is given implicit through the rewriting:

$$e(k; \mathbf{w}) = g(\mathbf{x}(k)) - f(\mathbf{x}(k); \mathbf{w}) + \varepsilon(k). \quad (6.12)$$

6.3.2 Average Generalization Error

The generalization error cf. sa:gentrue is defined for all possible weight vectors, \mathbf{w} ; however usually the generalization error when employing the estimated weights, i.e., $G(\hat{\mathbf{w}})$, is the quantity of interest since this measures the quality of the estimated model. Now, the estimated weights, $\hat{\mathbf{w}}$, depend on the actual training set, \mathcal{T} ; that is, if another training set of equal size were employed other weight estimates would result, and consequently, another generalization error emerges. In fact, the training set is just a random set of related, succeeding² input-output data drawn from an infinite reservoir. Consequently, the generalization error obtained with the present training set may be rather atypical. That is, the reason that we believe the model is pretty good or bad could be due to the actual training data rather than the chosen architecture. If high performance is obtained with the chosen architecture this is of course no problem at all. However, if low performance is obtained one might think that the architecture is inadequate, whereas the real reason is an atypical training set. Consider e.g., the following simple example: Suppose that the data are generated by the chosen filter architecture but only a small training set is available. A few atypical training samples³ then has a high influence on the weight estimate $\hat{\mathbf{w}}$ and may thereby result in poor performance.

In order to illuminate these effects we make the following definition:

²This fact is indeed a consequence of dealing with time series. For instance, within classification problems there may be no dependence among the individual training samples.

³For instance, samples where $|e(k)|$ is large.

DEFINITION 6.2 *The average generalization error, Γ , is defined by:*

$$\Gamma = E_{\mathcal{T}} \{G(\hat{\mathbf{w}})\} \quad (6.13)$$

where the expectation is done w.r.t. the training set⁴.

Notice two facts:

- Expectation w.r.t. to the training set, i.e., the samples $\{\mathbf{x}(k); y(k)\}$, $k = 1, 2, \dots, N$, and expectation w.r.t. to the samples $\{\mathbf{x}(k); \varepsilon(k)\}$ are identical according to sa:nonsys.
- $G(\hat{\mathbf{w}})$ depends on the training set via $\hat{\mathbf{w}}$.

6.3.3 Decomposition of the Generalization Error

The average generalization error, Γ , can be decomposed several ways in order to study the nature of various components which influence the generalization ability.

Consider the generalization error employing the estimated weights, $\hat{\mathbf{w}}$, which cf. sa:gentrue is

$$G(\hat{\mathbf{w}}) = E_{\mathbf{x}_t, \varepsilon_t} \left\{ e_t^2(\hat{\mathbf{w}}) \right\} \quad (6.14)$$

where the expectation is taken w.r.t. the test sample, $[\mathbf{x}_t, \varepsilon_t]$, which is independent of the training data – and with that, $\hat{\mathbf{w}}$. According to the system sa:nonsys and the model sa:model elimination of $y(k)$ yields

$$\begin{aligned} G(\hat{\mathbf{w}}) &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ [\varepsilon_t + g(\mathbf{x}_t) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \\ &= \underbrace{E_{\varepsilon} \left\{ \varepsilon_t^2 \right\}}_{\text{Noise Variance}} + \underbrace{E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\}}_{\text{Mean Square Adjustment Error}}. \end{aligned} \quad (6.15)$$

Above we assume that the inherent noise $\varepsilon(k)$ is independent of the input vector $\mathbf{x}(k)$. The first term in this decomposition is the variance⁵, σ_{ε}^2 , of the inherent noise which does not depend on the actual model. The second term is the mean square adjustment error which depends on the employed model, the algorithm for estimating the weights and the training set. Consequently, the issue of designing a proper filter consists in reducing the mean square adjustment error as much as possible.

Next we consider further decompositions of the average generalization error, Γ , which cf. sa:gamdef is given by $\Gamma = E_{\mathcal{T}} \{G(\hat{\mathbf{w}})\}$ where \mathcal{T} denotes the training set. Only the terms which depend on the estimated weights, $\hat{\mathbf{w}}$, are affected by the expectation w.r.t. the training set. Furthermore, notice that $\hat{\mathbf{w}}$ does not depend on the independent test sample.

⁴Expansion of the expectation reads:

$$\Gamma = \int \dots \int G(\hat{\mathbf{w}}(\{\mathbf{x}(k); y(k)\})) \cdot p(\mathbf{x}(1); y(1), \mathbf{x}(2); y(2), \dots, \mathbf{x}(N); y(N)) d\mathbf{x}(1)dy(1)d\mathbf{x}(2)dy(2) \dots d\mathbf{x}(N)dy(N)$$

⁵Recall that the mean value of the inherent noise equals zero per definition.

6.3.4 Model Error Decomposition

Recall that \mathbf{w}^* are the weights which minimize the expected cost function, $C(\mathbf{w})$, and that $\hat{\mathbf{w}} \rightarrow \mathbf{w}^*$ as $N \rightarrow \infty$ in general (see Th. 5.1). Hence, the best attainable filter within the actual filter architecture is: $\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w}^*)$ corresponding to the case where an infinite amount of training data were present. A possible decomposition is then done in terms of $f(\mathbf{x}(k); \mathbf{w}^*)$. According to sa:decom1 we get:

$$\begin{aligned}
\Gamma &= \sigma_\varepsilon^2 + E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \\
&= \sigma_\varepsilon^2 + E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \mathbf{w}^*) + f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \\
&= \sigma_\varepsilon^2 + \\
&\quad E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \mathbf{w}^*)]^2 \right\} \right\} + \\
&\quad E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} + \\
&\quad 2E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \mathbf{w}^*)] \cdot [f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})] \right\} \right\} \\
&= \underbrace{\sigma_\varepsilon^2}_{\text{Noise Variance}} + \underbrace{E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \mathbf{w}^*)]^2 \right\}}_{\text{Mean Square Model Error (MSME}(\mathbf{w}^*))} \\
&\quad + E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \quad \left. \vphantom{E_{\mathcal{T}}} \right\} \text{Weight Fluctuation} \\
&\quad + 2E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \mathbf{w}^*)] \cdot [f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})] \right\} \right\} \quad \left. \vphantom{E_{\mathcal{T}}} \right\} \text{Penalty (WFP)}
\end{aligned} \tag{6.16}$$

Γ is thus composed of three terms:

- The inherent noise variance which is the lower bound on Γ as well as $G(\hat{\mathbf{w}})$.
- The *mean square model error (MSME)* which convey the minimal increase in the (average) generalization error when applying the filter architecture $f(\cdot)$ in order to model $g(\cdot)$. Using sa:decom1 evaluated at the weights, \mathbf{w} , we get:

$$\begin{aligned}
G(\mathbf{w}) &= \sigma_\varepsilon^2 + E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}) - f(\mathbf{x}; \mathbf{w})]^2 \right\} \\
&= \sigma_\varepsilon^2 + MSME(\mathbf{w}).
\end{aligned} \tag{6.17}$$

Hence the optimal weights, \mathbf{w}^* – which minimize $G(\mathbf{w})$ – also minimize $MSME(\mathbf{w})$. That is by applying the LS cost function $S_N(\mathbf{w})$ the $MSME$ is minimal (provided that \mathbf{w}^* is the global optimum) since the expected cost function, in this case, equals $G(\mathbf{w})$ ⁶.

Now, consider the concept of complete models:

DEFINITION 6.3 *If there exist parameters, \mathbf{w}° , such that $\forall \mathbf{x}(k) : g(\mathbf{x}(k)) \equiv f(\mathbf{x}(k); \mathbf{w}^\circ)$ we signify the model as complete; otherwise, as incomplete. \mathbf{w}° is denoted the true parameters⁷.*

⁶This is of course a direct consequence of the definition of the generalization error.

⁷Note that the true parameters (weights) are not necessarily unique. However, this is not crucial to the arguments in the following.

When the model is complete then $MSME(\mathbf{w}^*)$ becomes zero provided that the LS cost function is employed (i.e., no regularization) and that \mathbf{w}^* is the global minimum. This is easily seen by the following arguments: In the case of no regularization then the expected cost function $C(\mathbf{w}) = G(\mathbf{w})$. Now $\mathbf{w}^* = \arg \min_{\mathbf{w}} G(\mathbf{w})$ and suppose that \mathbf{w}^* is the global optimum then cf. sa:gw

$$MSME(\mathbf{w}) = E_{\mathbf{x}_t} \left\{ [f(\mathbf{x}; \mathbf{w}^\circ) - f(\mathbf{x}; \mathbf{w})]^2 \right\}. \quad (6.18)$$

This equation shows that the global optimum is reached for $\mathbf{w} = \mathbf{w}^* = \mathbf{w}^\circ$ and thereby eliminating the $MSME$.

In general; however, the $MSME$ is non-zero due to the following circumstances:

- The model is incomplete. Usually the lack of knowledge concerning the structure of $g(\cdot)$ precludes the possibility of suggesting a complete model (with finite order m). Consequently, we claim that incomplete models are the common case.
- A regularization term is included in the cost function. In this case the optimal weights, say \mathbf{w}_R^* , are given by $\mathbf{w}_R^* = \arg \min_{\mathbf{w}} C(\mathbf{w})$ and $\mathbf{w}_R^* \neq \mathbf{w}^*$, where $\mathbf{w}^* = \arg \min G(\mathbf{w}) = \arg \min MSME(\mathbf{w})$. Hence the $MSME$ is increased.

Moreover, it is impossible to calculate the $MSME$ without knowledge of the system, $g(\mathbf{x})$, and the p.d.f. of the input. In practice the model error then acts as an extra noise source which is input dependent in contrast to the inherent noise which usually is assumed independent of the input.

- The last term is the *weight fluctuation penalty* (WFP) which constitutes the contribution to Γ due to a finite training set, i.e., the fluctuations in $\hat{\mathbf{w}}$ around \mathbf{w}^* when applying different training sets. Evaluating sa:decom1 at the optimal weights, \mathbf{w}^* we notice that $G(\mathbf{w}^*)$ equals the sum of the noise variance and the $MSME$. Consequently the WFP is also expressed as: $E_{\mathcal{T}}\{G(\hat{\mathbf{w}}) - G(\mathbf{w}^*)\}$. Now, if \mathbf{w}^* defines the global optimum then $G(\mathbf{w}) > G(\mathbf{w}^*)$, $\forall \mathbf{w} \neq \mathbf{w}^*$. In particular, $G(\hat{\mathbf{w}}) - G(\mathbf{w}^*)$ is positive and then the average w.r.t. different training sets (i.e., different estimates $\hat{\mathbf{w}}$) is positive, q.e., the WFP is positive⁸.

Since $\hat{\mathbf{w}} \rightarrow \mathbf{w}^*$ as $N \rightarrow \infty$ (cf. Th. 5.1) it is obvious that the WFP decreases when N increases. However, only when making some simplifying assumptions it is possible to state how WFP scales with N . The assumptions⁹ made in Sec. 6.5 and Sec. 6.6 lead to: $WFP = o(N^{-1})$, $N \rightarrow \infty$ for a fixed filter architecture.

In order to evaluate the WFP the first and second order moments:

$$\begin{aligned} E_{\mathcal{T}}\{f(\mathbf{x}_t; \hat{\mathbf{w}})\}, \\ E_{\mathcal{T}}\{f^2(\mathbf{x}_t; \hat{\mathbf{w}})\} \end{aligned}$$

are required. One way to perform these calculations is to apply a Taylor series expansion of $f(\mathbf{x}; \hat{\mathbf{w}})$ around \mathbf{w}^* , i.e.,

$$f(\mathbf{x}; \hat{\mathbf{w}}) = f(\mathbf{x}; \mathbf{w}^*) + \boldsymbol{\psi}(\mathbf{w}^*)\Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^\top \boldsymbol{\Psi}(\mathbf{w}^*)\Delta\mathbf{w} + \dots \quad (6.19)$$

⁸If \mathbf{w}^* is not the global minimum then the WFP will be positive for all estimates, $\hat{\mathbf{w}}$ (over various training sets) lying in a certain vicinity of \mathbf{w}^* . For a specific training set size, N , this claim may not be met; however, when N is large enough then it is obviously met since $\hat{\mathbf{w}} \rightarrow \mathbf{w}^*$ as $N \rightarrow \infty$.

⁹The main assumption is that the cost function when employing $\hat{\mathbf{w}}$ is properly approximated by a second order Taylor series expansion of the cost function around \mathbf{w}^* .

where

$$\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^* \quad (6.20)$$

$$\boldsymbol{\psi}(\mathbf{w}) = \frac{\partial f(\mathbf{x}; \mathbf{w})}{\partial \mathbf{w}} \quad (6.21)$$

$$\boldsymbol{\Psi}(\mathbf{w}) = \frac{\partial \boldsymbol{\psi}(\mathbf{w})}{\partial \mathbf{w}^\top} = \frac{\partial^2 f(\mathbf{x}; \mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \quad (6.22)$$

Since only $\Delta \mathbf{w}$ depends on the training set the essential terms to calculate are the bias $E_{\mathcal{T}}\{\Delta \mathbf{w}\}$, the covariance matrix¹⁰ $\mathbf{V} = E_{\mathcal{T}}\{\Delta \mathbf{w} \Delta \mathbf{w}^\top\}$ and higher order tensors. In Sec. 6.6 the calculation of the bias and variance is presented. Here we merely state some of the results. When dealing with complete models the bias is zero, i.e., $E_{\mathcal{T}}\{\Delta \mathbf{w}\} = \mathbf{0}$; whereas when dealing with incomplete models the bias is zero to order $1/N$, i.e., $E_{\mathcal{T}}\{\Delta \mathbf{w}\} = \mathbf{0} + \mathbf{o}(1/N)$. The general result concerning the covariance matrix is that $\mathbf{V} \propto N^{-1}$. That is, neglecting higher order terms in the Taylor series expansion above:

$$E_{\mathcal{T}}\{f(\mathbf{x}_t; \hat{\mathbf{w}})\} = f(\mathbf{x}; \mathbf{w}^*) + \text{tr}[\boldsymbol{\Psi}(\mathbf{w}^*)\mathbf{V}]. \quad (6.23)$$

The last addend of the *WFP* (see sa:moderr) vanishes in certain cases:

- When the model is an LX-model, i.e., $f(\mathbf{x}; \mathbf{w}) = \mathbf{z}^\top \mathbf{w}$, then $\boldsymbol{\psi} = \mathbf{z}$ and subsequently $\boldsymbol{\Psi} \equiv \mathbf{0}$. That means, when dealing with LX-models¹¹

$$E_{\mathcal{T}}\{f(\mathbf{x}_t; \hat{\mathbf{w}})\} = f(\mathbf{x}; \mathbf{w}^*), \quad (6.24)$$

and in consequence the last addend of the *WFP* vanishes, i.e.,

$$\begin{aligned} WFP &= E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \\ &= E_{\mathcal{T}} \left\{ E_{\mathbf{z}_t} \left\{ [\mathbf{z}_t^\top \mathbf{w}^* - \mathbf{z}_t^\top \hat{\mathbf{w}}]^2 \right\} \right\} \\ &= E_{\mathcal{T}} \left\{ E_{\mathbf{z}_t} \left\{ \Delta \mathbf{w}^\top \mathbf{z}_t \mathbf{z}_t^\top \Delta \mathbf{w} \right\} \right\} \\ &= E_{\mathcal{T}} \left\{ E_{\mathbf{z}_t} \left\{ \text{tr} \left[\mathbf{z}_t \mathbf{z}_t^\top \Delta \mathbf{w} \Delta \mathbf{w}^\top \right] \right\} \right\} \\ &= \text{tr} \left[E_{\mathbf{z}_t} \left\{ \mathbf{z}_t \mathbf{z}_t^\top \right\} \mathbf{V} \right] \end{aligned} \quad (6.25)$$

- When the model is complete, since in that case (cf. the discussion concerning the *MSME*) $\mathbf{w}^\circ = \mathbf{w}^*$, q.e.,

$$g(\mathbf{x}_t) \equiv f(\mathbf{x}_t; \mathbf{w}^*). \quad (6.26)$$

¹⁰The covariance matrix of the estimated weights enters the discussion via the identity: $\Delta \mathbf{w}^\top \boldsymbol{\Psi}(\mathbf{w}^*) \Delta \mathbf{w} = \text{tr}[\boldsymbol{\Psi}(\mathbf{w}^*) \Delta \mathbf{w} \Delta \mathbf{w}^\top]$ where $\text{tr}[\cdot]$ is the trace operator.

¹¹In fact no large N assumption is required in this case if a LS cost function (i.e., $\mathbf{w}^* = \arg \min_{\mathbf{w}} G(\mathbf{w})$) is employed since the last addend of the *WFP* is rewritten as follows:

$$\begin{aligned} 2E_{\mathbf{x}_t} \left\{ (g(\mathbf{x}) - f(\mathbf{x}_t; \mathbf{w}^*)) \mathbf{z}^\top \right\} E_{\mathcal{T}} \{ \Delta \mathbf{w} \} &= 2E_{\mathbf{x}_t, \varepsilon_t} \left\{ (g(\mathbf{x}) - f(\mathbf{x}_t; \mathbf{w}^*) + \varepsilon_t) \mathbf{z}^\top \right\} E_{\mathcal{T}} \{ \Delta \mathbf{w} \} \\ &= 2E_{\mathbf{x}_t, \varepsilon_t} \left\{ e_t \mathbf{z}^\top \right\} E_{\mathcal{T}} \{ \Delta \mathbf{w} \} \\ &= - \frac{\partial G(\mathbf{w}^*)}{\partial \mathbf{w}^\top} E_{\mathcal{T}} \{ \Delta \mathbf{w} \} \\ &= 0. \end{aligned}$$

This is due to the fact that \mathbf{w}^* minimizes $G(\mathbf{w})$, and consequently, $\partial G(\mathbf{w}^*)/\partial \mathbf{w} = \mathbf{0}$.

However, within general NN-models the last addend is non-zero.

The first term of the *WFP* vanishes only when all the following four claims are met:

1. The model is complete.
2. No inherent noise is present, i.e., $\varepsilon \equiv 0$.
3. $\hat{\mathbf{w}}$ is the global minimum.
4. No regularization is employed.

To see this recall from Ch. 5 that $\hat{\mathbf{w}}$ in the case of no regularization is determined so that the gradient $\partial S_N(\hat{\mathbf{w}})/\partial \mathbf{w} = \mathbf{0}$. Evaluating the gradient according to Ch. 5, using the claims above, and furthermore sagely yield:

$$\begin{aligned}
\frac{\partial S_N(\hat{\mathbf{w}})}{\partial \mathbf{w}} &= -\frac{2}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \\
&= -\frac{2}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) [g(\mathbf{x}(k)) - f(\mathbf{x}(k); \hat{\mathbf{w}}) + \varepsilon(k)] \\
&= -\frac{2}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) [f(\mathbf{x}(k); \mathbf{w}^*) - f(\mathbf{x}(k); \hat{\mathbf{w}})]. \tag{6.27}
\end{aligned}$$

Now the gradient vanishes if $\hat{\mathbf{w}} = \mathbf{w}^*$ and hence, the first term of *WFP* equals zero.

When the complexity of the filter architecture increases (e.g., by increasing the number of weights) the *MSME* is reduced¹². This is discussed within various filter architectures in Ch. 3. On the other hand, the *WFP* will typically increase when the number of weights are increased. This is due to the fact that the fluctuation in each weight provide a contribution to the *WFP*. The amount, with which a particular weight contributes, depends on the curvature of $G(\mathbf{w})$ around the optimal weights and the correlation among the individual weight fluctuations. In the case of complete models it is easy to show that the *WFP* increases with the number of parameters. In App. A it is shown that under certain assumptions (see e.g., gendgamcom and Ex. 6.2 below) and when dealing with complete models (i.e., the model error is zero):

$$\Gamma = \underbrace{\sigma_\varepsilon^2}_{\text{Noise Variance}} + \underbrace{\frac{m}{N} \sigma_\varepsilon^2}_{\text{Weight Fluctuation Penalty}} \tag{6.28}$$

where $m = \dim(\mathbf{w})$. However, in the general case this may not be true which is illustrated by the following example:

EXAMPLE 6.2

Consider the linear system:

$$y(k) = \mathbf{z}^\top(k) \mathbf{w}^\circ + \varepsilon(k) \tag{6.29}$$

¹²This fact requires in principle that the complex model contains all models with less complexity by proper settings of the weights. Furthermore, note that the *MSME* may remain unchanged when increasing the number of parameters. Consider, e.g., the case of adding an extra input variable which is independent of the system output, $y(k)$.

where $\mathbf{z}(k) = [z_1(k), z_2(k)]^\top$ is a two-dimensional stochastic Gaussian i.i.d.¹³ signal with zero mean and covariance matrix

$$\mathbf{H} = E \left\{ \mathbf{z}(k) \mathbf{z}^\top(k) \right\} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}. \quad (6.30)$$

$\varepsilon(k)$ is an i.i.d. inherent noise sequence with zero mean and variance σ_ε^2 . In addition, $\varepsilon(k)$ is independent of $\mathbf{z}(k)$. Finally, $\mathbf{w}^\circ = [w_1^\circ, w_2^\circ]^\top$ is the true weight vector.

Now consider two models of the linear system. The incomplete model (IM) (the $z_2(k)$ signal do not enter the model):

$$y(k) = w_1 z_1(k) + e(k), \quad (6.31)$$

and the complete model (CM):

$$y(k) = \mathbf{z}^\top(k) \mathbf{w} + e(k). \quad (6.32)$$

The aim is to show that the *WFP* may be less when dealing with the complete model (the most complex model) than dealing with the incomplete model. In App. F it is shown that if

$$\sigma_\varepsilon^2 < (w_2^\circ)^2 \cdot (\sigma_{12}^2 + \sigma_1^2 \sigma_2^2), \quad (6.33)$$

then the *WFP* of the IM will be less than the *WFP* of the CM although the CM is more complex than the IM. It is worth noting that the result is highly dependent on a number of facts concerning the system and the models. For instance the noise variance, the input distribution, the true weights, etc. Consequently, in general, one is really not capable of providing any firm statements on this topic. \square

Since the *MSME* decreases and, on the other hand, the *WFP* typically increases when the number of weights is increased then there will be a trade off between these two contributions to the average generalization error. There exists a number of ways to trade off these terms. One way is to control the number of weights in the model. In Sec. 6.9 below we focus on using regularization in order to perform the trade off.

6.3.5 Bias/Variance Decomposition

A decomposition which resembles the model error decomposition is the *bias/variance decomposition* which e.g., is proposed in [Geman et al. 92]. The decomposition proceeds as follows:

$$\begin{aligned} \Gamma &= \sigma_\varepsilon^2 + E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \\ &= \sigma_\varepsilon^2 + E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\} + E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\} - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \\ &= \sigma_\varepsilon^2 + \\ &\quad E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\}]^2 \right\} \right\} + \\ &\quad E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\} - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} + \end{aligned}$$

¹³That is the samples of the vector signal are independent, i.e., $\mathbf{z}(k_1)$ is independent of $\mathbf{z}(k_2)$, $\forall k_1 \neq k_2$ and identically distributed.

$$\begin{aligned}
& 2E_{\mathcal{T}} \{E_{\mathbf{x}_t} \{[g(\mathbf{x}_t) - E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\}] \cdot [E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\} - f(\mathbf{x}_t; \hat{\mathbf{w}})]\}\} \\
= & \underbrace{\sigma_{\varepsilon}^2}_{\text{Noise Variance}} + \\
& \underbrace{E_{\mathbf{x}_t} \{[g(\mathbf{x}_t) - E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\}]^2\}}_{\text{Model Bias}} + \\
& \underbrace{E_{\mathcal{T}} \{E_{\mathbf{x}_t} \{[E_{\mathcal{T}} \{f(\mathbf{x}_t; \hat{\mathbf{w}})\} - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2\}\}}_{\text{Model Variance}}. \tag{6.34}
\end{aligned}$$

The average generalization error is thus decomposed in three terms:

- The inherent noise variance which is the lower bound on Γ .
- The *model bias* which is the mean squared distance between the system $g(\mathbf{x})$ and the average filter $E_{\mathcal{T}}\{f(\mathbf{x}; \hat{\mathbf{w}})\}$. Recall that when the model is an LX-model then cf. sa:meanfwhat

$$E_{\mathcal{T}}\{f(\mathbf{x}; \hat{\mathbf{w}})\} = f(\mathbf{x}; \mathbf{w}^*), \tag{6.35}$$

and consequently, the model bias equals the *MSME*. Since the model bias, in general, depends on the fluctuation in the weight estimates we can not guarantee that the model bias decreases (or stay unchanged) when adding extra complexity to the model which is the case regarding the *MSME*. However, a decrease in model bias will still be the typical behavior when adding extra complexity.

- The *model variance* which reflects the average squared deviation from the average filter $E_{\mathcal{T}}\{f(\mathbf{x}; \hat{\mathbf{w}})\}$. Since the model bias equals the *MSME* when employing LX-models the model variance consequently equals the *WFP*.

The bias/variance seems less attractive than the model error decomposition since it is impossible to give a clear interpretation of the model bias/variance unless the model is of type LX. Moreover, general arguments which clarify how the model bias/variance is changed due to changes in the filter architecture are not available.

6.3.6 Simple Invariance Property of the Generalization Error within MFPNN

When dealing with the multi-layer perceptron neural network (MFPNN) architecture it is possible to state a simple invariance property of the generalization error.

Consider an MFPNN architecture with l layers, cf. Sec. 3.2.2. The processing in the r 'th layer is given by

$$\tilde{\mathbf{s}}^{(r)}(k) = \mathbf{h} \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)}(k) \right), \quad r = 1, 2, \dots, l-1 \tag{6.36}$$

where $\mathbf{s}^{(r)}(k)$ is the augmented $(m_r + 1)$ -dimensional state vector of the r 'th layer, $\mathbf{W}^{(r)}$ is the $m_r \times (m_{r-1} + 1)$ weight matrix, and $\mathbf{h}(\cdot)$ is the vector activation function. $\mathbf{s}^{(0)}(k)$ is, per definition, the input to the network. The filter output yields

$$\hat{y}(k) = \mathbf{W}^{(l)} \mathbf{s}^{(l-1)}(k) \tag{6.37}$$

The output can also be expressed as:

$$\hat{y}(k) = \zeta \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)}(k) \right) \quad (6.38)$$

where

$$\zeta \left(\mathbf{u}^{(r)} \right) = \begin{cases} \mathbf{W}^{(l)} \left(\mathbf{h} \left(\mathbf{W}^{(l-1)} \dots \mathbf{h} \left(\mathbf{u}^{(r)} \right) \dots \right) \right) & , 1 \leq r \leq l-1 \\ \mathbf{u}^{(r)} & , r = l \end{cases} \quad (6.39)$$

$$\mathbf{u}^{(r)} = \mathbf{W}^{(r)} \mathbf{s}^{(r-1)}(k) \quad (6.40)$$

is a function defining the processing within the layers $r, r+1, \dots, l$ and $r = 1, 2, \dots, l$.

THEOREM 6.1 Consider an MFPNN with l layers and suppose that only the weights associated with the r layer are adjustable, i.e.,

$$\hat{y}_1(k) = \zeta \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)}(k) \right) \quad (6.41)$$

where $\zeta(\cdot)$ is a fixed function defined in sa:zetadef. Furthermore, consider a similar MFPNN where the output of the $(r-1)$ 'th layer is exposed to linear transformation, i.e.,

$$\hat{y}_2(k) = \zeta \left(\mathbf{\Omega}^{(r)} \mathbf{Q} \mathbf{s}^{(r-1)}(k) \right) \quad (6.42)$$

where \mathbf{Q} is a non-singular $(m_{r-1} + 1) \times (m_{r-1} + 1)$ matrix and $\mathbf{\Omega}^{(r)}$ is the weight matrix of the r 'th layer.

Suppose that weight matrices of the two networks are estimated according to a cost function which only depends on the weights, $\mathbf{W}^{(r)}$, $\mathbf{\Omega}^{(r)}$, respectively. For instance, the usual LS cost function without a regularization term.

Provided that the assumptions hold, then the average generalization error of the two networks is identical. That is, the average generalization error is invariant to a linear transformation of any of the state vectors, $\mathbf{s}^{(r)}$, $r = 0, 1, \dots, l-1$.

PROOF The average generalization error by applying the filter sa:yhatmf1 is according to sa:gamdef and (6.3) given by:

$$\Gamma = E_{\mathcal{T}} \left\{ E_{\mathbf{s}_t^{(0)}, \varepsilon_t} \left\{ e_t^2 \left(\hat{\mathbf{w}}^{(r)} \right) \right\} \right\} \quad (6.43)$$

where $E_{\mathcal{T}}\{\cdot\}$ and $E_{\mathbf{s}_t^{(0)}, \varepsilon_t}$ is expectation w.r.t. the training set, $\mathcal{T} = \{\mathbf{s}^{(0)}(k); y(k)\}$, $k = 1, 2, \dots, N$ and an independent test sample, $[\mathbf{s}_t^{(0)}, \varepsilon_t]$, respectively. The error is given by:

$$\begin{aligned} e_t \left(\hat{\mathbf{w}}^{(r)} \right) &= y_t - \hat{y}_1 \\ &= y_t - \zeta \left(\hat{\mathbf{w}}^{(r)} \mathbf{s}^{(r-1)}(k) \right) \end{aligned} \quad (6.44)$$

where $\hat{\mathbf{w}}^{(r)}$ is the estimated weight matrix, i.e., the weight matrix which minimizes some cost function, $C_N(\mathbf{W}^{(r)})$, which depends on the weights though $\zeta(\cdot)$ only. That is,

$$C_N \left(\mathbf{W}^{(r)} \right) = \frac{1}{N} \sum_{k=1}^N c \left(y(k), \zeta \left(\mathbf{W}^{(r)} \mathbf{s}^{(r-1)}(k) \right) \right) \quad (6.45)$$

where $c(\cdot)$ is the instant cost, e.g., $c(y(k), \hat{y}(k)) = (y(k) - \hat{y}(k))^2$.

Concerning the evaluation of the above expectations one should notice two facts:

- The estimated weight matrix does not depend on the the independent test sample; only on the training set.
- $\mathbf{s}^{(r-1)}$ is a fixed function of the input vector $\mathbf{s}^{(0)}$ since the all weights but the weights in the r 'th layer are assumed fixed.

Consider next Γ associated with the network sa:yhatmf2, i.e

$$\Gamma = E_{\mathcal{T}} \left\{ E_{\mathbf{s}_t^{(0)}, \varepsilon_t} \left\{ e_t^2 \left(\hat{\boldsymbol{\Omega}}^{(r)} \right) \right\} \right\} \quad (6.46)$$

where

$$e_t \left(\hat{\boldsymbol{\Omega}}^{(r)} \right) = y_t - \zeta \left(\hat{\boldsymbol{\Omega}}^{(r)} \mathbf{Q} \mathbf{s}^{(r-1)}(k) \right). \quad (6.47)$$

The cost function is in this case, cf. sa:costinv

$$C_N \left(\boldsymbol{\Omega}^{(r)} \right) = \frac{1}{N} \sum_{k=1}^N c \left(y(k), \zeta \left(\boldsymbol{\Omega}^{(r)} \mathbf{Q} \mathbf{s}^{(r-1)}(k) \right) \right). \quad (6.48)$$

Obviously by letting¹⁴

$$\hat{\boldsymbol{\Omega}}^{(r)} = \hat{\mathbf{w}}^{(r)} \mathbf{Q}^{-1}, \quad (6.49)$$

then

$$C_N \left(\hat{\boldsymbol{\Omega}}^{(r)} \right) = C_N \left(\hat{\mathbf{w}}^{(r)} \right). \quad (6.50)$$

Consequently, the estimated weights, $\hat{\boldsymbol{\Omega}}^{(r)}$, are deterministically related to $\hat{\mathbf{w}}^{(r)}$. Accordingly, $e_t(\hat{\boldsymbol{\Omega}}^{(r)}) \equiv e_t(\hat{\mathbf{w}}^{(r)})$ and then Γ remains unchanged. ■

Even though there is no gain with regard to generalization error by performing linear transformations of the state vectors¹⁵ of an MFPNN there may still be numerical advantages. If the transformation matrix, \mathbf{Q} , is chosen so that the transformed vectors become uncorrelated the condition number of Hessian matrix will typically decrease. As regards the output layer this is definitely true since it is linear in the weights. Consequently, we expect less sensitivity to round off errors. Furthermore, the convergence speed of first order weight estimation algorithms – such as the stochastic gradient algorithm – is increased (see Ch. 5).

6.4 Fundamental Limitations

Recall that the goal is to find the architecture with minimal average generalization error. It is important to emphasize that the above decompositions are *descriptive* of nature, i.e., they explain various facts which give rise to average generalization error contributions. However, they do not directly lead to a *prescription* of how the filter architecture should be altered in order to minimize the average generalization error.

¹⁴The existence of \mathbf{Q}^{-1} is ensured by the assumption that \mathbf{Q} is non-singular.

¹⁵Formally, we showed linear invariance for a single layer only. However, when simultaneously performing linear transformations of several state vectors, the invariance is assured by using the theorem successively.

6.4.1 Complete Models

Regarding complete models the *MSME* and the last addend of *WFP* are zero. Consequently, the only contribution to Γ is

$$WFP = E_{\mathcal{T}} \left\{ E_{\mathbf{x}_t} \left\{ [f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2 \right\} \right\} \quad (6.51)$$

which merely stems from dealing with a finite training set. However, to be sure that the model is complete requires much a priori knowledge, so we claim the common case is that the model is incomplete.

Now consider two types of complete models:

1. The *minimal complete model* which contains the necessary weights only, i.e., if a weight is removed the model becomes incomplete. The only way to reduce Γ in this case is to collect more training data so that $\hat{\mathbf{w}}$ becomes closer to \mathbf{w}^* on the average.
2. The *over-parameterized complete model* which contains more than the necessary number of weights; hence, a subset of the weights will optimally be equal to zero¹⁶. If one is able to detect the weights which optimally are zero then these weights can be removed and consequently, the *WFP* is reduced as the “noise” stemming from these weights is eliminated. This issue is further treated in Sec. 6.6.

6.4.2 Incomplete Models

Regarding incomplete models the *MSME* is non-zero. Both the *MSME* and the *WFP* depends on the system; either directly when $g(\mathbf{x})$ enters the expression¹⁷ or indirectly through \mathbf{w}^* and $\hat{\mathbf{w}}$ ¹⁸.

In general it is impossible to predict how the *MSME* changes when the model is changed since it is influenced by both the structure of the system, i.e., $g(\mathbf{x})$, and the distribution of the input. The structure of the system is of course unknown and the distribution of the input is also often unknown. This fact is indeed a fundamental limitation in the search for the optimal architecture. In order to handle this limitation two strategies may be suggested:

1. A priori knowledge on the structure of the system may be used. Obviously only partial knowledge is available of the system and furthermore this knowledge may be hard to formulate explicitly. An example of such a priori knowledge may be that we believe that the system, $g(\mathbf{x})$, is a polynomial system (see Sec. 3.2.1) with unknown order. In addition, if the function, $g(\mathbf{x})$, is supposed to vary slowly with \mathbf{x} then the coefficients (the true weights) will decline to zero as the order increases according to some scheme.
2. Alternatively a trial and error technique may be used. That is, pick two architectures, estimate the average generalization error (see Sec. 6.5.1) and choose the architecture with the lowest average generalization error.

¹⁶The subset is not necessarily unique. Consider e.g., an MFPNN which contains two inputs which are linearly dependent. Then the weights associated with either the first or the second input can be set equal to zero.

¹⁷This is the case concerning the *MSME* and the second addend of *WFP* cf. sa:moderr.

¹⁸This is due to the fact that $\hat{\mathbf{w}}$ is the weights which solve the equation: $\partial C_N(\mathbf{w})/\partial \mathbf{w} = 0$. Now, the error signal $e(k) = y(k) - f(\mathbf{x}(k); \mathbf{w}) = g(\mathbf{x}) - f(\mathbf{x}(k); \mathbf{w}) + \varepsilon(k)$ which depends on the system, $g(\mathbf{x})$, enters this equation.

The prediction of how *WFP* changes when the number of weights, m , is increased is afflicted with the same difficulties as mentioned above. In particular, recall from Ex. 6.2 that *WFP* declined when adding extra weights. This fact indeed conflicted with the intuition (from complete models) that the *WFP* increases with m . On the other hand, it may still be possible to predict how the *WFP* changes when reducing m . This is illustrated by the following example: Suppose that an extra weight is considered to be included into the model. The effect of including this weight highly depends on the structure of $g(\mathbf{x})$. Consequently, the prediction whether the inclusion of the weight provides a significant improvement or not can only be answered if some a priori knowledge on $g(\mathbf{x})$ is available. On the other hand, when facing the problem of whether a particular weight should be removed the information concerning $g(\mathbf{x})$ is already available in terms of the weight estimate.

The aim of reducing m is to ensure that *WFP* declines without causing any increase in *MSME*. In Sec. 6.6 we present a statistical framework for testing hypotheses¹⁹ on the optimal weights. Consider, e.g., the hypothesis: $w_i^* = a$. If the hypothesis is true then fixing w_i at the value a – and thereby reducing the number of weights by one – does not cause any increase in the *MSME*. Moreover, the *WFP* will intuitively decrease as $\hat{w}_i = w_i^* = a$; hence, the contribution from the i 'th weight is eliminated. However, it has p.t. not been possible to give any cogent proof concerning a general incomplete NN-model. Instead the simple example below may provide an intuitive argument.

EXAMPLE 6.3

Consider the linear system:

$$y(k) = w_1^\circ + w_2^\circ x(k) + \varepsilon(k) \quad (6.52)$$

where $x(k) \in \mathcal{N}(0, \sigma_x^2)$ is an i.i.d. input signal, $\varepsilon(k)$ is an i.i.d. inherent noise sequence with zero mean and variance σ_ε^2 . In addition, $\varepsilon(k)$ is independent of $x(k)$. Finally, w_1°, w_2° are the true weights.

Now consider two incomplete models of the linear system. The first model, called the unrestricted model, obeys:

$$\begin{aligned} y(k) &= w_1 + w_3 x^2(k) + e(k) \\ &= \mathbf{w}^\top \mathbf{z}(k) + e(k; \mathbf{w}) \end{aligned} \quad (6.53)$$

where $\mathbf{w} = [w_1, w_3]^\top$, $\mathbf{z} = [1, x^2(k)]^\top$, and $e(k; \mathbf{w})$ is the error signal. The model is obviously incomplete since no linear term, i.e., $x(k)$, is present. On the other hand, the model contains a quadratic term, $x^2(k)$ which not is present in the system. Moreover, since $x(k)$ is a zero mean Gaussian signal $x(k)$ is uncorrelated with $x^2(k)$. This implies (as elaborated in App. F) that the optimal setting of w_3 w.r.t. the LS cost function²⁰ is $w_3^* = 0$. That is, none of the systematic error²¹ is explained by including the $x^2(k)$ term. Due to the finite training set w_3 is, in general, assigned a non-zero value which results in a contribution to the *WFP*. Intuitively it seems clear not to include this term after all. This leads to the consideration of the restricted model given by:

$$y(k) = w_1 + e(k; w_1). \quad (6.54)$$

¹⁹The common hypothesis is that some of the weights optimally are equal to zero.

²⁰Note that the optimal setting depends on the chosen cost function and the statistic of the input. That is, if $x(k)$ is not Gaussian one may profit by including the $x^2(k)$ term.

²¹That is, the part of the error signal $e(k; \mathbf{w})$ which depends on the input.

Since the optimal setting of w_3 in the unrestricted model equals zero the *MSME*'s of the two models are identical. That is, in the limit of an infinite training set the generalization ability is equally good. The goal is now to demonstrate that WFP_u of the unrestricted model is larger than WFP_r of the restricted model. Consequently, if some weight elimination procedure cf. Sec. 6.6 and Sec. 6.7 leads to the conclusion that a weight (optimally) can be removed then the restricted model has a better generalization ability.

In App. F it is shown that

$$WFP_u = \frac{6(w_2^\circ)^2 \sigma_x^2 + 2\sigma_\varepsilon^2}{N}, \quad (6.55)$$

and

$$WFP_r = \frac{(w_2^\circ)^2 \sigma_x^2 + \sigma_\varepsilon^2}{N}. \quad (6.56)$$

Accordingly, $WFP_u \geq WFP_r$, since all involved terms are positive. Consequently, the average generalization error of the restricted model sa:ex2m2 is better than that of the unrestricted model sa:ex2m1. \square

6.5 Generalization Error Estimates

In this section we attack the problem of estimating the generalization error. This serves two purposes:

1. It enables us to settle the performance of a filter architecture. That is, ascertaining whether the filter is capable of satisfying the specified requirements or not.
2. It provides a tool for synthesizing a proper filter architecture since we prefer the architecture which has the minimum generalization error.

Below several generalization error estimates from the literature are described and a novel estimator developed. The estimators differ on various model assumptions. The cross-validation estimators cf. Sec. 6.5.3, Sec. 6.5.4 are the less restrictive since they do not presume anything about the model. The remaining estimators differ mainly on two model properties:

- Complete versus incomplete models (see Def. 6.3). Most estimators assume that the model is complete, i.e., capable of modeling the system under consideration perfectly. However, when modeling nonlinear systems by neural networks lack of a priori knowledge typically results in that complete models cannot be guaranteed. The possibility of handling incomplete models is thus of major significance.
- LX- versus NN-models, i.e., whether the models are linear or nonlinear in the weights, respectively. In this context handling NN-models is important since many of the filter architectures presented in Ch. 3 (e.g., the multi-layer neural network) are nonlinear in the weights.

6.5.1 Basic Architecture Synthesis Algorithm

Estimation of the generalization error of various alternative filter architectures enables us to rank the models w.r.t. quality, i.e., the ability to pick the model within the possible

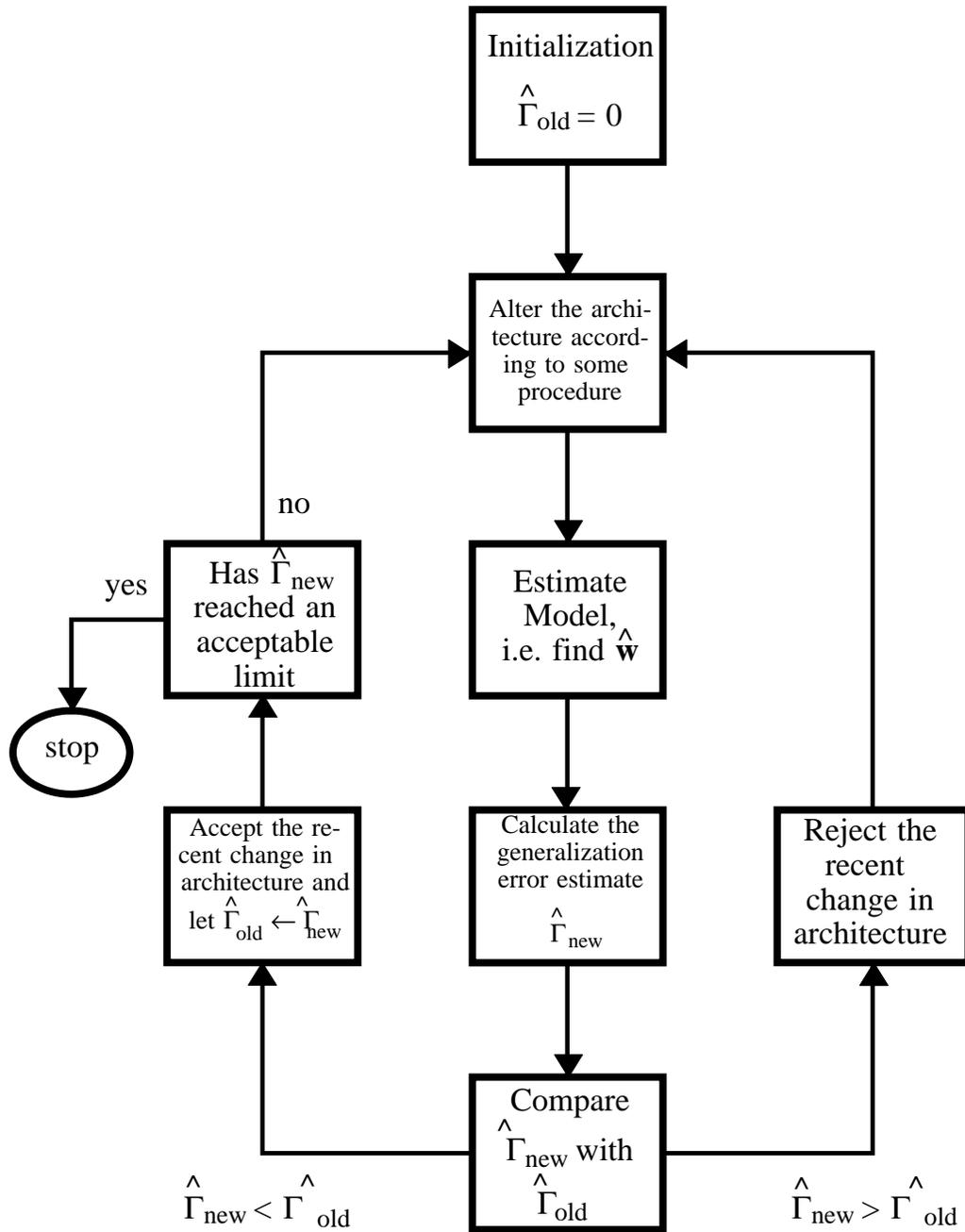


Figure 6.1: The flow chart of the basic architecture synthesis algorithm. $\hat{\Gamma}$ denotes an arbitrary generalization error estimate.

alternatives which generalize the best. This is the content of the basic architecture synthesis algorithm depicted in Fig. 6.1. A central element of the algorithm is a procedure for gradual alteration of the architecture. The most simple strategy is just to select a

basic architecture, e.g., multilayer feed-forward neural networks, and then vary the number of weights, for instance by varying the number of hidden neurons. However, in Sec. 6.8 and Sec. 6.7 several more intelligent proposals are considered. The box which specify the stop criterion may be replaced by other criteria. For instance – due to computational considerations – one may in advance specify a limited number of architecture candidates.

6.5.2 The Mean Square Training Error

A first attempt to estimate the generalization error could be to use the LS cost function, $S_N(\hat{\mathbf{w}})$ (see Ch. 5), since if $e^2(k; \mathbf{w})$ is a mean-ergodic sequence, i.e., cf. [Papoulis 84a, Ch. 9-5]

$$E_{\mathbf{x}(k), \varepsilon(k)} \{e^4(k; \hat{\mathbf{w}})\} < \infty, \quad (6.57)$$

and

$$E_{\mathbf{x}(k), \varepsilon(k)} \{e^2(k; \hat{\mathbf{w}})e^2(k + \tau; \hat{\mathbf{w}})\} \rightarrow 0, \quad \text{as } \tau \rightarrow \infty, \quad (6.58)$$

then

$$\lim_{N \rightarrow \infty} \{S_N(\hat{\mathbf{w}})\} = G(\hat{\mathbf{w}}). \quad (6.59)$$

That is, in the limit of large training sets the LS cost function coincides with the generalization error. However, in practice $S_N(\mathbf{w})$ is an unreliable estimator. This is due to the fact that the estimated weights depend on the training samples. Recall from sa:decom1 that the generalization error is decomposed as:

$$G(\hat{\mathbf{w}}) = \underbrace{E_{\varepsilon} \{\varepsilon_t^2\}}_{\text{Noise Variance}} + \underbrace{E_{\mathbf{x}_t} \{[g(\mathbf{x}_t) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2\}}_{\text{Mean Square Adjustment Error}}. \quad (6.60)$$

Both the noise variance and the mean square adjustment error are usually non-zero (and positive)²²; however the LS cost is bounded by

$$S_N(\hat{\mathbf{w}}) > 0. \quad (6.61)$$

The case of zero cost can in principle be obtained when the number of training data, N , equals the number of weights, m . Since for each sample, $k \in [1; N]$, the k 'th weight can be adjusted so that $e(k; \mathbf{w}) = 0$. Hence, the cost function typically will be smaller than the generalization error. In general this is not true as illustrated by the following example:

EXAMPLE 6.4

Consider a simple linear data generating system:

$$y(k) = \mathbf{z}^\top(k) \mathbf{w}^\circ + \varepsilon(k) \quad (6.62)$$

where

- $\mathbf{z}(k)$ is a 3-dimensional Gaussian distributed input vector with zero mean and (positive definite) covariance matrix:

$$\mathbf{H} = E\{\mathbf{z}(k)\mathbf{z}^\top(k)\} = \begin{bmatrix} 2.84 & -3.05 & -9.55 \\ -3.05 & 10.7 & 15.4 \\ -9.55 & 15.4 & 36.8 \end{bmatrix}. \quad (6.63)$$

²²Except the trivial case of dealing with a complete model and no inherent noise where the generalization error equals zero (see further Sec. 6.3.4).

- $\mathbf{w}^\circ = [1, 2, -1.5]^\top$ is the true weight vector.
- $\varepsilon(k)$ is a Gaussian noise with zero mean and variance $\sigma_\varepsilon^2 = 0.1 \cdot V\{y(k)\} = 5.23$.

The system is modeled by the complete LL-model

$$y(k) = \mathbf{z}^\top(k)\mathbf{w} + e(k; \mathbf{w}), \quad (6.64)$$

by minimizing the LS cost function, $S_N(\mathbf{w}) = N^{-1} \sum_{k=1}^N e^2(k; \mathbf{w})$, on the training set $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, $k = 1, 2, \dots, N$. According to Ch. 5 the estimated weight vector, $\hat{\mathbf{w}}$, becomes:

$$\hat{\mathbf{w}} = \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)y(k) \quad (6.65)$$

where $\mathbf{H}_N = N^{-1} \sum_{k=1}^N \mathbf{z}(k)\mathbf{z}^\top(k)$. According to sa:gwhatex the generalization error equals:

$$\begin{aligned} G(\hat{\mathbf{w}}) &= \sigma_\varepsilon^2 + E_{\mathbf{z}} \left\{ \left[\mathbf{z}^\top(k)\mathbf{w}^\circ - \mathbf{z}^\top(k)\hat{\mathbf{w}} \right]^2 \right\} \\ &= \sigma_\varepsilon^2 + \Delta\mathbf{w}^\top \mathbf{H} \Delta\mathbf{w} \end{aligned} \quad (6.66)$$

where $\Delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^\circ$. Note that $G(\hat{\mathbf{w}}) \geq \sigma_\varepsilon^2$ as \mathbf{H} is positive definite.

In order to comprehend the relation between the training cost, $S_N(\hat{\mathbf{w}})$ and the generalization error $G(\hat{\mathbf{w}})$ they are considered as stochastic variables due to the statistical variation in $\hat{\mathbf{w}}$ when applying different training sets. Experimentally Q independent training sets, $\mathcal{T}^{(s)}$, $s = 1, 2, \dots, Q$ are generated whereupon the training cost and generalization error are calculated for each realization. In the example we used $Q = 81000$. In Fig. 6.2 the estimated probability density functions (p.d.f.'s): $p_{S_N}(\sigma^2)$, $p_G(\sigma^2)$ of the training cost and the generalization error, respectively, are shown for $N = 6$ and $N = 20$. The first thing to notice is that the lower limit of $G(\hat{\mathbf{w}})$ actually equals σ_ε^2 indicated by the vertical dotted line. It is seen that the shape of $p_G(\sigma^2)$ is more narrow when $N = 20$. This is due to the fact that in the limit $N \rightarrow \infty$ $\hat{\mathbf{w}} \rightarrow \mathbf{w}^\circ$; consequently, the generalization error equals σ_ε^2 . The characteristic of $p_{S_N}(\sigma^2)$ is that the peak moves toward larger values of σ^2 when increasing N . Contemporary it becomes more narrow since in the limit $N \rightarrow \infty$ it equals the generalization error (equal to σ_ε^2). Observe that when $N = 6$ the lower limit of the training cost reaches zero approximately. The reason is that only two training samples are available per weight.

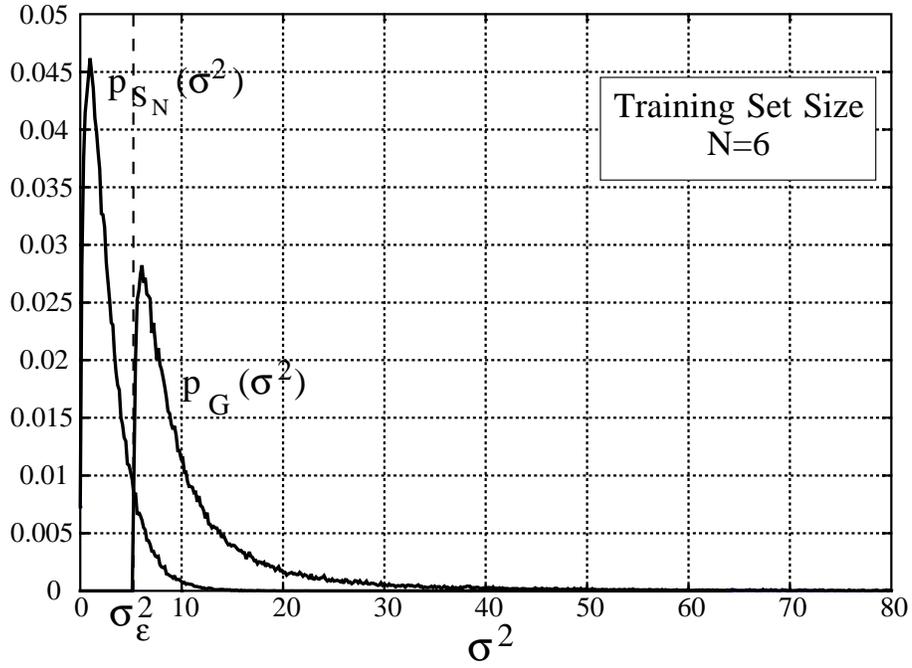
The statement: $S_N(\hat{\mathbf{w}})$ is lower than $G(\hat{\mathbf{w}})$ is only true within a certain probability. The experiment resulted in the following estimate:

$$\text{Prob} \{S_N(\hat{\mathbf{w}}) < G(\hat{\mathbf{w}})\} = \begin{cases} 0.97 & N = 6 \\ 0.85 & N = 20 \end{cases} \quad (6.67)$$

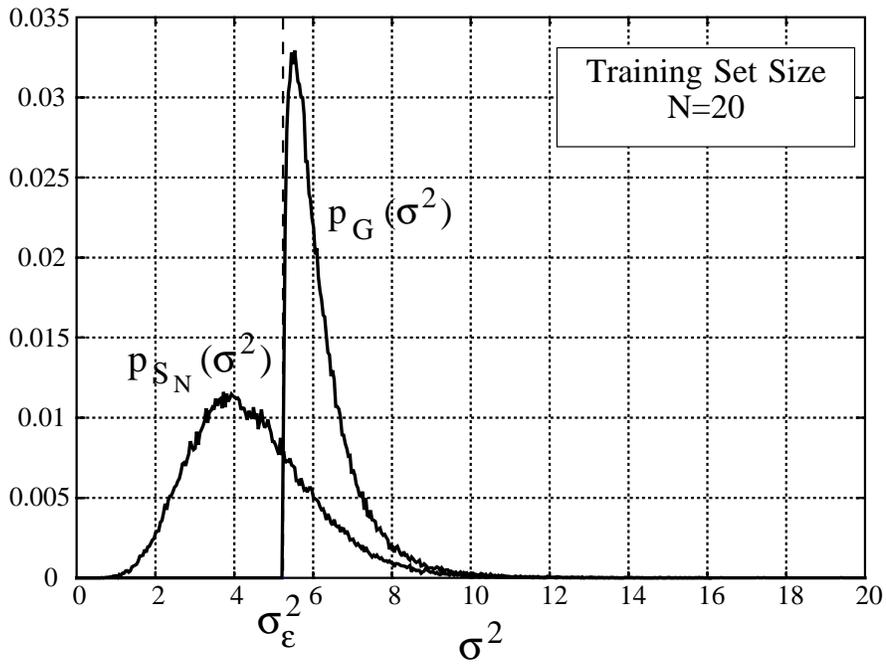
where $\text{Prob}\{\cdot\}$ denotes probability. That is, the statement achieve more evidence when N is small. \square

6.5.3 Cross-Validation

A straight forward generalization error estimate which remedies the dependence between $\hat{\mathbf{w}}$ and the data used for estimating the mean squared error is the *cross-validation estimate*, see e.g., [Draper & Smith 81, Sec. 6.8 & 8.3], [Ljung 87, Ch. 16], [Stone 74], [Toussaint 74].



(a)



(b)

Figure 6.2: The estimated p.d.f.'s $p_{S_N}(\sigma^2)$, $p_G(\sigma^2)$ of the training cost and the generalization error, respectively.

The technique simply consists in using $N - N_c$ of the samples, assembled in $\mathcal{T}_{\text{train}}$, for estimation of $\hat{\mathbf{w}}_{N-N_c}$ ²³ (i.e., training) and the remaining – preferably independent – N_c samples, assembled in $\mathcal{T}_{\text{cross}}$, for calculating the generalization error estimate as the average of the squared error signal. That is:

$$\hat{\mathbf{w}}_{N-N_c} = \arg \min_{\mathbf{w}} C_{N-N_c}(\mathbf{w}) \quad (6.68)$$

where $C_{N-N_c}(\mathbf{w})$ is the cost function employing $N - N_c$ training samples, The cross-validation estimate (C -estimate) becomes:

$$C_\nu(\mathcal{T}) = \frac{1}{N_c} \sum_{k=N-N_c+1}^N e^2(k; \hat{\mathbf{w}}_{N-N_c}). \quad (6.69)$$

The dependence on the training set used to calculate the estimate is sometimes explicitly indicated by \mathcal{T} . The index ν is defined as the percentage of the total number of training samples used for estimating the weights, i.e.,

$$\nu = \frac{N - N_c}{N} \cdot 100\%. \quad (6.70)$$

Note with this definition, $N - N_c = \nu N$ and $N_c = (1 - \nu)N$. Under mild assumptions on the error signal, cf. sa:cas1, (6.58), then

$$\lim_{N \rightarrow \infty} C_\nu = G(\mathbf{w}^*). \quad (6.71)$$

An obvious drawback using this technique is that only $N - N_c$ data can be used for training (weight estimation). Since the weight fluctuation penalty, cf. Sec. 6.3.4, increases when the number of training data decreases the resulting model will generalize worse than when using all data for training. Consequently, C_ν is a biased estimator of $G(\hat{\mathbf{w}}_N)$. Conversely, C_ν is an unbiased estimator of $G(\hat{\mathbf{w}}_{\nu N})$ if $\mathcal{T}_{\text{train}}$ is independent on $\mathcal{T}_{\text{cross}}$. However, in a signal processing context this independence is seldomly sustained due to correlation in the input signal, $\mathbf{x}(k)$. Furthermore, there is a variance contrintion to the error on the C -estimator which reads:

$$\begin{aligned} V\{C_\nu\} &= E_{\mathcal{T}_{\text{cross}}} \left\{ [C_\nu - E\{C_\nu\}]^2 \right\} \\ &= \frac{1}{N_c} \sum_{\tau=-N}^N \frac{N - |\tau|}{N} \gamma_C(\tau) \end{aligned} \quad (6.72)$$

where $\gamma_C(\tau)$ is the crosscovariance function

$$\gamma_C(\tau) = E \left\{ \left[e^2(k; \hat{\mathbf{w}}_{N-N_c}) - G(\hat{\mathbf{w}}_{N-N_c}) \right] \left[e^2(k + \tau; \hat{\mathbf{w}}_{N-N_c}) - G(\hat{\mathbf{w}}_{N-N_c}) \right] \right\}. \quad (6.73)$$

Thus $V\{C_\nu\} = O(N_c^{-1})$ which implies a trade off. On the one hand N_c should preferably be as small as possible in order to ensure a proper weight estimate; on the other hand N_c should be as large as possible ensuring C_ν to be a reliable estimator of $G(\mathbf{w})$. The optimal setting of N_c , and thereby ν , depends on various factors such as: the system, the model, the number of training data, required confidence interval on C_ν , etc.. Consequently, an

²³Here we explicitly denote that the weight estimate is based on $N - N_c$ training data.

optimal setting is commonly not accessible. Generally we require $\nu \geq 50\%$ with $\nu = 50\%$ being a frequent setting.

The C -estimate possesses two immediate advantages: First, notice that no assumptions on the system, the model, and the p.d.f.'s of the input and the inherent noise are required. Secondly, the computational complexity is low. Recall from Ch. 5 that the computational complexity, CP , in this work is measured as the number of multiplications and divisions²⁴. The computational complexity of the C -estimate, CP_C , is:

$$CP_C = CP_{\text{train}(\nu N)} + N_c \left(CP_{\hat{y}} + 1 \right) + 10 \quad (6.74)$$

where $CP_{\text{train}(\nu N)}$ is the complexity involved in training on νN examples, i.e., the estimation of the weights, $\hat{\mathbf{w}}_{\nu N}$ and $CP_{\hat{y}}$ is the complexity of calculating one sample of the filter output, \hat{y} , which is identical to the complexity involved in one sample of the error signal, since per definition: $e = y - \hat{y}$. These complexities are highly dependent on the employed architecture and the chosen weight estimation algorithm. In Ch. 5 a further elaboration is given, in particular, the multi-layer feed-forward perceptron neural network (MFPNN) architecture is considered. Dealing with a 2-layer network then cf. pa:ycom2 and (5.200)

•

$$CP_{\hat{y}} \approx m \frac{p + 12}{p + 2} \quad (6.75)$$

where m is the total number of weights and $p = \dim(\mathbf{z})$ is the number of input neurons.

• Applying a second-order algorithm, like the RGNB-algorithm then

$$CP_{\text{train}(N)} \approx \frac{3}{2} N \text{itr} m^2 \quad (6.76)$$

when m is appropriately large. itr denotes the number of times the training set is replicated during training.

That is cf. sa:cpcrs:

$$CP_C \approx \nu \frac{3}{2} N \text{itr} m^2 + (1 - \nu) N \left(m \frac{p + 12}{p + 2} + 1 \right) + 1 \approx \frac{3}{2} \nu N \text{itr} m^2. \quad (6.77)$$

6.5.4 Leave-One-Out Cross-Validation

A different cross-validation estimate is the leave-one-out cross-validation estimate (L -estimate) [Draper & Smith 81, Sec. 6.8 & 8.3], [Toussaint 74]. The idea is to successively leave one sample in the training set out for cross-validation and then use the other for training. This is depicted in Fig. 6.3. The estimator becomes:

$$L(\mathcal{T}) = \frac{1}{N} \sum_{j=1}^N e^2(j; \hat{\mathbf{w}}_{(j)}) \quad (6.78)$$

where $\hat{\mathbf{w}}_{(j)}$ is the weight estimate obtained by training on the training set: $\{\mathbf{x}(k); y(k)\}$,

$$\begin{aligned} k &= 2, 3, \dots, N & , j &= 1 \\ k &= 1, 2, \dots, j-1, j+1, \dots, N & , j &\in [2; N-1] . \\ k &= 1, 2, \dots, N-1 & , j &= N \end{aligned} \quad (6.79)$$

²⁴One division is considered to be equivalent to 10 multiplications cf. Ch. 5

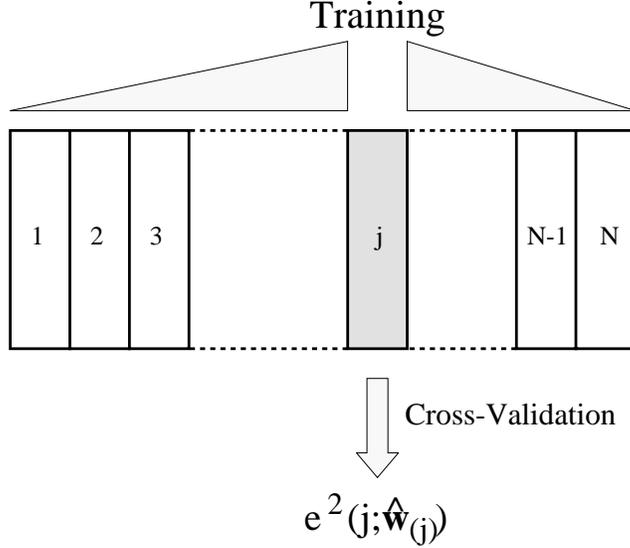


Figure 6.3: The principle of leave-one-out cross-validation. The estimate, $\hat{\mathbf{w}}_{(j)}$, is based on the $N - 1$ training samples $k = 1, 2, \dots, j - 1, j + 1, \dots, N$ and the j 'th sample is used for cross-validation, i.e., calculation of the squared error. The procedure is repeated for all $j \in [1; N]$ and the estimate results from averaging the squared errors.

Regarding the L -estimator as an estimator of the the generalization error involves the following matters: First notice that $N - 1$ is used for training, i.e., $E\{e^2(j; \hat{\mathbf{w}}_{(j)})\}$ is only slightly biased relative to $G(\mathbf{w}_N)$ if the j 'th sample is independent of the other. In practice; however, correlation among training samples implies an extra bias. Compared to the C -estimator we will expect that the L -estimator is less biased (see also [Toussaint 74]). Still the L -estimator possesses a significant variance due to the fluctuations in the estimates $\hat{\mathbf{w}}_j$. Comparing the C - and the L -estimators one faces a bias/variance dilemma. The C -estimator being a strongly biased estimator of $G(\mathbf{w}_N)$ with relatively small variance since we average w.r.t. N_c samples, whereas the L -estimator has a small bias but a significant variance. Furthermore, unlike the C -estimator, the L -estimator has an average over different (after all dependent) training sets.

These facts motivate an *intermediate* cross-validation estimator – cf. [Toussaint 74] – where a number of training samples, say N_c , are excluded for cross-validation and the remaining are used for training. This procedure is repeated $\lfloor N/N_c \rfloor$ times and the final estimate results by averaging the individual cross-validation estimates. The bias and variance are balanced by adjusting N_c .

Another drawback of the L -estimator is that the computational complexity, CP_L , is large due to the fact that training is repeated N times. We get:

$$CP_L = N \cdot CP_{\text{train}(N-1)} + N \left(CP_{\hat{y}} + 1 \right) + 10. \quad (6.80)$$

The complexities involved are explained in the previous subsection. Dealing with a 2-layer MFPNN architecture and the RGNB-algorithm for estimating the weights then cf. sa:cpy,

(6.76) the complexity is approximately given by:

$$CP_L \approx \frac{3}{2}N(N-1) \text{itr } m^2 + N \left(m \frac{p+12}{p+2} + 1 \right) + 1 \approx \frac{3}{2}N^2 \text{itr } m^2. \quad (6.81)$$

6.5.5 An Information Theoretic Criterion

In [Akaike 74] a generalization ability estimator, called the information theoretic criterion (*AIC*) is presented. The estimator is based on a maximum likelihood framework, q.e., that the weights are estimated due to a maximum likelihood procedure:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w}) \quad (6.82)$$

where the cost function C_N is defined by the negative log-likelihood function

$$C_N(\mathbf{w}) = -\frac{1}{N} \ln L_N(\mathbf{w}) \quad (6.83)$$

where $L_N(\cdot)$ is the likelihood function

$$L_N(\mathbf{w}) = p(y(1), \dots, y(N) | \mathbf{x}(1), \dots, \mathbf{x}(N); \mathbf{w}). \quad (6.84)$$

$p(\cdot|\cdot)$ denotes the joint conditional p.d.f. of all $y(k)$ in the training set conditioned on the inputs and the parameters. The generalization error is defined as the the expected cost function, i.e., the expected negative log-likelihood on an independent sample $\{\mathbf{x}_t; y_t\}$, cf. sa:explike

$$G(\mathbf{w}) = E_{\mathbf{x}_t, \varepsilon_t} \{ -\ln[p(y_t | \mathbf{x}_t; \mathbf{w})] \} \quad (6.85)$$

This definition of the generalization error has a close connection with the Kullback-Leibler mean information [Kullback 59] as pointed out in [Akaike 74]. Let $p^\circ(y|\mathbf{x})$ be the true distribution of y when \mathbf{x} is known²⁵. If the estimated model is properly chosen then a noticeable resemblance between $p(y|\mathbf{x}; \hat{\mathbf{w}})$ and $p^\circ(y|\mathbf{x})$ is expected. The Kullback-Leibler mean information,

$$I(p^\circ; p) = E_{\mathbf{x}, \varepsilon} \left\{ \ln \frac{p^\circ(y|\mathbf{x})}{p(y|\mathbf{x}; \hat{\mathbf{w}})} \right\} = E_{\mathbf{x}, \varepsilon} \{ \ln p^\circ(y|\mathbf{x}) \} + G(\hat{\mathbf{w}}) \quad (6.86)$$

measures the distance between these p.d.f.'s. Notice that the first term is model independent; hence, the mean information is small if the generalization is small, i.e., $p^\circ(y|\mathbf{x})$ is close to $p(y|\mathbf{x}; \hat{\mathbf{w}})$.

A crucial assumption concerning the *AIC* is that the model is complete, i.e., in terms of p.d.f.'s: There exists a true weight vector, \mathbf{w}° so that $p^\circ(y|\mathbf{x}) = p(y|\mathbf{x}; \mathbf{w}^\circ)$. Furthermore, if $\hat{\mathbf{w}}$ is sufficiently close to \mathbf{w}° ²⁶ then

$$AIC = -\frac{2 \ln L_N(\hat{\mathbf{w}})}{N} + \frac{2m_{\text{tot}}}{N} \quad (6.87)$$

where m_{tot} is the total number of adjustable parameters, this includes the m weights and some additional parameters specifying e.g., the p.d.f. of the inherent noise. A derivation

²⁵When the data generating system is $y = g(\mathbf{x}) + \varepsilon$ then $p^\circ(y|\mathbf{x}) = p_\varepsilon(y - g(\mathbf{x}))$ where $p_\varepsilon(\varepsilon)$ is the p.d.f. of the inherent noise.

²⁶This holds asymptotically, i.e., $\hat{\mathbf{w}} \rightarrow \mathbf{w}^\circ$, as $N \rightarrow \infty$. Consult e.g., the consistency result Th. 5.1.

of AIC is provided in [Akaike 74]; however, consult also [Ljung 87, Ch. 16] for a plain derivation.

AIC is thus a sum of the training cost, $C_N(\hat{\mathbf{w}})$, and a positive penalty term which penalizes having too many adjustable parameters. This seems reasonable since we know that the cost function typically is smaller than the generalization error (see Sec. 6.5.2 and Sec. 6.5.8). In the case of a Gaussian distributed error signal, i.e., $e(k) \in \mathcal{N}(0, \sigma_e^2)$, the AIC [Ljung 87, Ch. 16] reads (see also Ex. 6.1)

$$AIC = 1 + \ln(2\pi) + \ln S_N(\hat{\mathbf{w}}) + \frac{2(m+1)}{N} \quad (6.88)$$

where $S_N(\cdot)$ is the usual LS cost function $S_N(\hat{\mathbf{w}}) = N^{-1} \sum_{k=1}^N e^2(k; \hat{\mathbf{w}})$ and $\ln(\cdot)$ is the natural logarithm. The total number of adjustable parameters equal $m+1$; viz. the m weights and the unknown error variance, σ_e^2 . Note that the constant terms in connection with architecture synthesis cf. Sec. 6.5.1 can be omitted.

The AIC -estimator has been a popular tool within signal processing, system identification and time series analysis. However, it is known to have some drawbacks. First the complete model assumption may be questionable in nonlinear filtering applications. Secondly, the estimator is known to be inconsistent when used for determination of the order of an autoregressive model²⁷ (AR) [Haykin 91], [Kashyap 80], that is when $N \rightarrow \infty$ the AR-model with minimal AIC has not the correct order. The tendency is that the order is chosen too large. The inconsistency can be remedied [Kashyap 80] by introducing a penalty term of the form $2mK(N)/N$ where $K(N) > 0$ obeys,

$$K(N) \rightarrow \infty, \quad \frac{K(N)}{N} \rightarrow 0, \quad \text{as } N \rightarrow \infty. \quad (6.89)$$

A commonly used function is $K(N) = \ln N$ employed e.g., in the Minimum Description Length estimator [Rissanen 78]. Two facts should be noticed:

- One point of view may be that the goal is a low generalization error. That is, an incorrect model order may be accepted if no substantial increase in generalization error results.
- The inconsistency result is valid for infinite training sets only, i.e., the AIC -estimator may still be better for moderate training sets as pointed out by [Haykin 91, Ch. 2.10].

An estimator, called the Final Information Statistic [Fogel 91], based on the AIC -estimator has been developed to deal with multi-layer feed-forward neural networks. The estimator is restricted to handle classification problems (i.e., binary output(s)).

²⁷The AR-model takes the form:

$$y(k) = \sum_{i=1}^m a_i y(k-i) + \varepsilon(k)$$

where m is the order, a_i the AR-parameters and $\varepsilon(k)$ a white noise process. Concerning the nomenclature used in this thesis this equation should rather be denoted the AR-system, i.e., the process which generate data. The model used for identification (cf. the paragraph concerning time series identification Ch. 1) is $y(k) = \mathbf{x}^T(k)\mathbf{w} + e(k)$ where $\mathbf{x}(k) = [y(k-1), y(k-2), \dots, y(k-m)]^T$ and \mathbf{w} is the weight vector.

6.5.6 The Final Prediction Error Estimator

The Final Prediction Error Estimator (*FPE*) [Akaike 69] was invented with reference to the determination of AR-model orders (see above). *FPE* is an estimate of the average generalization error, Γ , (see sa:gamdef) given by:

$$FPE = \frac{N + m}{N - m} S_N(\hat{\mathbf{w}}) \quad (6.90)$$

where $S_N(\hat{\mathbf{w}})$ is the LS cost function evaluated at the estimated weights, $\hat{\mathbf{w}}$, m is the number of weights, and N is the number of training examples. For $m > N$, we note that the factor $(N + m)/(N - m) > 1$ penalizes having too many weights. In [Akaike 69] it is claimed that the *FPE* works both complete and incomplete AR-models, i.e., models which have more or less weights than the true model, respectively. However, in Sec. 6.5.8 the *FPE*-estimator will be discussed intensively and it is demonstrated that the *FPE*-estimate – in theory – is valid for complete models only. On the other hand, it is applicable for general complete NN-models of the form sa:model. The *FPE*-estimate is characterized by two immediate advantages: First, all data in the training set can be used for estimating the weight in contrast to dealing with cross-validation estimates. Secondly, the computational complexity (see above), *CP* is low. This involves training, one cost function evaluation, and one multiplication and division. That is:

$$CP_{FPE} = CP_{\text{train}(N)} + N \left(CP_{\hat{y}} + 1 \right) + 12. \quad (6.91)$$

$CP_{\text{train}(N)}$ is the complexity associated with training on N examples, and $CP_{\hat{y}}$ denotes the complexity involved in calculating one sample of the filter output. Considering a 2-layer feed-forward perceptron neural network and the RGNB-algorithm for weight estimation then cf. Sec. 6.5.3:

$$CP_{FPE} \approx \frac{3}{2} N \text{itr} m^2 \quad (6.92)$$

where *itr* is the number of iterations, i.e., the number of training set replications.

The issue of designing a generalization error estimator for incomplete LX-models is discussed in [Kannurpatti & Hart 91]. The employed LX-model is:

$$y(k) = \mathbf{z}_m^\top(k) \mathbf{w}_m + e(k) \quad (6.93)$$

where

$$\mathbf{z}_m(k) = [z_{m,1}, z_{m,2}, \dots, z_{m,m}]^\top = \boldsymbol{\varphi}(x(k)) \quad (6.94)$$

is the m -dimensional preprocessed input vector signal with $\boldsymbol{\varphi}(\cdot)$ denoting an arbitrary linear or nonlinear mapping of the stochastic i.i.d. input signal $x(k)$. The inputs $z_{m,i}(k)$ are assumed to be orthonormal, i.e., $E_{x(k)} \left\{ \mathbf{z}_m(k) \mathbf{z}_m^\top(k) \right\} = \mathbf{I}$, where \mathbf{I} is the $m \times m$ identity matrix. Furthermore, \mathbf{w}_m denotes the m -dimensional weight vector. The output signal, $y(k)$, is supposed to be generated by the system

$$y(k) = \mathbf{z}_{m^\circ}^\top(k) \mathbf{w}_{m^\circ}^\circ + \varepsilon(k) \quad (6.95)$$

where $\mathbf{w}_{m^\circ}^\circ$ is the true weight vector with dimension m° . $\varepsilon(k)$ is the i.i.d. inherent noise with zero mean and variance σ_ε^2 which is independent of $x(k)$. It is assumed that

$$\mathbf{z}_m = \begin{cases} [z_{m^\circ,1}, z_{m^\circ,2}, \dots, z_{m^\circ,m}]^\top & , m < m^\circ \\ \mathbf{z}_{m^\circ} & , m = m^\circ \\ [(\mathbf{z}_{m^\circ})^\top, z_{m,m^\circ+1}, \dots, z_{m,m}]^\top & , m > m^\circ \end{cases}, \quad (6.96)$$

and obviously if $m \geq m^\circ$ the model is complete cf. Def. 6.3; otherwise, incomplete.

Introduce the splitting:

$$\mathbf{w}_{m^\circ} = \left[(\mathbf{w}_m^\circ)^\top, (\mathbf{w}_{m^\circ-m}^\circ)^\top \right]^\top, \quad m < m^\circ. \quad (6.97)$$

The result of [Kannurpatti & Hart 91] is that the estimate of the average generalization error – denoted J – becomes:

$$J = \begin{cases} FPE - \frac{2m}{N-m} |\mathbf{w}_{m^\circ-m}^\circ|^2, & m < m^\circ \\ FPE & m \geq m^\circ \end{cases}. \quad (6.98)$$

This result clearly shows the matter discussed in Sec. 6.4 that if the model is incomplete (i.e., $m < m^\circ$) then the average generalization error depends on the system. In this case through the vector: $\mathbf{w}_{m^\circ-m}^\circ$. Since $\mathbf{w}_{m^\circ-m}^\circ$ is unknown [Kannurpatti & Hart 91] suggests to neglect the term²⁸, $2m|\mathbf{w}_{m^\circ-m}^\circ|^2/(N-m)$, on the condition that N is large, i.e., $J = FPE, \forall m$. Hence, the estimator equals the FPE -estimator.

[Kannurpatti & Hart 91] shows that if (approximately for $N \rightarrow \infty$)

$$\min_{m < m^\circ} \left\{ \frac{|\mathbf{w}_{m^\circ-m}^\circ|^2}{m^\circ - m} \right\} \geq \frac{\sigma_\varepsilon^2}{N}, \quad (6.99)$$

then

$$E_{\mathcal{T}} \{FPE(m)\} \geq E_{\mathcal{T}} \{FPE(m^\circ)\} \quad (6.100)$$

where $FPE(m)$ denotes the FPE when using m weights and $E_{\mathcal{T}}\{\cdot\}$ denote the expectation w.r.t. the training set²⁹, \mathcal{T} . That is, on the average over all possible (large) training sets we expect that minimization of the FPE -estimate result in the correct model. This result does not necessarily imply that the FPE -estimator is consistent – in fact – it is not³⁰. This is due to the fact that the FPE -estimator is asymptotically equivalent to the AIC -estimator when the noise is Gaussian sa:aicgauss cf. [Ljung 87, Ch. 16]. [Kannurpatti & Hart 91] provides a further refinement since it is shown that $\lim_{N \rightarrow \infty} \text{Prob}\{\hat{m} < m^\circ\} = 0$ where $\hat{m} = \arg \min_m FPE(m)$. That is, the number of weights is never underestimated in the limit $N \rightarrow \infty$.

6.5.7 The Generalized Prediction Error Estimator

The Generalized Prediction Error Estimator (GPE) [Moody 91], [Moody 92] [Moody 94] can be viewed as an extension of the FPE -estimator in order to accomplish the following matters:

- NN-models, i.e., models which are nonlinear in the weights, e.g., MFPNN's.
- Incomplete models (denoted biased models in [Moody 92]).
- The use of a regularization term in the cost function.

²⁸Note that the term vanishes in the limit $N \rightarrow \infty$.

²⁹Note that $FPE = (N+m)S_N(\hat{\mathbf{w}})/(N-m)$ depends on the training set through the estimated weights, $\hat{\mathbf{w}}$.

³⁰That is, $\min_m FPE(m) \neq m^\circ, N \rightarrow \infty$.

The estimator of Γ , defined in sa:gamdef, becomes:

$$GPE = S_N(\hat{\mathbf{w}}) + \frac{2\hat{\sigma}_e^2 \hat{m}_{\text{eff}}}{N} \quad (6.101)$$

where

- $\hat{\mathbf{w}}$ are the estimated weights, i.e., $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} C_N(\mathbf{w})$. Recall from Ch. 5 that $C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa R_N(\mathbf{w})$, i.e., the sum of the LS cost function, $S_N(\mathbf{w})$, and the regularization term, $\kappa R_N(\mathbf{w})$ where κ is the regularization parameter.
- \hat{m}_{eff} is the effective number of weights given by:

$$\hat{m}_{\text{eff}} = \text{tr} \left[\left(\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (6.102)$$

where

- $\text{tr}[\cdot]$ is the trace operator.
- $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$ is the gradient vector of the filter output (see sa:model), i.e.,

$$\boldsymbol{\psi}(k; \hat{\mathbf{w}}) = \frac{\partial f(\mathbf{x}(k); \hat{\mathbf{w}})}{\partial \mathbf{w}}. \quad (6.103)$$

- $\mathbf{J}_N(\hat{\mathbf{w}})$ is the Hessian matrix of the cost function, i.e.,

$$\mathbf{J}_N(\hat{\mathbf{w}}) = \frac{1}{2} \frac{\partial^2 C_N(\hat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (6.104)$$

Note that \hat{m}_{eff} depends on κ through $\mathbf{J}_N(\hat{\mathbf{w}})$.

- $\hat{\sigma}_e^2$ is an estimate of the noise variance which according to [Moody 91] e.g., can be computed as:

$$\hat{\sigma}_e^2 = \frac{N}{N - \hat{m}_{\text{eff}}} S_N(\hat{\mathbf{w}}). \quad (6.105)$$

A detailed derivation of the *GPE*-estimator is not published yet; however, according to [Moody 92] the idea is to consider the inherent noise as a perturbation to an idealized model fit to noiseless data. The perturbation is taken as a second order Taylor series expansion in the inherent noise due to the constraint that the estimated weights minimize the perturbed cost function. Finally, based on this perturbation the average generalization error is calculated. A prerequisite is that the inherent noise is an i.i.d. sequence and independent of the input. Recall that the average generalization is defined as $\Gamma = E_{\mathcal{T}}\{E_{[\mathbf{x}_t; y_t]}\{e^2(k; \hat{\mathbf{w}})\}\}$, i.e., the expectation of the squared error w.r.t. both the independent test sample, $[\mathbf{x}_t; y_t]$ and the training set, \mathcal{T} . This involves the statistical fluctuations in the input \mathbf{x} and the inherent noise, ε of both the training set and the test sample. In [Moody 92] the p.d.f. of the input, $p(\mathbf{x})$, is approximated by the empirical distribution:

$$\hat{p}(\mathbf{x}) = \frac{1}{N} \sum_{k=1}^N \delta(\mathbf{x} - \mathbf{x}(k)) \quad (6.106)$$

where $\delta(\cdot)$ denotes the Dirac delta function and $\mathbf{x}(k)$, $k = 1, 2, \dots, N$ are the input samples contained in the training set. In consequence, the *GPE* only measures the effect of the

fluctuation in the inherent noise; the fluctuations in the input is not taken into account. This disadvantage is also noticed in [MacKay 92a].

Note that with the choice in sa:sighate a simple rewriting gives:

$$GPE = \frac{N + \hat{m}_{\text{eff}}}{N - \hat{m}_{\text{eff}}} S_N(\hat{\mathbf{w}}). \quad (6.107)$$

This formula displays the close connection with the *FPE*-estimator. If the model is an LX-model, i.e., linear in the weights, and regularization is not employed then $\hat{m}_{\text{eff}} = m$, where m is the number of weights, see [Moody 91] and Sec. 6.5.8. Consequently, the *GPE*-estimator coincides with the *FPE*-estimator.

6.5.8 The Generalization Error Estimator for Incomplete, Nonlinear Models

In this section a novel generalization error estimator for incomplete, nonlinear models (i.e., NN-models) (*GEN*) [Larsen 92] (a reprint is given in App. I) will be presented. In later work [Murata et al. 94] derived a similar estimator. The estimator can be viewed as an extension of the *FPE*- and the *GPE*-estimators. The *GEN*-estimator is encumbered with some advantages and drawbacks which – in a concise form – are:

Advantages:

- All data in the training set are used for estimating the weights. This is especially important in situations where training data are sparse. This is not the case when considering the cross-validation estimators.
- The model cf. sa:model may be an LX-model as well as an NN-model. Recall that the *GPE*-estimator also takes NN-models into account, and furthermore that the cross-validation estimators do not require any model assumptions.
- Both incomplete and complete models are treated. Recall that the incomplete models are the common case within neural network modeling tasks.
- The input may be correlated³¹ as well as white. In a signal processing context we often deal with correlated input.
- The inherent noise may be correlated and dependent on the input. This extends the usual assumption that the inherent noise is an i.i.d. sequence independent of the input.
- Noiseless systems are also considered.
- The weight estimate, $\hat{\mathbf{w}}$, is not required to be the global minimum of the cost function.
- It is ensured that the estimator becomes an estimator of the average generalization error, i.e., the estimate provides for fluctuations in both the input and the inherent noise and in that way it extends the *GPE*-estimator.
- The possibility of including regularization is treated (the *GPE*-estimator also deals with regularization).

³¹That is, $E\{\mathbf{x}(k)\mathbf{x}(k + \tau)\} \neq \mathbf{0}$.

- The same framework can be used for general type of cost functions.

Drawbacks:

- A fundamental prerequisite for the derivation is that the training set is large; however, most estimators in fact require this assumption.
- The model is assumed to be properly approximated by a linear expansion in the weights in the vicinity of $\hat{\mathbf{w}}$. The same assumption is made in connection with the *GPE*-estimator.
- The weight estimate, $\hat{\mathbf{w}}$, is assumed to be locally unique, i.e., the cost function has a curvature around $\hat{\mathbf{w}}$. This assumption also enters the derivation of the *GPE*-estimator.
- Only local effects due to fluctuations in the weight estimate are considered (see Sec. 6.5.8.3 for further details). This is also a prerequisite for the *GPE*-estimator – although it was not mentioned explicitly in [Moody 92].
- The effects of imperfect training³² are not accounted for. In [Hansen 93] this is done for a LL-model with Gaussian input.

6.5.8.1 Basic Assumptions

ASSUMPTION 6.1 *The data generating system is given by:*

$$y(k) = g(\mathbf{x}(k)) + \varepsilon(k) \quad (6.108)$$

and the nonlinear function $g(\cdot)$ is assumed time invariant³³.

ASSUMPTION 6.2 *The general model is given by:*

$$y(k) = f(\mathbf{x}; \mathbf{w}) + e(k; \mathbf{w}) \quad (6.109)$$

and the nonlinear function $f(\cdot)$ is assumed time invariant. \mathbf{w} denotes the m -dimensional weight vector and $e(k; \mathbf{w})$ the error signal.

The restriction of the general model to an *LX*-model results in:

$$y(k) = \mathbf{w}^\top \mathbf{z}(k) + e(k) \quad (6.110)$$

where $\mathbf{z} = \phi(\mathbf{x})$ and $\phi(\cdot)$ is an arbitrary time invariant linear or nonlinear mapping: $\mathbb{R}^L \mapsto \mathbb{R}^m$.

The models may be complete or incomplete according to Def. 6.3.

ASSUMPTION 6.3 *The input $\mathbf{x}(k)$ and the inherent noise $\varepsilon(k)$ are assumed to be strictly stationary sequences. Furthermore, $E\{\varepsilon(k)\} = 0$ and $E\{\varepsilon^2(k)\} = \sigma_\varepsilon^2$.*

However, also consider the following restriction of As. 6.3:

³²Recall cf. Ch. 5 that any training algorithm will have some amount of misadjustment, and typically not be able to reach the global minimum.

³³In fact the specific assumption on the data generating system is not required for deriving generalization error estimators; however, it is appealing the sake of clarity. Further, the strong motivation for choosing a model of form given in As. 6.2 is that the data is believed to be generated by `sa:datag`.

ASSUMPTION 6.4 The input $\mathbf{x}(k)$ is assumed to be a strictly stationary sequence and $\varepsilon(k)$ is a white³⁴, strictly stationary sequence with zero mean and variance σ_ε^2 . Furthermore, $\mathbf{x}(k)$ is assumed independent of $\varepsilon(k)$, $\forall k$.

ASSUMPTION 6.5 $\mathbf{x}(k)$ is assumed to be an M -dependent stationary sequence³⁵, i.e., $\mathbf{x}(k)$ and $\mathbf{x}(k + \tau)$ are independent $\forall |\tau| > M$. M is denoted the dependence lag.

Let Ω be a closed, bounded set (i.e., compact set) in weight space over which the cost function cf. pa:cstfun, $C_N(\mathbf{w})$, is minimized.

DEFINITION 6.4 Define \mathcal{W} as the set of weight vectors, $\hat{\mathbf{w}}$, which locally minimize $C_N(\mathbf{w})$, i.e., let $\dim(\boldsymbol{\theta}) = \dim(\mathbf{w})$ then

$$\mathcal{W} = \{\mathbf{w} \in \Omega \mid \exists \delta > 0, \forall \|\boldsymbol{\theta}\| < \delta, C_N(\mathbf{w} + \boldsymbol{\theta}) \geq C_N(\mathbf{w})\} \quad (6.111)$$

where $\|\cdot\|$ denotes a vector norm.

The cost function is restricted to comply with the following assumptions:

ASSUMPTION 6.6 Assume that there exists a covering of Ω in compact subsets³⁶:

$$\Omega = \bigcup_i \Omega(i) \quad (6.112)$$

such that $\hat{\mathbf{w}}(i) \in \Omega(i)$ uniquely minimizes $C_N(\mathbf{w})$ within the the partition $\Omega(i)$

ASSUMPTION 6.7 Assume that $\forall i$ ³⁷:

$$\frac{\partial C_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w}} = \mathbf{0}, \quad \mathbf{a}^\top \frac{\partial^2 C_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}. \quad (6.113)$$

Given As. 6.6 and 6.7, then \mathcal{W} according to Def. 6.4 is the countable (possibly infinite) set:

$$\mathcal{W} = \{\hat{\mathbf{w}}(i)\}. \quad (6.114)$$

A particular LS-estimator within the set \mathcal{W} is denoted $\hat{\mathbf{w}}$. Fig. 6.4 shows an example of a cost function in accordance with the above assumption. Similar to As. 6.6 and 6.7 we make the following assumptions:

³⁴By “white” is simply meant: $E\{\varepsilon(k)\varepsilon(k + \tau)\} \propto \delta(k)$.

³⁵A weaker assumption which may substitute As. 6.5 aims at assuming $\mathbf{x}(k)$ to be a strongly mixing sequence [Rosenblatt 85, p. 62]. That is, $\mathbf{x}(k)$, $\mathbf{x}(k + \tau)$ are asymptotically independent as $|\tau| \rightarrow \infty$. Formally,

$$\sup_{\mathbf{x}(k), \mathbf{x}(k+\tau)} |\text{Prob}\{\mathbf{x}(k)\mathbf{x}(k + \tau)\} - \text{Prob}\{\mathbf{x}(k)\}\text{Prob}\{\mathbf{x}(k + \tau)\}| \rightarrow \mathbf{0}, \quad \text{as } |\tau| \rightarrow \infty$$

where $\text{Prob}\{\cdot\}$ denotes probability.

³⁶Note that this assumption is irrelevant when dealing with LX-models.

³⁷Here and in the following, by definition

$$\left. \frac{\partial C_N(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\hat{\mathbf{w}}(i)} = \frac{\partial C_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w}}$$

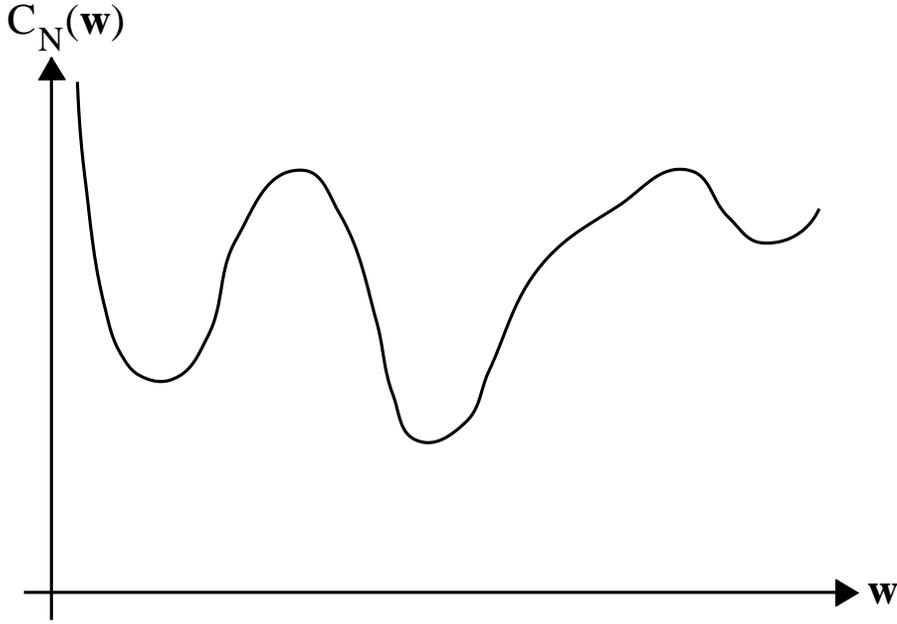


Figure 6.4: Shape of a cost function which meets Ass. 6.6 , 6.7. Note that all minima are unique and that the curvature exists (i.e., not equal to zero) near a minimum

ASSUMPTION 6.8 Assume that there exists a covering of Ω in compact subsets³⁸ (in general different from that in sa:lssplit):

$$\Omega = \bigcup_i \Omega^*(i) \quad (6.115)$$

such that $\mathbf{w}^*(i) \in \Omega^*(i)$ uniquely minimizes the expected cost function pa:expct , $C(\mathbf{w})$, within the the partition $\Omega^*(i)$

ASSUMPTION 6.9 Assume that $\forall i$:

$$\frac{\partial C(\mathbf{w}^*(i))}{\partial \mathbf{w}} = \mathbf{0}, \quad \mathbf{a}^\top \frac{\partial^2 C(\mathbf{w}^*(i))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}. \quad (6.116)$$

DEFINITION 6.5 Given As. 6.8 and 6.9, \mathcal{W}^* is defined as the countable (possibly infinite) set:

$$\mathcal{W}^* = \{\mathbf{w}^*(i)\}. \quad (6.117)$$

A particular optimal weight vector within the set \mathcal{W}^* is denoted \mathbf{w}^* . The recipe for deriving the GEN-estimate is a second order Taylor series expansion of the cost function, formally:

DEFINITION 6.6 Provided that As. 6.3 or As. 6.4 and As. 6.5 through As. 6.9 hold then GEN is defined as a consistent ($N \rightarrow \infty$) estimator of the average generalization error

³⁸Note that this assumption is irrelevant when dealing with LX-models.

sa:gamdef based on a second order Taylor series expansion of the cost function cf. As. 6.10 below.

The derivation of *GEN* depends on the chosen cost function. In this thesis we merely consider the the LS cost function with a regularization term which is independent of the training data, cf. Ch. 5, i.e.,

$$\begin{aligned} C_N(\mathbf{w}) &= \frac{1}{N} \sum_{k=1}^N c(k; \mathbf{w}) = S_N(\mathbf{w}) + \kappa R_N(\mathbf{w}) \\ &= S_N(\mathbf{w}) + \kappa r(\mathbf{w}) \end{aligned} \quad (6.118)$$

where

- $S_N(\mathbf{w})$ is the usual LS cost function,

$$S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N (y(k) - f(\mathbf{x}(k); \mathbf{w}))^2 \quad (6.119)$$

and the training data are given by: $\mathcal{T} = \{\mathbf{x}(k); y(k)\}$, $k = 1, 2, \dots, N$.

- $\kappa \geq 0$ is the regularization parameter. When $\kappa = 0$ the cost function passes into the conventional LS cost function.
- $r(\mathbf{w}) = R_N(\mathbf{w})$ where $r(\cdot)$ is assumed to be an arbitrary two times differentiable function which depends on the weight vector only.

In that case, the expected cost function, $C(\mathbf{w})$, is the sum of the generalization error and the regularization term, i.e.,

$$C(\mathbf{w}) = G(\mathbf{w}) + \kappa r(\mathbf{w}). \quad (6.120)$$

Recall the following definitions cf. Ch. 5:

DEFINITION 6.7

1. The instantaneous gradient vector of the output:

$$\boldsymbol{\psi}(k; \mathbf{w}) = \frac{\partial f(\mathbf{x}(k); \mathbf{w})}{\partial \mathbf{w}} \quad (6.121)$$

which is assumed to exist $\forall \mathbf{w} \in \Omega$.

2. The instantaneous second order derivative matrix of the output:

$$\boldsymbol{\Psi}(k; \mathbf{w}) = \frac{\partial \boldsymbol{\psi}(k; \mathbf{w})}{\partial \mathbf{w}^\top} \quad (6.122)$$

which is assumed to exist $\forall \mathbf{w} \in \Omega$. Note that $\boldsymbol{\Psi}$ is a symmetric matrix.

3. The Hessian matrix of the cost function:

$$\begin{aligned} \mathbf{J}_N(\mathbf{w}) &= \frac{1}{2} \frac{\partial^2 C_N(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\ &= \mathbf{H}_N(\mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\ &= \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}) \boldsymbol{\psi}^\top(k; \mathbf{w}) - \boldsymbol{\Psi}(k; \mathbf{w}) e(k; \mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \end{aligned} \quad (6.123)$$

4. The Hessian matrix of the expected cost function:

$$\begin{aligned}
\mathbf{J}(\mathbf{w}) &= \frac{1}{2} \frac{\partial^2 C(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\
&= \mathbf{H}(\mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\
&= E_{\mathbf{x}_t, \varepsilon_t} \left\{ \boldsymbol{\psi}_t(\mathbf{w}) \boldsymbol{\psi}_t^\top(\mathbf{w}) - \boldsymbol{\Psi}_t(\mathbf{w}) e_t(\mathbf{w}) \right\} + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (6.124)
\end{aligned}$$

The second order Taylor series expansion is formally expressed by the following assumption:

ASSUMPTION 6.10 *Let the minimization of C_N on the training set \mathcal{T} result in the estimate: $\hat{\mathbf{w}}$. Assume the existence of an optimal weight vector \mathbf{w}^* such that the remainders of the second order Taylor series expansion of C around \mathbf{w}^* are negligible. That is: Let $\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$ then*

$$C(\hat{\mathbf{w}}) \approx C(\mathbf{w}^*) + \Delta \mathbf{w}^\top \mathbf{J}(\mathbf{w}^*) \Delta \mathbf{w}, \quad (6.125)$$

as $\partial C(\mathbf{w}^*)/\partial \mathbf{w} = \mathbf{0}$ cf. As. 6.9. Further, let Ξ denote the hypersphere with centre in \mathbf{w}^* in which the second order expansion is valid (w.r.t. to prescribed bounds on the higher order derivatives). Further assume that the remainders of expanding C_N around $\hat{\mathbf{w}}$ to the second order is negligible, i.e.,

$$C_N(\mathbf{w}^*) \approx C_N(\hat{\mathbf{w}}) + \Delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w}, \quad (6.126)$$

as $\partial C_N(\hat{\mathbf{w}})/\partial \mathbf{w} = \mathbf{0}$ cf. As. 6.7.

Comments:

- The weight estimate, $\hat{\mathbf{w}}$, is highly dependent on the chosen weight estimation algorithm due to local optimization, initial conditions, etc. An alternative algorithm used on the same training set may therefore result in a different weight estimate. However, these facts are not crucial for the derivation of the *GEN*-estimate since it is the current weight estimate which *defines* the associated optimal weight vector, \mathbf{w}^* , and with that the hypersphere, Ξ . Consequently, we merely focus on weight estimate fluctuations which are kept within Ξ (see further Sec. 6.5.8.3).
- Due to the definitions of the cost function and the expected cost function sa:cost, (6.120) the Taylor series expansions in sa:Gtaylorr and (6.126) can be split into two parts, i.e.,

$$G(\hat{\mathbf{w}}) \approx G(\mathbf{w}^*) + \Delta \mathbf{w}^\top \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \quad (6.127)$$

$$r(\hat{\mathbf{w}}) \approx r(\mathbf{w}^*) + \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \Delta \mathbf{w} + \Delta \mathbf{w}^\top \frac{1}{2} \frac{\partial^2 r(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \Delta \mathbf{w}, \quad (6.128)$$

and

$$S_N(\mathbf{w}^*) \approx S_N(\hat{\mathbf{w}}) + \Delta \mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \quad (6.129)$$

$$r(\mathbf{w}^*) \approx r(\hat{\mathbf{w}}) - \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^\top} \Delta \mathbf{w} + \Delta \mathbf{w}^\top \frac{1}{2} \frac{\partial^2 r(\hat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \Delta \mathbf{w}. \quad (6.130)$$

Now, if the model is an LX-model the expansions sa:gexp, (6.129) are exact cf. Sec. 5.2. Furthermore, if the usual weight decay regularizer, $r(\mathbf{w}) = |\mathbf{w}|^2$, is employed second order expansions of $r(\cdot)$ (around any point) is exact.

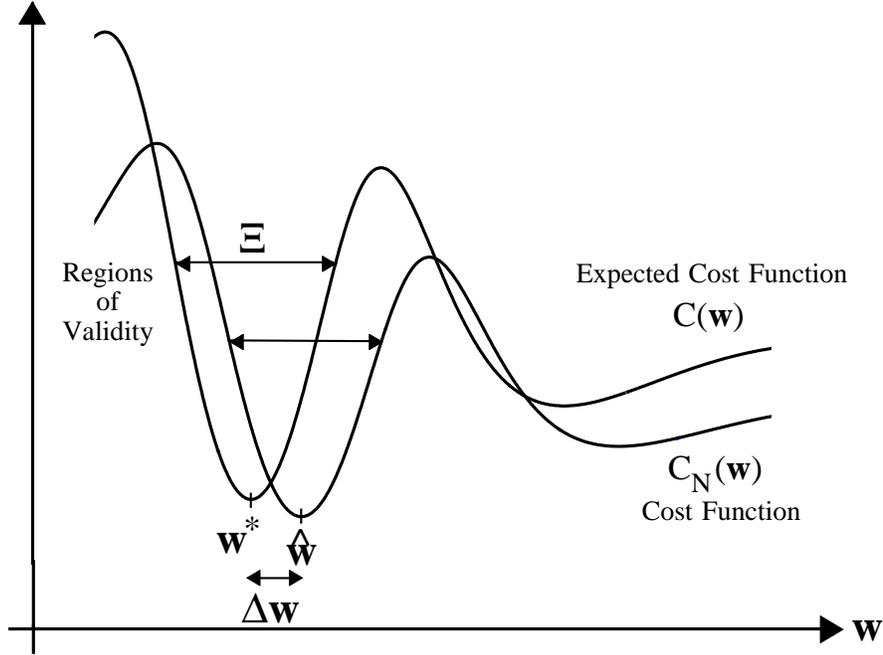


Figure 6.5: Example of cost function $C_N(\mathbf{w})$ and the expected cost function $C(\mathbf{w})$ which fulfill As. 6.10, q.e., that a proper expansion of both $C_N(\mathbf{w})$ and $C(\mathbf{w})$ around their respective minima to the second order is possible. The regions of validity (hyperspheres) for the second order expansion are shown. Notice, in particular, the region of validity around \mathbf{w}^* which is denoted by Ξ .

In Fig. 6.5 an example of cost functions which fulfill As. 6.10 is shown. To ensure the validity of the *GEN*-estimate we make the following additional assumption:

ASSUMPTION 6.11 *We assume large training sets, $N \rightarrow \infty$ and $N \gg 2M + 1$ where M is the dependence lag defined by As. 6.5. Further we assume that the dimension of the weight vector, m , is finite.*

6.5.8.2 Theorems

On the basis of these assumptions it is possible to derive a number of theorems stating the *GEN*-estimate under various conditions concerning the model, input, noise, and regularization. These assumptions are summarized in Table 6.1.

THEOREM 6.2 *Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an NN-model which is either **incomplete** or alternatively **complete** with the restriction that \mathbf{w}^* (defined in As. 6.10) is not the global optimum of $G(\mathbf{w})$. Further, suppose that As. 6.3, 6.5 – 6.11 hold. The *GEN*-estimate is then given*

Regularization	Model		Input	
			Independent	Dependent
Yes	No particular assumptions		Theorem 6.9	Theorem 6.8
No	Incomplete	NN-model	Theorem 6.4	Theorem 6.2
		LX-model	Theorem 6.5	Theorem 6.3
	Complete	NN-model	Theorem 6.6 (If \mathbf{w}^* is the global minimum)	
		LX-model	Theorem 6.6 Theorem 6.7 (If the input is Gaussian)	

Table 6.1: Table which shows the major conditions for employing the various theorems estimating the generalization error. Recall that the LX-model is linear in the parameters while the NN-model is nonlinear in the parameters.

by:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (6.131)$$

where $\text{tr}[\cdot]$ denotes the trace and the correlation matrices $\mathbf{R}(\tau)$, $0 \leq \tau \leq M$ are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}), \quad (6.132)$$

and the Hessian matrix $\mathbf{H}_N(\hat{\mathbf{w}})$ is cf. sa:jndef calculated as:

$$\mathbf{H}_N(\hat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \quad (6.133)$$

THEOREM 6.3 Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an **incomplete** LX-model cf. sa:linmod. Further, suppose that

As. 6.3, 6.5, 6.7, 6.9, and 6.11 hold. The GEN-estimate is then given by:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1} \right] \quad (6.134)$$

where the correlation matrices $\mathbf{R}(\tau)$, $0 \leq \tau \leq M$ are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \mathbf{z}(k) e(k; \hat{\mathbf{w}}) \mathbf{z}^\top(k + \tau) e(k + \tau; \hat{\mathbf{w}}), \quad (6.135)$$

and

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \quad (6.136)$$

where $\mathbf{z}(k)$ is defined by `sa:linmod`.

THEOREM 6.4 Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an NN-model which is either **incomplete** or alternatively **complete** with the restriction that \mathbf{w}^* (defined in As. 6.10) is not the global optimum of $G(\mathbf{w})$. Further, suppose that As. 6.4 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. 6.6 – 6.11 are supposed to hold. The GEN-estimate is then given by:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\mathbf{R}(0) \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (6.137)$$

where $\mathbf{H}(\hat{\mathbf{w}})$ is given by `sa:hndef`, and $\mathbf{R}(0)$ reads cf. `sa:R`:

$$\mathbf{R}(0) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) e^2(k; \hat{\mathbf{w}}). \quad (6.138)$$

THEOREM 6.5 Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an **incomplete** LX-model, cf. `sa:linmod`. Further, suppose that As. 6.4 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. 6.7, 6.9, and 6.11 are supposed to hold. The GEN-estimate then yields:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) e^2(k; \hat{\mathbf{w}}) \right) \left(\sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \right)^{-1} \right] \quad (6.139)$$

where $\mathbf{z}(k)$ is defined by `sa:linmod`.

THEOREM 6.6 Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be a **complete** NN- or LX-model. Further, suppose that As. 6.4 – 6.11 hold and that \mathbf{w}^* defined in As. 6.10 is the **global minimum**³⁹ of $G(\mathbf{w})$. Finally, let $E\{\varepsilon^2\} = \sigma_\varepsilon^2 \neq 0$. The GEN-estimate then coincides with the FPE-criterion cf. Sec. 6.5.6:

$$\text{GEN} = \text{FPE} = \frac{N+m}{N-m} S_N(\hat{\mathbf{w}}), \quad N > m. \quad (6.140)$$

³⁹Note the possibility of more coexisting global minima in which the expected cost (i.e., $G(\mathbf{w}^*)$) reach the same level. Further note that the requirement is trivially fulfilled when dealing with LX-models.

THEOREM 6.7 Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be a **complete** LX-model cf. sa:linmod. Further, suppose that As. 6.4, 6.7, 6.9 hold, the input vector is marginally Gaussian distributed with zero mean and positive definite covariance matrix \mathbf{H} , i.e., $\mathbf{z}(k) \in \mathcal{N}(\mathbf{0}, \mathbf{H})$, $\forall k$, and that $\mathbf{z}(k_1)$ is independent of $\mathbf{z}(k_2)$ as $k_1 \neq k_2$. Finally, let $E\{\varepsilon^2\} = \sigma_\varepsilon^2 \neq 0$. The GEN-estimate coincides with [Hansen 93] and is given by:

$$\text{GEN} = \frac{N(N-1)}{(N-m)(N-m-1)} S_N(\hat{\mathbf{w}}), \quad N > m+1. \quad (6.141)$$

In addition, the estimator is unbiased.

THEOREM 6.8 Suppose that As. 6.1, 6.2 hold and regularization is employed. The model may be an NN- or an LX-model, **complete** as well as **incomplete**. Further, suppose that As. 6.3, 6.5 – 6.11 hold. The GEN-estimate is then given by:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (6.142)$$

where $\text{tr}[\cdot]$ denotes the trace and the involved matrices are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k+\tau; \hat{\mathbf{w}}) e(k+\tau; \hat{\mathbf{w}}), \quad 0 \leq \tau \leq M, \quad (6.143)$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^\top}, \quad (6.144)$$

$$\mathbf{J}_N(\hat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) + \frac{\kappa}{2} \frac{\partial^2 r(\hat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (6.145)$$

If considering a LX-model then $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$ is replaced by $\mathbf{z}(k)$ defined in sa:linmod.

THEOREM 6.9 Suppose that As. 6.1, 6.2 hold and regularization is employed. The model may be an NN- or an LX-model, **complete** as well as **incomplete**. Further, suppose that As. 6.4 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. 6.6 – 6.11 are supposed to hold. The GEN-estimate is then given by:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[(\mathbf{S} + \mathbf{R}(0)) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right]. \quad (6.146)$$

The involved matrices are calculated as:

$$\mathbf{R}(0) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) e^2(k; \hat{\mathbf{w}}), \quad (6.147)$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^\top}, \quad (6.148)$$

$$\mathbf{J}_N(\hat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) + \frac{\kappa}{2} \frac{\partial^2 r(\hat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (6.149)$$

If considering a LX-model then $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$ is replaced by $\mathbf{z}(k)$ defined in sa:linmod.

6.5.8.3 Sketch of Proofs and Discussion

The complete proofs of the above theorems are given in App. A. Here only a sketch of the proofs will be provided.

The basis is the Taylor series expansions given in As. 6.10. Taking the expectation w.r.t. the training set, \mathcal{T} , of the Taylor series expansions sa:Gtaylorr and (6.126) yield:

$$E_{\mathcal{T}} \{C_N(\mathbf{w}^*)\} \approx E_{\mathcal{T}} \{C_N(\hat{\mathbf{w}})\} + E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} \quad (6.150)$$

$$E_{\mathcal{T}} \{C(\hat{\mathbf{w}})\} \approx E_{\mathcal{T}} \{C(\mathbf{w}^*)\} + E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}(\mathbf{w}^*) \Delta \mathbf{w} \right\} \quad (6.151)$$

where $\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$. The average generalization error – which we want to estimate – cf. sa:gamdef and sa:expcost becomes:

$$\Gamma = E_{\mathcal{T}} \{G(\hat{\mathbf{w}})\} = E_{\mathcal{T}} \{C(\hat{\mathbf{w}})\} - \kappa E_{\mathcal{T}} \{r(\hat{\mathbf{w}})\}. \quad (6.152)$$

Recall that the weight estimate, $\hat{\mathbf{w}}$ obtained by training on the present training set defines an associated \mathbf{w}^* through the required validity of the Taylor series expansion concerning $C_N(\cdot)$ cf. sa:Staylorr. Furthermore, recollect that Ξ is the hypersphere with centre \mathbf{w}^* and radius $\Delta \mathbf{w}$, i.e., the region in weight-space in which the expansion of $C(\cdot)$ is valid cf. sa:Gtaylorr. Now consider all possible training sets of size N and the corresponding sets of minimizers, \mathcal{W} . In general only a *finite number of training sets* will result in sets, \mathcal{W} , which contain an estimate $\hat{\mathbf{w}}$ which is in Ξ . Consequently, the above averaging – formally considered – is done w.r.t. training sets which result in sets, \mathcal{W} , containing a $\hat{\mathbf{w}} \in \Xi$. However⁴⁰, $C_N \rightarrow C$ in the limit $N \rightarrow \infty$. Consequently, in this limit, every training set contains a $\hat{\mathbf{w}}$ which is in Ξ .

Since \mathbf{w}^* does not dependent on the training set then

$$E_{\mathcal{T}} \{C_N(\mathbf{w}^*)\} = \frac{1}{N} \sum_{k=1}^N E_{\mathcal{T}} \{c(k; \mathbf{w}^*)\} = C(\mathbf{w}^*) = E_{\mathcal{T}} \{C(\mathbf{w}^*)\} \quad (6.153)$$

where $c(k; \mathbf{w}^*) = e^2(k; \mathbf{w}) + \kappa r(\mathbf{w})$. This equality forms the trick which allow us to substitute sa:esr into sa:egr. Consequently, using sa:gamdefr and the relation

$$E_{\mathcal{T}} \{C_N(\hat{\mathbf{w}})\} = E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + E_{\mathcal{T}} \{r(\hat{\mathbf{w}})\} \kappa \quad (6.154)$$

results in:

$$\Gamma \approx E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} + E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}(\mathbf{w}^*) \Delta \mathbf{w} \right\}. \quad (6.155)$$

Note that $\Gamma \geq E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\}$ due to the fact that both $\mathbf{J}_N(\hat{\mathbf{w}})$ and $\mathbf{J}(\mathbf{w}^*)$ are positive definite matrices. This is in contrast to the fact that $G(\hat{\mathbf{w}})$ is greater than $S_N(\hat{\mathbf{w}})$ with some probability less than one, cf. Ex. 6.4.

A suitable rewriting of sa:lamp is done by using the rule of matrix algebra: $\text{tr}[\mathbf{A}\mathbf{B}] = \text{tr}[\mathbf{B}\mathbf{A}]$ where $\mathbf{A} \in \mathbb{R}^{p \times q}$, $\mathbf{B} \in \mathbb{R}^{q \times p}$ are arbitrary matrices and $\text{tr}[\cdot]$ is the trace operator. That is:

$$\Gamma \approx E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + \text{tr} \left[E_{\mathcal{T}} \left\{ \mathbf{J}_N \Delta \mathbf{w} \Delta \mathbf{w}^\top \right\} \right] + \text{tr} \left[E_{\mathcal{T}} \left\{ \mathbf{J} \Delta \mathbf{w} \Delta \mathbf{w}^\top \right\} \right]. \quad (6.156)$$

⁴⁰This limit transition is ensured if $c(k; \mathbf{w}) = e^2(k; \mathbf{w}) + \kappa r(\mathbf{w})$ is a mean-ergodic sequence according to [Papoulis 84a, Ch. 9-5].

The first term is estimated by:

$$E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} \approx S_N(\hat{\mathbf{w}}) \quad (6.157)$$

since only one training set is available. In order to evaluate the second and third term an expression of $\Delta \mathbf{w}$ is required. As $\hat{\mathbf{w}}$ minimizes $C_N(\mathbf{w})$

$$\frac{\partial C_N(\hat{\mathbf{w}})}{\partial \mathbf{w}} = \mathbf{0}. \quad (6.158)$$

Now application of the mean value theorem yields cf. App. A

$$\begin{aligned} \Delta \mathbf{w} &\approx - \left[\frac{\partial^2 C_N(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right]^{-1} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \\ &= \mathbf{J}_N^{-1}(\mathbf{w}^*) \left(\frac{1}{N} \sum_{k=1}^N \psi(k; \mathbf{w}^*) e(k; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right). \end{aligned} \quad (6.159)$$

That is, the matrix $\Delta \mathbf{w} \Delta \mathbf{w}^\top$ becomes:

$$\begin{aligned} \Delta \mathbf{w} \Delta \mathbf{w}^\top &= \mathbf{J}_N^{-1}(\mathbf{w}^*) \cdot \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \left(\psi(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \cdot \\ &\quad \left(\psi^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \right) \cdot \mathbf{J}_N^{-1}(\mathbf{w}^*). \end{aligned} \quad (6.160)$$

According to sa:lam the second and the third term result from pre-multiplying $\Delta \mathbf{w} \Delta \mathbf{w}^\top$ with $\mathbf{J}_N(\mathbf{w}^*)$ and $\mathbf{J}(\mathbf{w}^*)$, respectively, and subsequently performing the expectation w.r.t. \mathcal{T} . In that context certain facts should be emphasized:

Approximation of the Inverse Hessian The first matter which complicates the expectation is the inverse Hessian matrix $\mathbf{J}_N^{-1}(\mathbf{w}^*)$. In order to manage this difficulty a series expansion of the inverse Hessian is suggested⁴¹ cf. Co. B.1. The details are given in App. B. If all terms which tend to zero faster than $(2M+1)/N$ as $N \rightarrow \infty$ are neglected⁴² then

$$\mathbf{J}_N^{-1}(\mathbf{w}^*) \approx \mathbf{J}^{-1}(\mathbf{w}^*), \quad N \gg 2M + 1. \quad (6.161)$$

This approximation has the obvious advantage that $\mathbf{J}^{-1}(\mathbf{w}^*)$ does not depend on the training set; consequently, it goes outside the expectation operator. In theory more terms of the series expansion can be included; however, a tremendous increase in the computational complexity results, thereby making the final estimator unattractive.

In App. B a few numerical experiments are given in order to examine the quality of the approximation. First the impact of the condition number of the Hessian on the approximation is studied. The results indicate that the requisite N grows with the condition number. Secondly, the effect of (time) correlation in the input vector is investigated. It turns out that if the assumption $N \gg 2M + 1$ is not met then the approximation may be rather poor. Consequently, the stronger correlation (i.e., the larger M) in the input the larger N is required.

⁴¹Even in the most plain case, viz. LX-models, the series expansion is indispensable unless the input is a zero mean Gaussian distributed vector.

⁴²That is, terms proportional to $1/N^q$, $q \geq 2$. Furthermore, recall that M is the dependence lag, i.e., $\mathbf{x}(k)$, $\mathbf{x}(k + \tau)$ are independent as $|\tau| > M$.

Performing Expectations Since the approximated inverse Hessians go outside the expectation operator the essential term to handle is (cf. sa:lam, (6.160))

$$E_{\mathcal{T}} \left\{ \left(\boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \left(\boldsymbol{\psi}(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \right) \right\}. \quad (6.162)$$

The expectation w.r.t. the training set implies expectation concerning the joint p.d.f. of the input and the output, or equivalently the joint p.d.f. of the input and the inherent noise. The instantaneous gradient vector, $\boldsymbol{\psi}(k; \mathbf{w}^*)$, depends on the input only. However, the optimal error, $e(k; \mathbf{w}^*)$, depends in general both on the input and on the inherent noise. This is seen by using the equations defining the system and the model, cf. As. 6.1, 6.2, (consult also Sec. 6.3.4)

$$e(k; \mathbf{w}^*) = \underbrace{\varepsilon(k)}_{\text{Inherent Noise}} + \underbrace{g(\mathbf{x}(k)) - f(\mathbf{x}(k); \mathbf{w}^*)}_{\text{Model Error}}. \quad (6.163)$$

In consequence, $\boldsymbol{\psi}(k; \mathbf{w}^*)$ and $e(k; \mathbf{w}^*)$ are dependent quantities. However, if the model is complete, no regularization is employed, and \mathbf{w}^* is the global minimum then the model error vanishes cf. Sec. 6.3.4. Consequently,

$$e(k; \mathbf{w}^*) = \varepsilon(k). \quad (6.164)$$

The dependence among the instantaneous gradient vector and the optimal error is thus an immediate consequence of dealing with incomplete models and employing regularization and should be taken into account. This fact has been pointed out in the statistical literature [White 81]. Within the *GPE*-estimator [Moody 92] – which is intended to deal with regularized, incomplete models – this issue is never dealt with. This is due to the fact that statistical fluctuations in the input are not considered (i.e., the input is actually not regarded as being stochastic). In Sec. A.2.2.1 it is shown that only if the regularization parameter, κ , is small and the model is complete then the *GEN*-estimate coincides with the *GPE*-estimator.

Now, if the input and the inherent noise are independent, we deal with a complete model, and no regularization is employed; consequently, the expectation w.r.t. the training set can be calculated by using the relation:

$$E_{\mathcal{T}}\{\cdot\} = E_{\{\mathbf{x}(k), \varepsilon(k)\}}\{\cdot\} = E_{\{\mathbf{x}(k)\}} \left\{ E_{\{\varepsilon(k)\}}\{\cdot | \{\mathbf{x}(k)\}\} \right\} \quad (6.165)$$

where $\{\cdot\}$ denotes a set of N samples and $E\{\cdot | \cdot\}$ denotes a conditional expectation. That is, the expectation is done in two steps and leads to simple formulas.

In general cogent evaluation of sa:expect is impossible without knowledge of both p.d.f.'s and the true system. Therefore an approximate result is obtained by replacing the ensemble-averaging with a time-averaging (i.e., mean-ergodicity is assumed). This arrangement involves a statistical error which – in total – scales like $o(Mm^2/N)$ (see App. A) where m is the number of weights.

Finally, the general form of the *GEN*-estimate is cf. sa:lam, (6.157) and the results of App. A

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} \left(\mathbf{R}(\tau) + \mathbf{R}^\top(\tau) \right) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (6.166)$$

where

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}), \quad 0 \leq \tau \leq M \quad (6.167)$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^\top}. \quad (6.168)$$

The *GEN*-estimate is thus a sum of the training cost and an extra term which is an estimate of $2\text{tr}[\mathbf{J}(\mathbf{w}^*)\mathbf{V}]$ where \mathbf{V} is the variance matrix of the estimated weights. That is, this term effectively measures the expected increase in the training cost due to fluctuations in the estimated weights.

6.5.8.4 Summary of the Essential Assumptions

In summary the essential assumptions done in order to derive the *GEN*-estimate are:

- A second order Taylor series expansion of the cost function and the expected cost function (cf. sa:esr and sa:egr).
- Averaging over all possible training sets, \mathcal{T} , is approximated by averaging w.r.t. the training sets which contain a (local) minimum, $\hat{\mathbf{w}}$, which lies within the hypersphere, Ξ . Recall that Ξ defines the region in weight-space where the second order Taylor series expansion of the expected cost function is suitable.
- The average training cost $E_{\mathcal{T}}(S_N(\hat{\mathbf{w}}))$ is approximated by the only available value, viz. $S_N(\hat{\mathbf{w}})$, cf. sa:snwhatst.
- The inverse Hessian matrices are approximated by using a series expansion. This results in the requirement: $N \gg 2M + 1$ cf. App. B. What is meant by the operator \gg is highly dependent on the structure of the Hessian. Numerical examples (see App. B) indicate that the required N increases with increasing condition number of the Hessian matrix.
- Ensemble-averages are substituted by time-averages. This implies a statistical error which scales like Mm^2/N , cf. App. A.
- The nature of the true data generating system is such that certain higher order moments exists (i.e., are limited).

Theorems stating statistical estimators are normally attended by some measures of the quality of the estimator, e.g., bias and variance. In principle, it is possible to derive formulas of bias and variance of the *GEN*-estimator; however, they will not provide much insight since they depend on p.d.f.'s of the input and the inherent noise along with the structure of the system (i.e., $g(\mathbf{x})$). Consequently, they can only be calculated in some simple cases, and even then; obtaining an exact expression may require very extensive algebra. The only general statements concerning the quality of the *GEN*-estimator is that it is consistent and biased. The biasness is, among other things, due to the fact that the employed inverse Hessian approximation is biased.

In Ch. 7 a numerical study of the quality of the *GEN*-estimator is provided.

6.5.8.5 Computational Complexity

In preparation for comparisons of the computational complexity⁴³ of the *GEN*-estimator is provided the following:

The First Term According to sa:genfin the first term is the computational complexity of calculating the training cost, CP_{S_N} , which equals (see also e.g., Sec. 6.5.3)

$$CP_{S_N} = CP_{\text{train}(N)} + N \left(CP_{\hat{y}} + 1 \right) + 10 \quad (6.169)$$

where $CP_{\text{train}(N)}$ is the complexity involved in training on N examples (i.e., the estimation of the weights) and $CP_{\hat{y}}$ is the complexity of calculating one sample of the filter output, \hat{y} . The quantity $1/N$, which was used above, is stored for future use. Using the RGNB-algorithm and employing a 2-layer MFPNN gives cf. Sec. 6.5.3

$$CP_{S_N} \approx \frac{3}{2} N \text{itr} m^2 \quad (6.170)$$

where *itr* is the number of iterations, i.e., the number of training set replications.

The Second Term The second term of sa:genfin is evaluated in several steps⁴⁴. First consider the terms in the round bracket:

- Consider, $CP_{N\mathbf{R}(\tau)}$, the complexity of calculating $N\mathbf{R}(\tau)$ cf. sa:rtaufin. The instantaneous gradient, $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$, is assumed to be calculated within the training process. That is, for $1 \leq \tau \leq M$ we get:

$$CP_{N\mathbf{R}(\tau)} = (N - \tau)(m^2 + 2m), \quad 1 \leq \tau \leq M. \quad (6.171)$$

Whereas, when $\tau = 0$ the symmetry of $\mathbf{R}(0)$ is exploited, which means:

$$CP_{N\mathbf{R}(0)} = N \left(\frac{m^2 + m}{2} + m \right) = \frac{N}{2}(m^2 + 3m). \quad (6.172)$$

- The complexity of calculating $N\mathbf{S}$ is cf. sa:sfin

$$CP_{N\mathbf{S}} = m + 2, \quad (6.173)$$

as we assume that $-\kappa^2/4$ is stored as a separate quantity.

The total complexity within the round brackets, $CP_{()}$, thus equals

$$\begin{aligned} CP_{()} &= CP_{N\mathbf{S}} + CP_{N\mathbf{R}(0)} + \sum_{\tau=1}^M 2 + CP_{N\mathbf{R}(\tau)} \\ &= m + 2 + \frac{N}{2}(m^2 + 3m) + 2M + \left(NM - \frac{M}{2}(M + 1) \right) (m^2 + 2m) \\ &= \frac{m^2}{2}(M(2N - M - 1) + N) + m \left(M(2N - M - 1) + \frac{3N}{2} + 1 \right) + 2M + 2. \end{aligned} \quad (6.174)$$

⁴³Recall that the computational complexity is measured as the number of multiplications and divisions. One division equals 10 multiplications cf. Ch. 5.

⁴⁴In order to avoid needless divisions the terms within the round brackets are multiplied with N . Furthermore, the inverse Hessian is multiplied with N^{-1} .

As N normally is somewhat larger than M and m we may use the approximation

$$CP_{()} \approx NMm^2. \quad (6.175)$$

The inverse Hessian, $\mathbf{J}_N^{-1}(\hat{\mathbf{w}})$ may be obtained differently:

- If a second order algorithm is used for training (e.g., the RGNB-algorithm Ch. 5) and a pseudo Hessian approximation is used⁴⁵ then an approximation⁴⁶ of the inverse Hessian results directly from the training process. However, if the RGNB-algorithm is used we have to form the matrix product $\mathbf{U}\mathbf{D}\mathbf{U}^\top$ at the terminal stage. Since \mathbf{U} is an $m \times m$ unit upper triangular matrix and \mathbf{D} an $m \times m$ diagonal matrix the matrix product involves $(m^3 - m)/3$ multiplications.
- If the full Hessian is employed then this cf. sa:jndef involves the complexity⁴⁷:

$$\begin{aligned} CP_{\mathbf{J}} &= \underbrace{N \frac{m^2 + m}{2}}_{\psi(k; \mathbf{w})\psi^\top(k; \mathbf{w})} + N \underbrace{\left(CP_{\Psi} + \frac{m^2 + m}{2} \right)}_{\Psi(k; \mathbf{w})e(k; \mathbf{w})} \\ &= N \left(m^2 + m + CP_{\Psi} \right) \end{aligned} \quad (6.176)$$

where CP_{Ψ} is the complexity of calculating $\Psi(k; \hat{\mathbf{w}})$. Furthermore, the inverse has to be calculated. Using Gauss-elimination this involves a complexity proportional to m^3 . The total amount is thus approximately: Nm^2 .

Finally, the second term of sa:genfin results from calculating the trace of the product of the matrix in the round brackets and the inverse Hessian; i.e., it is only necessary to compute the diagonal. This involves m^2 multiplications. In addition, one multiplication is spent on multiplying with $2/N$.

To sum up the individual contributions, the complexity of the *GEN*-estimate employing a RGNB-algorithm and 2-layer MFPNN is approximately:

$$CP_{GEN} \approx \frac{3}{2}N itr m^2 + NMm^2 + Nm^2 = Nm^2 \left(\frac{3}{2} itr + M + 1 \right). \quad (6.177)$$

6.5.8.6 Additional Theorem Concerning Complete LX-Models

In connection with complete LX-models and no regularization it is possible to elaborate on properties of the average generalization error.

THEOREM 6.10 *Suppose that the system given by sa:nonsys, the model sa:model is a **complete** (see Def. 6.3) LX-model cf. sa:linmod and no regularization is employed. Furthermore, suppose that As. 6.4 – 6.2, 6.7, 6.9 hold. The average generalization error, Γ , complies with the following ranking:*

$$\Gamma > \frac{N + m}{N - m} E_{\mathcal{T}}\{S_N(\hat{\mathbf{w}})\} > \left(1 + 2\frac{m}{N} \right) E_{\mathcal{T}}\{S_N(\hat{\mathbf{w}})\}. \quad (6.178)$$

⁴⁵That is, the term $-\sum_{k=1}^N \Psi(k; \mathbf{w})e(k; \mathbf{w})$ cf. Def. 6.7 and Ch. 5 is neglected.

⁴⁶The result is merely approximate due to the fact that the inverse pseudo Hessian is iteratively constructed on different weight estimates. In principle, the weights should be kept at $\hat{\mathbf{w}}$ and subsequently the iterative process had to be run once more.

⁴⁷Here we assume that the standard weight decay regularization is used, i.e., $\partial r(\mathbf{w})/\partial \mathbf{w} \partial \mathbf{w}^\top = \mathbf{I}$.

From this theorem it is clear that on the average the FPE -estimate (i.e., $E_{\mathcal{T}}\{FPE\}$) is an underestimate of the average generalization error, Γ . On the other hand, on the average the FPE -estimate is larger than the estimate, $(1 + 2m/N)S_N(\hat{\mathbf{w}})$, i.e., closer to Γ . This estimator is known as the *predicted squared error* (PSE) estimator [Barron 84].

Notice, that the general form of the GEN -estimator resembles the PSE-estimator like that; it is a sum of the training cost plus $2/N$ times a penalty term. If the complete model (either LX- or NN-models) assumption is exploited and a consistent expansion is done in terms of m/N then the GEN -estimator in fact coincides with the PSE-estimator. However, the ranking in Th. 6.10 shows that for LX-models it is not profitable to make a consistent expansion in m/N . We suggest using the FPE -estimator for complete NN-models also (cf. Th. 6.6), even though it has not been possible to show that the FPE -estimate underestimates Γ on the average.

6.6 Statistical Methods for Improving Generalization

As mentioned in Sec. 6.4 it is possible to predict if the average generalization error decreases when imposing some constraints on the weights and thereby eliminating degrees of freedom in the model. In this section we present a statistical framework for predicting if some constraints among the weights exist.

6.6.1 Linear Hypotheses

Consider the general linear hypothesis, \mathcal{H} , and the alternative \mathcal{H}_a given by:

$$\begin{aligned}\mathcal{H} : \quad \Xi \mathbf{w}^* &= \mathbf{a} \\ \mathcal{H}_a : \quad \Xi \mathbf{w}^* &\neq \mathbf{a}\end{aligned}\tag{6.179}$$

where Ξ is an $n \times m$ dimensional *restriction matrix* with rank n , \mathbf{a} is an n -dimensional vector, and \mathbf{w}^* is the m -dimensional optimal weight vector. The hypothesis thus impose $n < m$ *linear* restrictions⁴⁸ on the optimal weights. That is, the question is: Do the optimal weights lie in an $(m - n)$ -dimensional subspace.

An impending issue is the choice of Ξ and \mathbf{a} . It is important to emphasize that the hypothesis naturally has to be formulated a priori, i.e., the training data *may not* be used to estimate a hypothesis; otherwise, the result of testing the hypothesis may be misleading. In some cases the nature of the task under consideration may guide the formulation of plausible hypotheses. However, in general, we may stay content with the hypothesis that some of the weights equal zero. This hypothesis is motivated by the following facts:

- Consider a filter architecture with a relatively large number of weights. Since the system $g(\mathbf{x})$ is supposed to be unknown we really perform a “black-box” modeling. Hence, it seems reasonable that some of these weights only provide an inferior contribution to the explanation of $g(\mathbf{x})$ and may therefore better be excluded.
- The deletion of weights is a convenient hypothesis since it results in a reduction of the computational complexity.

⁴⁸It is also possible to formulate nonlinear hypotheses (e.g., [Seber & Wild 89, Ch. 5.9]); however, this topic is not further elaborated.

On the other hand, one may argue that the case of an optimal weight exactly equal to zero is rather improbable. However, due to the fact that the number of training examples is limited the fluctuation in the weight estimate implies that it may be indistinguishable from zero with a certain confidence, i.e., it is evident that the optimal weight value equals zero. Paraphrased, if a test of the hypothesis (see the details below) results in accept then it should be fixed at zero. If the size of the training set were increased it would possibly be realized that the concerned weight better is kept in the model.

In the case of deleting weights, $\mathbf{a} = \mathbf{0}$, and the i 'th row of $\mathbf{\Xi}$ contains a one in position j if we believe that the j 'th weight is zero; otherwise, zeros. For instance, suppose that $m = 6$ and we test if the weights # 1, 4, 5 can be removed then the restriction matrix is given by⁴⁹:

$$\mathbf{\Xi} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \quad (6.180)$$

A severe problem is the existence of $2^m - 1$ possible hypotheses⁵⁰ since the number usually is astronomic. Consequently, algorithms for selecting promising subsets are required. This is the subject of Sec. 6.7.

6.6.2 Hypothesis Testing

In order to test the hypothesis sa:hypo we will apply a special case of the so-called Wald test statistic (e.g., [Seber & Wild 89, Ch. 5.3 & 5.9], [White 87]) which is given by:

$$T = (\mathbf{\Xi}\hat{\mathbf{w}} - \mathbf{a})^\top (\mathbf{\Xi}\mathbf{V}\mathbf{\Xi}^\top)^{-1} (\mathbf{\Xi}\hat{\mathbf{w}} - \mathbf{a}) \quad (6.181)$$

where $\hat{\mathbf{w}}$ is the estimated weight vector and \mathbf{V} is the weight covariance matrix $\mathbf{V} = E_{\mathcal{T}}\{\Delta\mathbf{w}\Delta\mathbf{w}^\top\}$ with $\Delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$. Note that $\mathbf{\Xi}\mathbf{V}\mathbf{\Xi}^\top$ is an $n \times n$ matrix⁵¹, i.e., as n normally is small compared to m then the inversion of this matrix is not computational extensive. Note that in the case of testing the hypothesis of superfluous weights the matrix product $\mathbf{\Xi}\mathbf{V}\mathbf{\Xi}^\top$ simply pick out the rows and columns of \mathbf{V} corresponding to weights which are supposed to be zero. Consequently, the matrix product is never implemented directly in practice; however, it is convenient w.r.t. theoretical considerations.

Under the assumption that $\hat{\mathbf{w}}$ is Gaussian distributed with mean \mathbf{w}^* and covariance matrix \mathbf{V} then the test statistic follows a chi-square distribution with n degrees of freedom, $\chi^2(n)$. In the section below this assumption along with the calculation of \mathbf{V} is treated.

Let α denote the significance level⁵² and $\chi_{1-\alpha}^2(n)$ the $1 - \alpha$ fractile of the $\chi^2(n)$ -distribution then the hypothesis is rejected if

$$T > \chi_{1-\alpha}^2(n). \quad (6.182)$$

This is also denoted the reject region. Note that the test statistic sa:teststat is a quadratic form. In order to interpret the test statistic consider the simple hypothesis $w^* = 0$ where w is one of the weights. In that case the test statistic becomes: $T = \hat{w}^2/V\{\hat{w}\}$. That is, it measures the square value of the estimate relative to the variance. Of course, if the

⁴⁹Note that the rows can be interchanged arbitrarily.

⁵⁰This is due to the fact that any of the m weights may be considered as zero or not. Furthermore, at least one weight is supposed to be zero.

⁵¹This matrix is invertible (i.e., the rank equals n) since the rank of $\mathbf{\Xi}$ is assumed to be n .

⁵²That is, the probability of rejecting a true hypothesis.

estimated weight value is small we then accept the hypothesis of the weight being equal to zero.

A number of alternative test statistics is available cf. [Seber & Wild 89, Ch. 5.3 & 5.9]. They are all asymptotic equivalent as $N \rightarrow \infty$; however, the Wald statistic may be less powerful⁵³. On the other hand, two immediate reasons for choosing the Wald statistic are:

1. Different hypotheses can be tested without expensive retraining, q.e., only the original (unconstrained) estimate $\hat{\mathbf{w}}$ is required. The majority of other methods require a constrained weight estimate, i.e., the estimate of the weights under the hypothesis. However, note that after accepting a particular hypothesis the weights naturally have to be reestimated in order to improve generalization ability. This is elaborated in Sec. 6.6.5.
2. The alternative methods are not directly applicable to deal with the case of incomplete models.

6.6.3 Asymptotic Distribution of the Weight Estimate

The basic of deriving an asymptotic distribution is the expression of $\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$ cf. sa:dwr The expression is essentially due to a second order expansion of the cost function cf. As. 6.10. Consequently, two important assumptions are required: First, the inverse Hessian matrix, \mathbf{J}_N^{-1} , has to exist under the hypothesis. Secondly, the second order cost function expansion is supposed to hold under the hypothesis. This is illustrated by the following example:

EXAMPLE 6.5

Consider a simple hypothesis involving one of the weights: $\mathcal{H} : w^* = a$. The hypothesis is then tested on the basis of a second cost function approximation, say $\hat{C}_N(\mathbf{w})$. An impending issue is the quality of the test. As we shall see below the variance of $\hat{\mathbf{w}}$ is in principle inversely proportional to the curvature of the cost function around $\hat{\mathbf{w}}$. Let us focus on a particular case shown in Fig. 6.6. It is seen that $\hat{C}_N(\mathbf{w})$ has a higher curvature than the “true” cost function $C_N(\mathbf{w})$. We then expect that the weight variance is underestimated. That is, cf. sa:teststat, (6.182), the test based on $\hat{C}_N(\mathbf{w})$ has a smaller acceptance range in terms of the squared deviation, $(\hat{\mathbf{w}} - a)^2$. Consequently, if the test is rejected in the light of $\hat{C}_N(\mathbf{w})$ it is also rejected when using $C_N(\mathbf{w})$. On the other hand, this also means that a true hypothesis may not be found.

Obviously the opposite case also exists, i.e., $\hat{C}_N(\mathbf{w})$ has a lower curvature than $C_N(\mathbf{w})$. In that case, a hypothesis may be wrongly accepted.

A method for checking the validity of the cost function approximation is described in Sec. 6.6.5. □

According to sa:dwr:

$$\begin{aligned} \Delta \mathbf{w} &\approx - \left[\frac{\partial^2 C_N(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right]^{-1} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \\ &= \mathbf{J}_N^{-1}(\mathbf{w}^*) \left(\frac{1}{N} \sum_{k=1}^N \psi(k; \mathbf{w}^*) e(k; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right). \end{aligned} \quad (6.183)$$

⁵³That is, the probability that the hypothesis is rejected under the alternative is smaller.

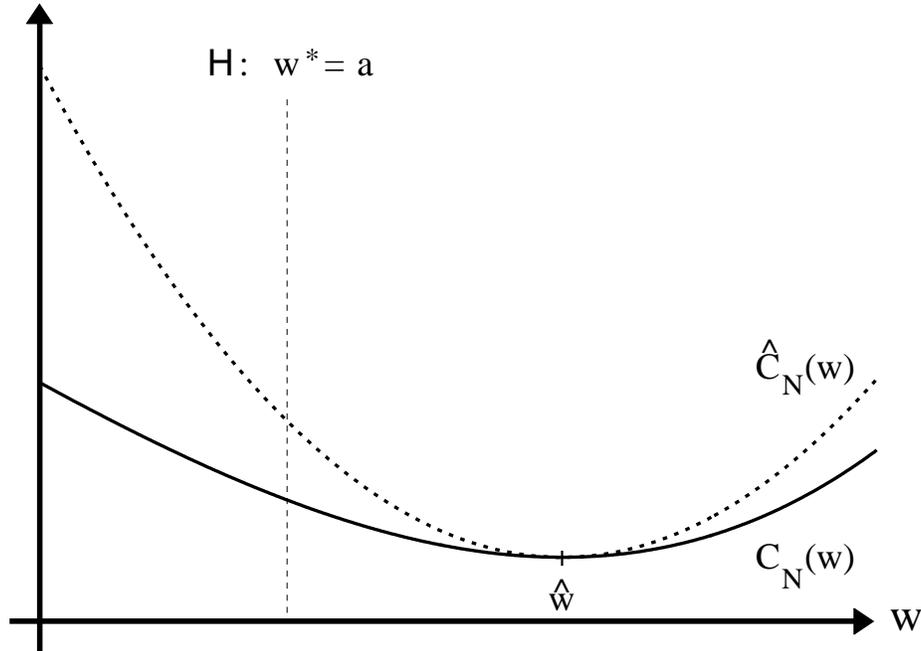


Figure 6.6: The “true” cost function, $C_N(\mathbf{w})$, and the approximation, $\hat{C}_N(\mathbf{w})$, based on a second order expansion around \hat{w} .

Using the inverse Hessian approximation $\mathbf{J}_N^{-1}(\mathbf{w}^*) \approx \mathbf{J}^{-1}(\mathbf{w}^*)$, $N \gg 2M + 1$ where M is the dependence lag (correlation length) of the input then

$$\begin{aligned} \Delta \mathbf{w} &\approx \mathbf{J}^{-1}(\mathbf{w}^*) \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \\ &= \mathbf{J}^{-1}(\mathbf{w}^*) \left(\frac{1}{N} \sum_{k=1}^N \psi(k; \mathbf{w}^*) e(k; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right). \end{aligned} \quad (6.184)$$

That is, $\Delta \mathbf{w}$, becomes a sum of N stochastic vector variables. Since the input, $\mathbf{x}(k)$ is not an i.i.d. sequence the stochastic variables are dependent, i.e., the usual form of the central limit theorem is not applicable. However, [Rosenblatt 85, Ch. 3.3] presents a central limit theorem which allows dependence.

In that context make the following definition⁵⁴:

DEFINITION 6.8 *The strictly stationary sequence $\mathbf{x}(k)$ is said to be strongly mixing [Rosenblatt 85, p. 62] if $\mathbf{x}(k)$, $\mathbf{x}(k + \tau)$ are asymptotically independent as $|\tau| \rightarrow \infty$. That is,*

$$\sup_{\mathbf{x}(k), \mathbf{x}(k+\tau)} |\text{Prob}\{\mathbf{x}(k)\mathbf{x}(k+\tau)\} - \text{Prob}\{\mathbf{x}(k)\}\text{Prob}\{\mathbf{x}(k+\tau)\}| \rightarrow 0, \text{ as } |\tau| \rightarrow \infty \quad (6.185)$$

where $\text{Prob}\{\cdot\}$ denotes probability.

⁵⁴Note that if $\mathbf{x}(k)$ is an M -dependent sequence cf. As. 6.5 then obviously it is a strongly mixing sequence.

The central limit theorem of [Rosenblatt 85, Ch. 3.3] concerns one-dimensional variables only and is in a restricted form given by:

THEOREM 6.11 *Let $\eta(k)$, $k = 1, 2, \dots$ be a strongly mixing sequence with zero mean, i.e., $E\{\eta(k)\} = 0$. Let*

$$\sigma^2 = E \left\{ \left[\sum_{k=1}^N \eta(k) \right]^2 \right\} \quad (6.186)$$

be the total variance of the sum of N samples of $\eta(k)$. Further all higher order moments are supposed to obey

$$\frac{E \left\{ \left| \sum_{k=1}^N \eta(k) \right|^{2+\delta} \right\}}{\sigma^{2+\delta}} \rightarrow 0, \quad N \rightarrow \infty \quad (6.187)$$

where $\delta > 0$.

Then as $N \rightarrow \infty$

$$\sum_{k=1}^N \eta(k) \rightarrow \mathcal{N}(0, \sigma^2). \quad (6.188)$$

This theorem can be used to state a theorem concerning vector variables also. A vector variable is Gaussian distributed if every linear combination of its components is Gaussian distributed. Furthermore, the correlation among the single components has to be taken into consideration. See further e.g., [Rao 65]. The result is that the sum of vector variables with zero mean vectors converges to a multivariate Gaussian distribution with zero mean and covariance matrix equal to the covariance matrix of the sum.

According to sa:dwstat the weight estimate is unbiased, i.e.,

$$E_T\{\Delta \mathbf{w}\} = J^{-1}(\mathbf{w}^*) E_T \left\{ \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \right\} = J^{-1}(\mathbf{w}^*) \frac{\partial C(\mathbf{w}^*)}{\partial \mathbf{w}} = \mathbf{0}, \quad (6.189)$$

since \mathbf{w}^* is the weights which minimize $C(\mathbf{w})$. Provided that As. 6.5 holds, or – less restrictive – $\mathbf{x}(k)$ is a strongly mixing sequence cf. Def. 6.8, then asymptotically $\Delta \mathbf{w}$ is Gaussian distributed with zero mean vector and covariance matrix (cf. sa:dwstat)

$$\mathbf{V} = \mathbf{J}^{-1}(\mathbf{w}^*) \cdot E_T \left\{ \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \left(\psi(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \cdot \left(\psi^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \right) \right\} \cdot \mathbf{J}^{-1}(\mathbf{w}^*). \quad (6.190)$$

By comparison with Sec. 6.5.8.3 we note that the evaluation of this covariance matrix is indeed a part of the derivation leading to the *GEN*-estimate when focusing on the term within the expectation sign in sa:covar. Dependent on various assumptions on the model, the input, etc., several covariance matrix estimates – denoted $\widehat{\mathbf{V}}$ – appear. Here we merely state the theorems. The reader is referred to the discussion in Sec. 6.5.8.3 and App. A for the details.

In the cases of an independent input sequence and no regularization the result coincides with those obtained in [White 81]. Furthermore, the case of dealing with complete models and no regularization is identical to the classical result regarding LS-estimates, see e.g., [Seber & Wild 89, Ch. 12].

THEOREM 6.12 *Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an NN-model which is either **incomplete** or alternatively **complete** with the restriction that \mathbf{w}^* (defined in As. 6.10) is not the global optimum of $G(\mathbf{w})$. Further, suppose that As. 6.3, 6.5 – 6.11 hold. The covariance estimate is then given by:*

$$\widehat{\mathbf{V}} = \frac{1}{N} \cdot \mathbf{H}_N^{-1}(\widehat{\mathbf{w}}) \left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1}(\widehat{\mathbf{w}}) \quad (6.191)$$

where the correlation matrices $\mathbf{R}(\tau)$, $0 \leq \tau \leq M$ are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) e(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \widehat{\mathbf{w}}) e(k + \tau; \widehat{\mathbf{w}}). \quad (6.192)$$

and the Hessian matrix $\mathbf{H}_N(\widehat{\mathbf{w}})$ is cf. sa:hndef calculated as:

$$\mathbf{H}_N(\widehat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \widehat{\mathbf{w}}) e(k; \widehat{\mathbf{w}}) \quad (6.193)$$

THEOREM 6.13 *Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an **incomplete** LX-model cf. sa:linmod. Further, suppose that As. 6.3, 6.5, 6.7, 6.9, and 6.11 hold. The covariance estimate is then given by:*

$$\widehat{\mathbf{V}} = \frac{1}{N} \cdot \mathbf{H}_N^{-1} \left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1} \quad (6.194)$$

where the correlation matrices $\mathbf{R}(\tau)$, $0 \leq \tau \leq M$ are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \mathbf{z}(k) e(k; \widehat{\mathbf{w}}) \mathbf{z}^\top(k + \tau) e(k + \tau; \widehat{\mathbf{w}}), \quad (6.195)$$

and

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \quad (6.196)$$

where $\mathbf{z}(k)$ is defined by sa:linmod.

THEOREM 6.14 *Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an NN-model which is either **incomplete** or alternatively **complete** with the restriction that \mathbf{w}^* (defined in As. 6.10) is not the global optimum of $G(\mathbf{w})$. Further, suppose that As. 6.4 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. 6.6 – 6.11 are supposed to hold. The covariance estimate is then given by:*

$$\widehat{\mathbf{V}} = \frac{1}{N} \cdot \mathbf{H}_N^{-1}(\widehat{\mathbf{w}}) \left(\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}) e^2(k; \widehat{\mathbf{w}}) \right) \mathbf{H}_N^{-1}(\widehat{\mathbf{w}}). \quad (6.197)$$

where $\mathbf{H}(\widehat{\mathbf{w}})$ is given by sa:hndeftwo.

THEOREM 6.15 *Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be an **incomplete** LX-model, cf. sa:linmod. Further, suppose that As. 6.4 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. 6.7, 6.9, and 6.11 are supposed to hold. The covariance estimate is then given by:*

$$\widehat{\mathbf{V}} = \frac{1}{N} \cdot \mathbf{H}_N^{-1} \left(\frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) e^2(k; \widehat{\mathbf{w}}) \right) \mathbf{H}_N^{-1}, \quad (6.198)$$

and

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \quad (6.199)$$

where $\mathbf{z}(k)$ is defined by sa:linmod.

THEOREM 6.16 *Suppose that As. 6.1, 6.2 hold and no regularization is employed. The model is assumed to be a **complete** NN- or LX-model. Further, suppose that As. 6.4 – 6.11 hold and that \mathbf{w}^* defined in As. 6.10 is the **global minimum**⁵⁵ of $G(\mathbf{w})$. Finally, let $E\{\varepsilon^2\} = \sigma_\varepsilon^2 \neq 0$. The covariance estimate is then given by:*

$$\widehat{\mathbf{V}} = \frac{S_N(\widehat{\mathbf{w}})}{N - m} \cdot \mathbf{H}_N(\widehat{\mathbf{w}})^{-1}, \quad N > m \quad (6.200)$$

where

$$\mathbf{H}_N(\widehat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}). \quad (6.201)$$

THEOREM 6.17 *Suppose that As. 6.1, 6.2 hold and regularization is employed. The model may be an NN- or an LX-model, **complete** as well as **incomplete**. Further, suppose that As. 6.3, 6.5 – 6.11 hold. The covariance estimate is then given by:*

$$\widehat{\mathbf{V}} = \frac{1}{N} \cdot \mathbf{J}_N^{-1}(\widehat{\mathbf{w}}) \left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N - \tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{J}_N^{-1}(\widehat{\mathbf{w}}) \quad (6.202)$$

where the involved matrices are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) e(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \widehat{\mathbf{w}}) e(k + \tau; \widehat{\mathbf{w}}), \quad 0 \leq \tau \leq M, \quad (6.203)$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\widehat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\widehat{\mathbf{w}})}{\partial \mathbf{w}^\top}, \quad (6.204)$$

$$\mathbf{J}_N(\widehat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \widehat{\mathbf{w}}) e(k; \widehat{\mathbf{w}}) + \frac{\kappa}{2} \frac{\partial^2 r(\widehat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (6.205)$$

If considering a LX-model then $\boldsymbol{\psi}(k; \widehat{\mathbf{w}})$ is replaced by $\mathbf{z}(k)$ defined in sa:linmod.

THEOREM 6.18 *Suppose that As. 6.1, 6.2 hold and regularization is employed. The model may be an NN- or an LX-model, **complete** as well as **incomplete**. Further, suppose that*

⁵⁵Note the possibility of more coexisting global minima in which the expected cost (i.e., $G(\mathbf{w}^*)$) reaches the same level. Further note that the requirement is trivially fulfilled when dealing with LX-models.

As. 6.4 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. 6.6 – 6.11 are supposed to hold. The covariance estimate is then given by:

$$\widehat{\mathbf{V}} = \frac{1}{N} \cdot \mathbf{J}_N^{-1}(\widehat{\mathbf{w}}) (\mathbf{S} + \mathbf{R}(0)) \mathbf{J}_N^{-1}(\widehat{\mathbf{w}}). \quad (6.206)$$

The involved matrices are calculated as:

$$\mathbf{R}(0) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}) e^2(k; \widehat{\mathbf{w}}), \quad (6.207)$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\widehat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\widehat{\mathbf{w}})}{\partial \mathbf{w}^\top}, \quad (6.208)$$

$$\mathbf{J}_N(\widehat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \widehat{\mathbf{w}}) e(k; \widehat{\mathbf{w}}) + \frac{\kappa}{2} \frac{\partial^2 r(\widehat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (6.209)$$

If considering a LX-model then $\boldsymbol{\psi}(k; \widehat{\mathbf{w}})$ is replaced by $\mathbf{z}(k)$ defined in `sa:linmod`.

When using the estimated covariance matrix in the test statistic `sa:teststat` then, in principle, it does not comply with the χ^2 -distribution any more. In the case of complete models cf. Th. 6.16 the test statistic follows a F -distribution, viz. $n \cdot F(n, N - m)$, see e.g., [Seber & Wild 89, Ch. 5.3]. However, when $N \rightarrow \infty$ then $n \cdot F(n, N - m) \rightarrow \chi^2(n)$. In the other cases it seems more troublesome to devise how the distribution is modified. Consequently, we are content that the test statistic obeys the χ^2 -distribution when N is sufficiently large.

6.6.4 Irregular Hypotheses

Testing the hypothesis that some weights are equal to zero may in certain cases be afflicted with difficulties. The problem is that under the hypothesis some of the weights may not be identifiable⁵⁶. That means the assumption concerning the existence of the inverse Hessian breaks down and consequently, the estimates do not obey a simple Gaussian distribution.

In connection with layered filter architectures as e.g., the multi-layer feed-forward perceptron neural network (MFPNN) the problem appears if one wants to test if a neuron can be removed. This difficulty has been pointed out by [White 89a].

As an illustrative example pick a 2-layer MFPNN (single linear output neuron) with two hidden neurons, i.e., cf. Sec. 3.2.2

$$\widehat{y}(k) = w_0^{(2)} + w_1^{(2)} h(\mathbf{z}^\top(k) \mathbf{w}_1^{(1)}) + w_2^{(2)} h(\mathbf{z}^\top(k) \mathbf{w}_2^{(1)}). \quad (6.210)$$

Testing if the second hidden neuron is removable is thus equivalent to test if all weights $w_2^{(2)}$, $\mathbf{w}_2^{(1)}$ are equal to zero. Now when $w_2^{(2)} = 0$ then $\mathbf{w}_2^{(1)}$ can take any value without influencing $\widehat{y}(k)$. Consequently, the cost function will be “flat” in the $\mathbf{w}_2^{(1)}$ -direction and as a result, the Hessian will be singular.

A procedure to remedy the obstacles is presented in [Gallant 77]. The idea is to treat $\mathbf{w}_2^{(1)}$ as a fixed, i.e., non-adjustable, vector. That is, the output of the neuron, $s_2^{(1)}(k) = h(\mathbf{z}^\top(k) \mathbf{w}_2^{(1)})$, acts like a regressor or external input signal to the output neuron.

⁵⁶In the statistical literature these weights are known as nuisance parameters which are identifiable only under the alternative, see e.g., [White 89a].

Under the hypothesis $(w_2^{(2)})^* = 0$ then all remaining weights are identifiable and follows a Gaussian distribution. Hence, the conventional test statistic and reject region listed above can be used.

Even though the issue of formulating a test has been solved the choice of the fixed weights, $\mathbf{w}_2^{(1)}$, still remains. In principle, these weights should be chosen so that the power of the test is maximized, i.e., maximizing the probability of rejecting the hypothesis under the alternative⁵⁷. However, in practice this goal is not attainable. [Gallant 77] suggests a sampling of the $\mathbf{w}_2^{(1)}$ -space which results in, say $Q - 1$, different regressors (or additional fixed neurons) $s_i^{(1)}(k)$, $i = 2, 3, \dots, Q$. Next estimate the model when these regressors are included and perform a test of the hypothesis: $(w_i^{(2)})^* = 0$, $\forall i = 2, 3, \dots, Q$. When Q increases the $\mathbf{w}_2^{(1)}$ -space becomes more densely sampled, i.e., if a non-zero optimal value of $w_i^{(2)}$ exists for some setting of $\mathbf{w}_2^{(1)}$ then the probability of finding this setting is increased. However, the power of the test decreases when Q increases, i.e., the probability of accepting a false hypothesis increases. To alleviate this obstacle [Gallant 77] suggest to use a principal component analysis (PCA) [Gallant 77] of the vector $[s_2^{(1)}(k), s_3^{(1)}(k), \dots, s_Q^{(1)}(k)]^\top$ and then only employ the significant principal components. The PCA is treated in detail in Ch. 4.

An alternative method is to estimate all weights in the model and then use the actual estimate, $\hat{\mathbf{w}}_2^{(1)}$, as the fixed values of these weights. When testing the hypothesis $(w_2^{(2)})^* = 0$ we simply do not regard $\mathbf{w}_2^{(1)}$ as adjustable weights, i.e., they are removed from the weight vector and furthermore the corresponding rows and columns of the covariance matrix are deleted. This scheme naturally reduces the power of the test; however it is tractable w.r.t. computational complexity. An additional hypothesis, $(\mathbf{w}_2^{(1)})^* = \mathbf{0}$, can be tested without particularly extra computational burden. Consequently, only if both hypotheses are accepted the neuron can be removed.

6.6.5 Retraining

After accepting a linear hypothesis $\Xi \mathbf{w}^* = \mathbf{a}$ the weights have to be reestimated in order to improve the generalization ability. Since the distribution of the weight estimates and the covariance matrix estimates presented above are based on a second order expansion of the cost function it seems natural to reestimate the weights *within* this second order expansion.

The second order approximation, say $\hat{C}_N(\mathbf{w})$, of the cost function around $\hat{\mathbf{w}}$ yields:

$$\hat{C}_N(\mathbf{w}) = C_N(\hat{\mathbf{w}}) + \delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \delta \mathbf{w} \quad (6.211)$$

where $\delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}$. The objective is now to find the weights, $\hat{\mathbf{w}}_{\mathcal{H}}$, which minimize $\hat{C}_N(\mathbf{w})$ under the constraint $\Xi \mathbf{w} = \mathbf{a}$ imposed by the hypothesis sa:hypo. In Fig. 6.7 an example of retraining is depicted. The optimization task is solved by using a Lagrange technique⁵⁸. That is, define the Lagrange cost function

$$E(\mathbf{w}; \boldsymbol{\lambda}) = \hat{C}_N(\hat{\mathbf{w}}) + \boldsymbol{\lambda}^\top (\Xi \mathbf{w} - \mathbf{a}) \quad (6.212)$$

⁵⁷In other words the weights $\mathbf{w}_2^{(1)}$ should be chosen so that the optimal value of the weight, $w_2^{(2)}$, is as large as possible.

⁵⁸A similar result is reported in e.g., [Seber & Wild 89, App. D].

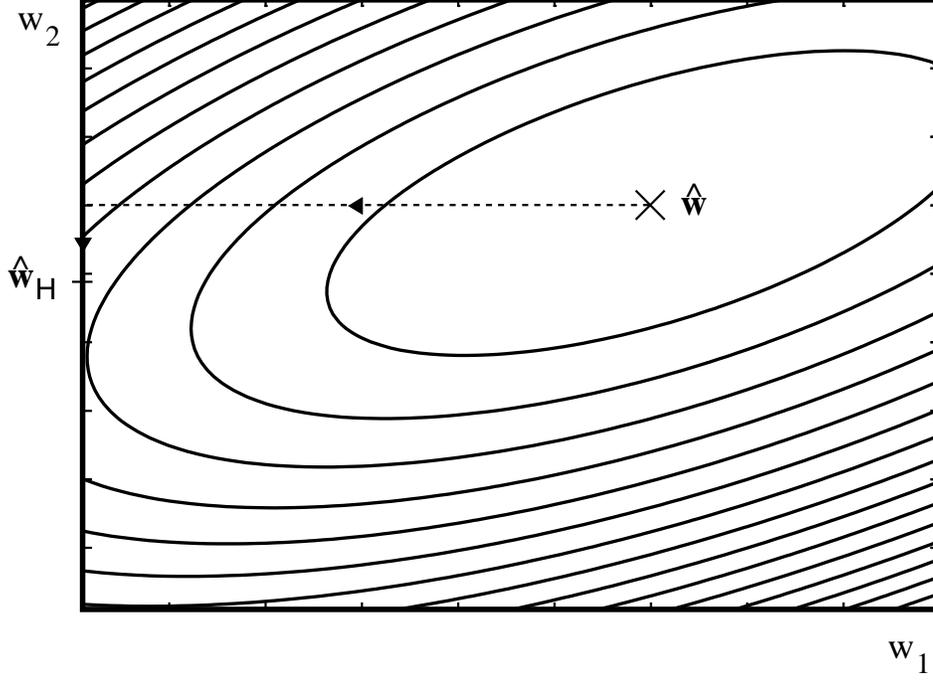


Figure 6.7: Retraining in the case of a model with two weights, w_1, w_2 . The hypothesis is that $w_1^* = 0$. The ellipses are equidistant levels of the cost function.

where λ is a n -dimensional vector of Lagrange multipliers. Now at the minimum, $\hat{\mathbf{w}}_{\mathcal{H}}$, the partial derivatives of $E(\cdot)$ w.r.t. \mathbf{w} and λ equal zero, i.e.,

$$\begin{aligned} \frac{\partial E(\hat{\mathbf{w}}_{\mathcal{H}}; \lambda)}{\partial \mathbf{w}} &= \frac{\partial C_N(\hat{\mathbf{w}}_{\mathcal{H}})}{\partial \mathbf{w}} + \Xi \lambda \\ &= -2\mathbf{J}_N(\hat{\mathbf{w}})\delta\hat{\mathbf{w}}_{\mathcal{H}} + \Xi \lambda \\ &= \mathbf{0}, \end{aligned} \quad (6.213)$$

and

$$\frac{\partial E(\hat{\mathbf{w}}_{\mathcal{H}}; \lambda)}{\partial \lambda} = \Xi \hat{\mathbf{w}}_{\mathcal{H}} - \mathbf{a} = \mathbf{0}. \quad (6.214)$$

Premultiplying sa:lg1 with $\Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}})$ and using the relation imposed by sa:lg2 yield:

$$\begin{aligned} -2\Xi \delta\hat{\mathbf{w}}_{\mathcal{H}} + \Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^T \lambda &= \mathbf{0} \\ -2\Xi \hat{\mathbf{w}} + 2\Xi \hat{\mathbf{w}}_{\mathcal{H}} + \Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^T \lambda &= \mathbf{0} \\ 2(-\Xi \hat{\mathbf{w}} + \mathbf{a}) + \Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^T \lambda &= \mathbf{0}. \end{aligned} \quad (6.215)$$

Consequently, isolating λ gives

$$\lambda = 2 \left(\Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^T \right)^{-1} (\Xi \hat{\mathbf{w}} - \mathbf{a}). \quad (6.216)$$

Finally, substituting this expression into sa:lg1 and rearranging give

$$\hat{\mathbf{w}}_{\mathcal{H}} = \hat{\mathbf{w}} - \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^T \left(\Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^T \right)^{-1} (\Xi \hat{\mathbf{w}} - \mathbf{a}). \quad (6.217)$$

If a second order algorithm – like the RGNB-algorithm – is employed for estimating the weights at first then $\mathbf{J}_N^{-1}(\hat{\mathbf{w}})$, or a suitable approximation, is an immediate spin-off. Consequently, the retraining involves the inversion of the $n \times n$ matrix $\mathbf{\Xi} \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \mathbf{\Xi}^\top$ only. This is in contrast to a full retraining which, in principle, requires the inversion of an $m \times m$ matrix. Often n is much less than m so the suggested scheme effectively reduces the computational burden.

A consequence of the retraining is that the cost increases slightly. This amount, say ΔC , is easily calculated without server extra computations. Using the expansion sa:costapp and applying sa:whyp result in

$$\begin{aligned} \Delta C &= \hat{C}_N(\hat{\mathbf{w}}_{\mathcal{H}}) - C_N(\hat{\mathbf{w}}) \\ &= \delta \hat{\mathbf{w}}_{\mathcal{H}}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \delta \hat{\mathbf{w}}_{\mathcal{H}} \\ &= (\mathbf{\Xi} \hat{\mathbf{w}} - \mathbf{a})^\top \left(\mathbf{\Xi} \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \mathbf{\Xi}^\top \right)^{-1} (\mathbf{\Xi} \hat{\mathbf{w}} - \mathbf{a}). \end{aligned} \quad (6.218)$$

Notice that the quantities which enter sa:deltac also enter the calculation of $\hat{\mathbf{w}}_{\mathcal{H}}$.

In Sec. 6.7 we shall see how this equation can be used to guide the search for weights which may possibly be removed from the model.

Another application of sa:deltac is that it enables a check of the validity of the presumed cost function approximation. This is done by comparing the cost at the reestimated weights, $C_N(\hat{\mathbf{w}}_{\mathcal{H}})$, with the estimate based on the second order expansion, viz. $C_N(\hat{\mathbf{w}}) + \Delta C$. If these quantities have a significant discrepancy then one should not trust the result of the performed test.

6.6.6 Similarities Between the Statistical Framework and Generalization Error Estimates

In this section we shall point out similarities between the statistical framework and the basic architecture synthesis algorithm based on generalization error estimates. These similarities have also been reported elsewhere, e.g., [Leontaritis & Billings 87], [Ljung 87].

Consider the simple case of complete modeling and no regularization, i.e., using the cost function $S_N(\mathbf{w})$. The covariance estimate yields in this case cf. Th. 6.16

$$\hat{\mathbf{V}} = \frac{S_N(\hat{\mathbf{w}})}{N - m} \cdot \mathbf{H}_N^{-1}(\hat{\mathbf{w}}). \quad (6.219)$$

Consequently, cf. sa:teststat the test statistic for testing the hypothesis, $\mathbf{\Xi} \mathbf{w}^* = \mathbf{a}$, becomes

$$T = \frac{(N - m) \cdot (\mathbf{\Xi} \hat{\mathbf{w}} - \mathbf{a})^\top \left(\mathbf{\Xi} \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \mathbf{\Xi}^\top \right)^{-1} (\mathbf{\Xi} \hat{\mathbf{w}} - \mathbf{a})}{S_N(\hat{\mathbf{w}})}, \quad (6.220)$$

and the hypothesis is rejected if $T > \chi_{1-\alpha}^2(n)$ according to sa:reject where α is the significance level. By comparison with the change in the cost function sa:deltac⁵⁹ we notice a strong resemblance between ΔS and the test statistic T . The reject region is characterized by the inequality:

$$\Delta S > \chi_{1-\alpha}^2(n) \frac{S_N(\hat{\mathbf{w}})}{N - m}. \quad (6.221)$$

⁵⁹Notice that in the case of no regularization then $C_N(\mathbf{w})$ translates into $S_N(\mathbf{w})$ and furthermore, the Hessian matrix \mathbf{J} translates into \mathbf{H} .

On the other hand, using the basic synthesis algorithm the hypothesis is rejected if the generalization error is maximum under the hypothesis. The generalization error estimator is in the present case the FPE -estimator according to Th. 6.6. Under the hypothesis the estimate yields

$$FPE_{\mathcal{H}} = \frac{N + m - n}{N - m + n} S_N(\hat{\mathbf{w}}_{\mathcal{H}}), \quad (6.222)$$

since the model contains $m - n$ degrees of freedom⁶⁰. Under the alternative the estimate is

$$FPE = \frac{N + m}{N - m} S_N(\hat{\mathbf{w}}). \quad (6.223)$$

Consequently, the reject region is given by the following inequality:

$$FPE_{\mathcal{H}} > FPE. \quad (6.224)$$

Using the relation, $S_N(\hat{\mathbf{w}}_{\mathcal{H}}) = S_N(\hat{\mathbf{w}}) + \Delta S$, and the expressions sa:fpe1, (6.223) then the reject region is expressed as:

$$\Delta S > \frac{2Nn}{N + m - n} \frac{S_N(\hat{\mathbf{w}})}{N - m}. \quad (6.225)$$

Comparing with the inequality for rejection based on statistical testing sa:ineqtest then the two equations are seen to be equal if

$$\chi_{1-\alpha}^2(n) \equiv \frac{2Nn}{N + m - n}. \quad (6.226)$$

An example of connected values of $\{N, m, n, \alpha\}$ in order to accomplish this equality is given in table Table 6.2.

n	$N = 100, m = 20$	$N \rightarrow \infty, \forall m$
1	20%	16%
2	19%	14%
5	13%	8%
10	5%	3%

Table 6.2: Significance levels, α , when varying n .

We notice a relatively high significance level when n is small corresponding to high probability that the hypothesis is rejected when it is actually true. Paraphrased, when using the FPE -estimator there is a tendency to select models which contain too many weights⁶¹.

6.7 Procedures for Pruning the Architecture

In this section various pruning procedures are discussed. In particular, we present novel procedures based on combining the Optimal Brian Damage recipe [Le Cun et al. 90] (and extensions hereof) with respectively the basic synthesis algorithm (Sec. 6.5.1) and the statistical framework (Sec. 6.6).

⁶⁰That is, a constrained weight vector with dimension $m - n$ can be found by projecting the original weight vector onto the subspace imposed by the hypothesis.

⁶¹This fact is closely connected with the inconsistency pointed out in Sec. 6.5.6.

6.7.1 Simple Pruning in Neural Networks

A simple procedure for pruning MFPNN's is described in [Sietsma & Dow 88]. The idea is to compare the output of the neurons in the different layers. Consider the state vector, $\mathbf{s}(k)$ which constitutes the collection of the neuron outputs in a particular layer with m neurons (including the bias neuron equal to unity). If $s_i(k)$ and $s_j(k)$, $i, j \in [0; m]$ respond similarly, i.e., carry the same information, then one of the neurons i or j can be removed. In [Sietsma & Dow 88] no particular method for measuring the similarity is provided; however, an obvious possibility is to use crosscorrelation. This is e.g., estimated by

$$\hat{\rho}_{s_i, s_j} = \frac{\hat{\gamma}_{s_i, s_j}}{\sqrt{\hat{\sigma}_{s_i}^2 \hat{\sigma}_{s_j}^2}} \quad (6.227)$$

where $\hat{\gamma}_{s_i, s_j}$ is the estimated crosscovariance

$$\hat{\gamma}_{s_i, s_j} = \frac{1}{N} \sum_{k=1}^N (s_i(k) - \langle s_i(k) \rangle) \cdot (s_j(k) - \langle s_j(k) \rangle) \quad (6.228)$$

with $\langle \cdot \rangle$ denoting the time-average and $\hat{\sigma}_{s_i}^2$ is the estimated variance equal to $\hat{\gamma}_{s_i, s_i}$. Since $|\hat{\rho}_{s_i, s_j}| \leq 1$ then if the correlation exceeds some prescribed threshold (less than 1) remove one of the neurons. However some disadvantages with this method exist:

- A heuristic procedure for setting the threshold is required. In principle, the threshold should be related to a confidence interval since we employ an estimate of the true crosscorrelation. This is; however, difficult since the distribution of the neuron outputs are not known.
- Only the removal of entire neurons is comprised. For instance, the possible removal of single weights is not accomplished.
- A high correlation among two neuron signals does not directly reflect how the generalization error is changed when removing one of the neurons. In particular, the weights which connect the neurons with the subsequent layer also enter the matter.
- Only the linear interaction between $s_i(k)$ and $s_j(k)$ is considered. That is, there may still be some vanishing nonlinear interaction and consequently both neurons carry relevant information.

6.7.2 Statistically Based Pruning Procedures

Consider testing hypotheses concerning the deletion of irrelevant weights, i.e., hypotheses of the form $\Xi \mathbf{w}^* = \mathbf{0}$. As mentioned previously, since each weight in the m -dimensional weight vector is either included or deleted then $2^m - 1$ possible hypotheses exist⁶². This number is even for moderate values of m an astronomic number, e.g., $2^{50} - 1 \approx 10^{15}$. This motivates the development of algorithms which select a limited number of plausible hypotheses which later on will be denoted: *weight deletion vector* (WDV) algorithms. The statistical framework presented in Sec. 6.6 makes such algorithms feasible.

⁶²The minus one stems from excluding the case of deleting all weights.

A particular hypothesis – indicated by the superscript (s) – concerning the deletion of n weights is specified by the $n \times m$ restriction matrix $\Xi_{n,m}^{(s)}$. In that context define the associated WDV:

$$\mathbf{d}_n^{(s)} = [d_{n,1}^{(s)}, d_{n,2}^{(s)}, \dots, d_{n,n}^{(s)}]^\top = \Xi_{n,m}^{(s)} \cdot [1, 2, \dots, m]^\top \quad (6.229)$$

which contains the indices of the weights under consideration for deletion. The superscript (s) then indicates a particular set of n weights. Obviously, there exist $C_{m,n}$ ⁶³ possible WDV's with dimension n , q.e., $s \in [1; C_{m,n}]$. In what follows we will use the designations WDV and hypothesis indiscriminately, and often the indices of the restriction matrix are omitted.

The task of a WDV-algorithm is to find a set of Q WDV's vectors

$$\mathcal{Q} = \{ \mathbf{d}_{n_1}^{(s_1)}, \mathbf{d}_{n_2}^{(s_2)}, \dots, \mathbf{d}_{n_Q}^{(s_Q)} \} \quad (6.230)$$

which *subsequently* are tested for deletion using statistical tests cf. Sec. 6.6 or by using the basic architecture synthesis algorithm cf. Sec. 6.5.1. Examples are provided below. In principle, \mathcal{Q} can contain more WDV's with same n ; however, the algorithms suggested below deal with only one WDV for each n , i.e., $n_i = n$, $n = 1, 2, \dots, Q$, $Q \leq m - 1$.

The statistical framework presented in Sec. 6.6 makes the construction of WDV-algorithms feasible. Recall cf. Sec. 6.6.6 that if the model is complete then the test statistic, T , is related to the change in the cost function, ΔS , when imposing the hypothesis. This is due to the fact that under the complete model assumption: $\widehat{\mathbf{V}} \propto \mathbf{H}_N^{-1}(\widehat{\mathbf{w}})$ where $\widehat{\mathbf{V}}$ is the estimated weight covariance matrix, and $\mathbf{H}_N^{-1}(\widehat{\mathbf{w}})$ is the inverse Hessian of the LS cost function. The reject region of a particular hypothesis is cf. sa:ineqtest given by:

$$\Delta S > \chi_{1-\alpha}^2(n) \frac{S_N(\widehat{\mathbf{w}})}{N - m}, \quad (6.231)$$

and

$$\begin{aligned} \Delta S &= S_N(\widehat{\mathbf{w}}_{\mathcal{H}}) - S_N(\widehat{\mathbf{w}}) \\ &= \widehat{\mathbf{w}}^\top \Xi^\top \left(\Xi \mathbf{H}_N^{-1}(\widehat{\mathbf{w}}) \Xi^\top \right)^{-1} \Xi \widehat{\mathbf{w}}. \end{aligned} \quad (6.232)$$

Recall that $\widehat{\mathbf{w}}_{\mathcal{H}}$ are the estimated weights under the hypothesis when performing a re-training based on a second order approximation of the cost function, cf. Sec. 6.6.5.

The hypothesis is accepted when ΔS is below the quantity at the right hand side of sa:rrpru. Evidently, WDV's which result in small ΔS 's are believed to be the most promising candidates⁶⁴. However, for a fixed n it is not a simple task to construct an algorithm which pick the WDV with minimum ΔS without performing an exhaustive search. Consequently, three different approximations of the expression for ΔS are contemplated. A particular approximation is denoted δS .

6.7.2.1 The Diagonal Hessian Approach

The first approach is based on a diagonal assumption of the Hessian, i.e., $\mathbf{H}_N(\widehat{\mathbf{w}}) = \text{diag} \{ [H_{11}, H_{22}, \dots, H_{mm}]^\top \}$. This approach coincides with that of the Optimal Brain

⁶³ $C_{m,n} = m!/(n!(m-n)!)$ is the binomial coefficient.

⁶⁴If we employ a cost which includes regularization then sa:deltas may be replaced by the similar expression for ΔC sa:deltac

Damage (OBD) recipe [Le Cun et al. 90]. Consider an n -dimensional WDV, \mathbf{d}_n , and substitute the Hessian approximation into sa:deltas. The approximation, $\delta S(\mathbf{d}_n)$ then yields:

$$\begin{aligned}\delta S(\mathbf{d}_n) &= \hat{\mathbf{w}}^\top \mathbf{\Xi}^\top \left(\mathbf{\Xi} \text{diag} \left\{ \left[H_{11}^{-1}, H_{22}^{-1}, \dots, H_{mm}^{-1} \right]^\top \right\} \mathbf{\Xi}^\top \right)^{-1} \mathbf{\Xi} \hat{\mathbf{w}} \\ &= \sum_{r=1}^n \hat{w}_{d_{n,r}}^2 \cdot H_{d_{n,r}, d_{n,r}}.\end{aligned}\quad (6.233)$$

Notice two implications of the diagonal Hessian assumption:

- The total change in the cost using the WDV, \mathbf{d}_n , is a sum of positive changes corresponding to an individual deletion of the weights. The individual changes are denoted the *saliencies* [Le Cun et al. 90] and defined by:

$$\varrho_{d_{n,r}} = \hat{w}_{d_{n,r}}^2 H_{d_{n,r}, d_{n,r}}. \quad (6.234)$$

- Since the Hessian is diagonal the estimates of the weights which are not in the WDV remain unaffected by the deletion (see also sa:whyp).

Now the WDV, \mathbf{d}_n^* , which minimizes $\delta S(\mathbf{d}_n)$ is given by:

$$\begin{aligned}\mathbf{d}_n^* &= \arg \min_{\mathbf{d}_n} \delta S(\mathbf{d}_n) \\ &= [\ell_1, \ell_2, \dots, \ell_n]^\top\end{aligned}\quad (6.235)$$

where ℓ_i , $i = 1, 2, \dots, m$ are the indices which appear by the ranking:

$$\varrho_{\ell_1} \leq \varrho_{\ell_2} \leq \dots \leq \varrho_{\ell_m}. \quad (6.236)$$

6.7.2.2 Approach based on Individual Deletion

The second approach is based on the assumption that the weights in the WDV can be deleted independently, q.e., sa:deltas is a sum of individual contributions. The increase in cost, $\tilde{\varrho}_{d_{n,r}}$ due to the deletion of weight # $d_{n,r}$ is cf. sa:deltas

$$\tilde{\varrho}_{d_{n,r}} = \frac{\hat{w}_{d_{n,r}}^2}{P_{d_{n,r}, d_{n,r}}} \quad (6.237)$$

where $\text{diag}\{\mathbf{H}_N^{-1}(\hat{\mathbf{w}})\} = [P_{11}, P_{22}, \dots, P_{mm}]^\top$. $\tilde{\varrho}$ is denoted the *modified saliency*. Accordingly, the ΔS approximation regarding the WDV, \mathbf{d}_n , becomes:

$$\delta S(\mathbf{d}_n) = \sum_{r=1}^n \tilde{\varrho}_{d_{n,r}}. \quad (6.238)$$

As in the previous subsection:

$$\begin{aligned}\mathbf{d}_n^* &= \arg \min_{\mathbf{d}_n} \delta S(\mathbf{d}_n) \\ &= [\ell_1, \ell_2, \dots, \ell_n]^\top\end{aligned}\quad (6.239)$$

where ℓ_i , $i = 1, 2, \dots, m$ are the indices which appear by the ranking:

$$\tilde{\varrho}_{\ell_1} \leq \tilde{\varrho}_{\ell_2} \leq \dots \leq \tilde{\varrho}_{\ell_m}. \quad (6.240)$$

Note that the only modification compared to the diagonal Hessian approach is that H_{nn} is replaced by P_{nn}^{-1} . However, the modification may be essential when a considerable correlation among the weights is present. This is indeed the usual case. Consider e.g., a model with two weights, w_1, w_2 which are highly correlated, i.e., let $\mathbf{H}(\hat{\mathbf{w}}) = \{H_{ij}\}$ then $H_{11}H_{22} - H_{12}^2$ is close to zero. This allows the deletion of one of the weights. Let us focus on weight #1. The saliency is given by: $\varrho_1 = \hat{w}_1^2 H_{11}$. On the other hand, the modified saliency yields:

$$\tilde{\varrho}_1 = \hat{w}_1^2 P_{11}^{-1} = \hat{w}_1^2 \left(H_{11} - \frac{H_{12}^2}{H_{22}} \right). \quad (6.241)$$

Obviously, $\tilde{\varrho}_1 \ll \varrho_1$ which indicates that weight #1 is regarded as a good candidate for deletion when using the modified saliency while this fact possibly is not detected at all when using the saliency.

6.7.2.3 Approach based on a Block Hessian Structure

In the previous approach the correlation among the weights is only partly taken into account. The correlation which enters the issue concerning simultaneous deletion of more weights is neglected. Recall the example above considering a model with two highly correlated weights. The deletion of both weights yields according to sa:modsal, (6.238)

$$\begin{aligned} \delta S &= \hat{w}_1^2 P_{11}^{-1} + \hat{w}_2^2 P_{22}^{-1} \\ &= \hat{w}_1^2 \left(H_{11} - \frac{H_{12}^2}{H_{22}} \right) + \hat{w}_2^2 \left(H_{22} - \frac{H_{12}^2}{H_{11}} \right), \end{aligned} \quad (6.242)$$

while the exact expression (cf. sa:deltas with $\Xi = \mathbf{I}$) gives:

$$\begin{aligned} \Delta S &= \hat{\mathbf{w}}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \hat{\mathbf{w}} \\ &= \hat{w}_1^2 H_{11} + 2\hat{w}_1 \hat{w}_2 H_{12} + \hat{w}_2^2 H_{22}. \end{aligned} \quad (6.243)$$

Since the high correlation implies that $H_{11}H_{22} - H_{12}^2$ is close to zero δS is typically a small value. This indicates that the WDV containing both weights is a good candidate. However, ΔS may still be rather large for which reason both weights can not be deleted.

In order to take this drawback in hand we make the following assumption: The weights within the current WDV are uncorrelated with the remaining weights. That is, if $\mathbf{d}_n = [1, 2, \dots, n]^\top$ then the Hessian has the block diagonal structure:

$$\mathbf{H}_N(\hat{\mathbf{w}}) = \begin{bmatrix} \mathbf{H}_{n,n} & \mathbf{0}_{n,m-n} \\ \mathbf{0}_{m-n,n} & \mathbf{H}_{m-n,m-n} \end{bmatrix}. \quad (6.244)$$

In general the assumption implies:

$$\left(\Xi \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \Xi \right)^\top = \Xi \mathbf{H}_N(\hat{\mathbf{w}}) \Xi. \quad (6.245)$$

Consequently, the approximate expression for the change in cost yields:

$$\delta S(\mathbf{d}_n) = \hat{\mathbf{w}}^\top \Xi^\top \Xi \mathbf{H}_N(\hat{\mathbf{w}}) \Xi^\top \Xi \hat{\mathbf{w}}. \quad (6.246)$$

Note that $\delta S(\mathbf{d}_n)$ is the change in the cost function when deleting the weights \mathbf{d}_n without retraining the remaining weights. The next step is to solve the minimization task:

$$\mathbf{d}_n^* = \arg \min_{\mathbf{d}_n} \delta S(\mathbf{d}_n), \quad n = 1, 2, \dots, m-1. \quad (6.247)$$

The expression for $\delta S(\mathbf{d}_n)$ possesses a nice recursive property. Suppose that $\delta S(\mathbf{d}_n)$ is calculated then $\delta S(\mathbf{d}_{n+1})$ where $\mathbf{d}_{n+1} = [\mathbf{d}_n^\top, d_{n+1,n+1}]^\top$ is given by:

$$\delta S(\mathbf{d}_{n+1}) = \delta S(\mathbf{d}_n) + \hat{w}_{d_{n+1,n+1}}^2 H_{d_{n+1,n+1}} + 2\hat{w}_{d_{n+1,n+1}} \sum_{r=1}^n \hat{w}_{d_{n+1,r}} H_{d_{n+1,n+1},d_{n+1,r}} \quad (6.248)$$

where $\mathbf{H}_N(\hat{\mathbf{w}}) = \{H_{ij}\}$. This property opens up the prospect of using recursive search algorithms. We suggest a modification of the Branch-and-Bound algorithm [Winston 84, pp. 102–114] the so-called Stack Algorithm.

Define the l 'th stack element as the 2-tuple:

$$\mathcal{E}_l = \left\{ \mathbf{d}_{r_l}^{(s_l)}, \delta S(\mathbf{d}_{r_l}^{(s_l)}) \right\}. \quad (6.249)$$

consisting of a WDV and the corresponding cost. r is the current dimension of the WDV and $s \in [1; C_{m,r}]$ indicates the particular WDV of the $C_{m,r}$ possible. The stack is given by $\mathcal{S} = \{\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_l\}$, $l \leq \varsigma$ where ς is the stack-size. The element, \mathcal{E}_1 , is referred to as the top of the stack.

The optimal WDV, \mathbf{d}_n^* , for each $n = 1, 2, \dots, m-1$ is found by running the

Stack Algorithm Initialize the stack:

$$\mathcal{S} = \{\emptyset, 0\} \quad (6.250)$$

where \emptyset is the empty WDV. Further initialize the stack element counter $l = 1$ and the dimension of the top element $r = 0$. Until the top element contains a WDV with dimension $r = n$ do:

- Step 2*
- a. Remove the top element, $\mathcal{E}_1 = \{\mathbf{d}_r^{(s_1)}, \delta S(\mathbf{d}_r^{(s_1)})\}$, of the stack and decrement l .
 - b. Form a set of WDV's, $\mathbf{d}_{r+1}^{(s_{l+j})}$ with dimension $r+1$ by expanding the removed WDV according to:

$$\mathbf{d}_{r+1}^{(s_{l+j})} = \left[\mathbf{d}_r^{(s_1)}, \ell_j \right]^\top, \quad j = 1, 2, \dots, j_{\max} \quad (6.251)$$

where

$$j_{\max} = m - n + r + 1 - \max_i d_{r,i}^{(s_1)} \quad (6.252)$$

$$\ell_j = j + \max_i d_{r,i}^{(s_1)}. \quad (6.253)$$

Note, that $\max_i d_{0,i}^{(s_1)} = 0$.

Calculate the corresponding cost changes, $\delta S(\mathbf{d}_{r+1}^{(s_{l+j})})$ according to sa:costrec and add novel elements to the stack:

$$\mathcal{S} = \mathcal{S} \cup \left\{ \mathbf{d}_{r+1}^{(s_{l+j})}, \delta S(\mathbf{d}_{r+1}^{(s_{l+j})}) \right\}, \quad j = 1, 2, \dots, j_{\max}. \quad (6.254)$$

- c. Sort the elements in the stack according to the δS 's so that the cost of the top element – i.e., $\delta S(\mathbf{d}_{r_1}^{(s_1)})$ – is the lowest. Update the dimension of the top element: $r = r_1$.

- d. If the number of elements in the stack is larger than the stack-size, i.e., $l + j_{\max} > \varsigma$, then discard the surplus elements in the bottom of the stack and update $l = \varsigma$; otherwise $l \leftarrow l + j_{\max}$.

Step 3 The optimal WDV is the top element of the stack, i.e., $\mathbf{d}_n^* = \mathbf{d}_r^{(1)}$.

In Table 6.3 the functionality of the Stack Algorithm is demonstrated by showing the search tree and the corresponding stack for a simple example with parameters: $m = 6$, $n = 4$, $\varsigma = 4$.

Notice certain facts concerning the Stack Algorithm:

- The expansion strategy, cf. *Step 2b.*, ensures:
 - All WDV's which stem from expanding $\mathbf{d}_r^{(s_1)}$ are different from the WDV's due to an expansion of $\mathbf{d}_r^{(s_2)}$, $s_1 \neq s_2$.
 - Any of the $C_{m,n}$ possible WDV's with dimension n can be attained in n steps by recursively expanding the empty WDV.
- If the stack-size is larger than all possible WDV with dimension less than or equal to the current dimension n , i.e., if $\varsigma > \sum_{r=1}^n C_{m,r}$, and secondly, δS is a non-decreasing function of r then the Stack Algorithm finds the WDV which globally minimizes the change in cost. Frequently, the second assumption is not met⁶⁵; however, the guarantee of finding the global minimum can be retained by searching according to the modified cost:

$$\tilde{\delta S}(\mathbf{d}_r^{(s_r)}) = \delta S(\mathbf{d}_r^{(s_r)}) + L(\mathbf{d}_r^{(s_r)}), \quad \forall r < n \quad (6.255)$$

where $L(\mathbf{d}_r^{(s_r)})$ is an estimate of the remaining δS when expanding $\mathbf{d}_r^{(s_r)}$ to an n -dimensional WDV. If $L(\cdot)$ is an *underestimate* of the true remaining cost then the global minimum is found. An estimate of $L(\cdot)$ can be obtained by using the diagonal Hessian approach, cf. Sec. 6.7.2.1

$$L(\mathbf{d}_r^{(s_r)}) = \sum_{j=1}^{n-r} \varrho_{\ell_j}, \quad (6.256)$$

and ℓ_j are the $n - r$ indices which comply with:

1. ℓ_j is larger than the minimum required by the expansion rule. That is, according to sa:elmin: $j_{\min} = \max_i d_{r,i}^{(s_r)}$.
2. ℓ_j is required to be as small as possible according to the ranking of saliencies (cf. sa:obdrank), i.e.,

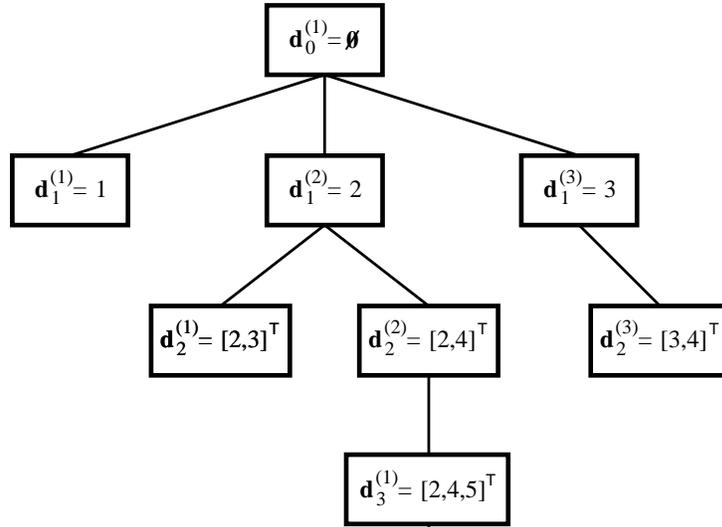
$$\varrho_{\ell_1} \leq \varrho_{\ell_2} \leq \cdots \leq \varrho_{\ell_{m-j_{\min}}} \wedge \ell_j \in [j_{\min} + 1; m]. \quad (6.257)$$

Note that the proposed estimate of $L(\cdot)$ is positive; consequently, it is not ensured to be an underestimate.

In addition, the introduction of the modified cost may be beneficial since a lot of irrelevant paths in the search tree are sooner detected and thereby the search time is reduced.

However, the simulations presented in Ch. 7 are not based on the modified cost.

⁶⁵For instance, including an extra weight which is highly correlated with some of the weights already in the WDV will possibly result in a drop in δS .



The Stack Content, First Pass	
$\mathbf{d}_1^{(2)} = 2$	$\delta S(\mathbf{d}_1^{(2)}) = \hat{w}_2^2 H_{22}$
$\mathbf{d}_1^{(3)} = 3$	$\delta S(\mathbf{d}_1^{(3)}) = \hat{w}_3^2 H_{33}$
$\mathbf{d}_1^{(1)} = 1$	$\delta S(\mathbf{d}_1^{(1)}) = \hat{w}_1^2 H_{11}$

The Stack Content, Second Pass	
$\mathbf{d}_1^{(3)} = 3$	$\delta S(\mathbf{d}_1^{(3)})$
$\mathbf{d}_2^{(2)} = [2, 4]^\top$	$\delta S(\mathbf{d}_2^{(2)}) = \delta S(\mathbf{d}_1^{(2)}) + \hat{w}_4^2 H_{44} + 2\hat{w}_2\hat{w}_4 H_{24}$
$\mathbf{d}_1^{(1)} = 1$	$\delta S(\mathbf{d}_1^{(1)})$
$\mathbf{d}_2^{(1)} = [2, 3]^\top$	$\delta S(\mathbf{d}_2^{(1)}) = \delta S(\mathbf{d}_1^{(2)}) + \hat{w}_3^2 H_{33} + 2\hat{w}_2\hat{w}_3 H_{23}$

The Stack Content, Third Pass	
$\mathbf{d}_2^{(2)} = [2, 4]^\top$	$\delta S(\mathbf{d}_2^{(2)})$
$\mathbf{d}_2^{(3)} = [3, 4]^\top$	$\delta S(\mathbf{d}_2^{(3)}) = \delta S(\mathbf{d}_1^{(3)}) + \hat{w}_4^2 H_{44} + 2\hat{w}_3\hat{w}_4 H_{34}$
$\mathbf{d}_1^{(1)} = 1$	$\delta S(\mathbf{d}_1^{(1)})$
$\mathbf{d}_2^{(1)} = [2, 3]^\top$	$\delta S(\mathbf{d}_2^{(1)})$

The Stack Content, Fourth Pass	
$\mathbf{d}_3^{(1)} = [2, 4, 5]^\top$	$\delta S(\mathbf{d}_3^{(1)}) = \delta S(\mathbf{d}_2^{(2)}) + \hat{w}_5^2 H_{55} + 2\hat{w}_2\hat{w}_5 H_{25} + 2\hat{w}_4\hat{w}_5 H_{45}$
$\mathbf{d}_2^{(3)} = [3, 4]^\top$	$\delta S(\mathbf{d}_2^{(3)})$
$\mathbf{d}_1^{(1)} = 1$	$\delta S(\mathbf{d}_1^{(1)})$
$\mathbf{d}_2^{(1)} = [2, 3]^\top$	$\delta S(\mathbf{d}_2^{(1)})$

Table 6.3: Example of running the Stack Algorithm.

- In general the above requirement on the stack-size, i.e., $\varsigma > \sum_{r=1}^n C_{m,r}$ may turn out to be inexpedient due to the involved computational complexity. Consequently we

suggest to use:

$$\varsigma = c \cdot (m - r + 1) \quad (6.258)$$

where c is a proper factor. In all simulations in Ch. 7 we used $c = 4$. Notice that $m - r + 1$ is the maximum number of novel WDV's created by the expansion rule cf. sa:elmin.

6.7.3 Statistical Pruning Algorithms

The WDV-algorithms presented above can be combined with the statistical framework Sec. 6.6 for hypothesis testing in order to form the following algorithm⁶⁶:

Statistical Pruning Algorithm Choose a filter architecture parameterized by the m -dimensional weight vector, \mathbf{w} , and select a WDV-algorithm cf. Sec. 6.7.2.1–6.7.2.3. Furthermore, specify the significance level α . Calculate the estimated weights, $\hat{\mathbf{w}}$, the Hessian matrix $\mathbf{J}_N(\hat{\mathbf{w}})$, and the inverse Hessian $\mathbf{J}_N^{-1}(\hat{\mathbf{w}})$. Estimate the weight covariance matrix, $\widehat{\mathbf{V}}$, according to Th. 6.12–6.18. Initialize the number of weights for deletion $n = Q < m$. Find the optimal WDV: \mathbf{d}_n^* and the corresponding $n \times m$ dimensional restriction matrix Ξ . If the hypothesis given by the WDV is irregular then employ the methods described in Sec. 6.6.4 to construct a modified test. Calculate the test statistic:

$$T = \hat{\mathbf{w}}^\top \Xi^\top \left(\Xi \widehat{\mathbf{V}} \Xi^\top \right)^{-1} \Xi \hat{\mathbf{w}}. \quad (6.259)$$

When $T > \chi_{1-\alpha}^2(n)$ and if $n > 1$ then decrement n and go to *Step 5*.; otherwise, go to *Step 8*. If $n > 0$ reestimate the weights within a second order cost function approximation, cf. Sec. 6.6.5, i.e.,

$$\hat{\mathbf{w}}_{\mathcal{H}} = \hat{\mathbf{w}} - \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^\top \left(\Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^\top \right)^{-1} \Xi \hat{\mathbf{w}} \quad (6.260)$$

where $\hat{\mathbf{w}}_{\mathcal{H}}$ is the reestimated weight vector under the hypothesis given by \mathbf{d}_n^* . In addition the second order approximation of the training cost yields:

$$\widehat{C}_N(\hat{\mathbf{w}}_{\mathcal{H}}) = C_N(\hat{\mathbf{w}}) + \hat{\mathbf{w}}^\top \Xi^\top \left(\Xi \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \Xi^\top \right)^{-1} \Xi \hat{\mathbf{w}}. \quad (6.261)$$

6.7.4 Pruning Algorithms based on Generalization Error Estimates

Another possibility consists in using WDV-algorithms within the basic filter synthesis algorithm Sec. 6.5.1. That is, the (estimated) generalization error of the original model is compared to the generalization errors when imposing the restrictions given by the set of WDV's. Finally, the model which generalizes the best is selected. A direct implementation requires that the weights have to be reestimated for each imposed restriction which cause a tremendous computational burden. However, performing the reestimation within

⁶⁶Notice, that the algorithm is presented in its most general form employing a cost function with regularization. That is, the Hessian is in general denoted $\mathbf{J}(\cdot)$.

a second order approximation of the cost function (see Sec. 6.6.5) significantly reduces the computational complexity. The task is next to demonstrate how the generalization error estimate is evaluated under the hypothesis imposed by a particular WDV $\mathbf{d}_n^{(r)}$ – or equivalently the $n \times m$ restriction matrix Ξ . Let us consider two particular generalization error estimators:

- Step 1**
1. The *GEN*-estimator for the case of a complete model cf. Th. 6.6, i.e., no regularization is employed. This estimator coincides with the *FPE*-estimator presented in Sec. 6.5.6.
 2. The *GEN*-estimator for a general incomplete model when employing regularization, according to Th. 6.8.

The matter concerning the *FPE*-estimator is previously discussed in Sec. 6.6.6. The estimate under the hypothesis, $FPE_{\mathcal{H}}$, given by the current WDV is:

$$\begin{aligned} FPE_{\mathcal{H}} &= \frac{N+m-n}{N-m+n} S_N(\hat{\mathbf{w}}_{\mathcal{H}}) \\ &= \frac{N+m-n}{N-m+n} (S_N(\hat{\mathbf{w}}) + \Delta S) \end{aligned} \quad (6.262)$$

where $\hat{\mathbf{w}}_{\mathcal{H}}$ is the estimated weights under the hypothesis, and ΔS is the increase in cost given by (see sa:deltas):

$$\Delta S = \hat{\mathbf{w}}^{\top} \Xi^{\top} \left(\Xi \mathbf{H}_N(\mathbf{w})^{-1} \Xi^{\top} \right)^{-1} \Xi \hat{\mathbf{w}}. \quad (6.263)$$

Consequently, the only inverse matrix operation involved in the determination of $FPE_{\mathcal{H}}$ is the inversion of the $n \times n$ matrix $\Xi \mathbf{H}_N(\mathbf{w})^{-1} \Xi^{\top}$ in contrast to a fully retraining which would involve inversion of an $(m-n) \times (m-n)$ matrix.

Next focus on the *GEN*-estimator which according to Th. 6.8 is given by:

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^{\top}(\tau)) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (6.264)$$

where

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^{\top}(k+\tau; \hat{\mathbf{w}}) e(k+\tau; \hat{\mathbf{w}}), \quad 0 \leq \tau \leq M, \quad (6.265)$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^{\top}}, \quad (6.266)$$

and

$$\mathbf{J}_N(\hat{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}) \boldsymbol{\psi}^{\top}(k; \mathbf{w}) - \boldsymbol{\Psi}(k; \mathbf{w}) e(k; \mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^{\top}}. \quad (6.267)$$

The Hessian matrix, $\mathbf{J}_N(\hat{\mathbf{w}})$, the correlation matrices, $\mathbf{R}(\tau)$, and \mathbf{S} are not changed within a second order expansion of the cost function cf. App. A. That means, when evaluating the *GEN*-estimate under the hypothesis, i.e., $GEN_{\mathcal{H}}$, then the only operation required is to pick the rows and columns corresponding to the weights which are retained. In that context define the $(m-n) \times m$ retention matrix, $\boldsymbol{\Upsilon}$, which specifies the weights *which are*

not in the WDV. That is i 'th row of \mathbf{Y} contains a one in position j if the j 'th weight is not in the WDV; otherwise, zeros. Hence, let $\mathbf{P} = \mathbf{J}_N^{-1}(\hat{\mathbf{w}})$ then the involved matrices are substituted by:

$$\begin{aligned} \mathbf{P} &\rightsquigarrow \tilde{\mathbf{P}} = \left(\mathbf{Y} \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \mathbf{Y}^\top \right)^{-1} \\ \mathbf{S} &\rightsquigarrow \tilde{\mathbf{S}} = \mathbf{Y} \mathbf{S} \mathbf{Y}^\top \\ \mathbf{R}(\tau) &\rightsquigarrow \tilde{\mathbf{R}}(\tau) = \mathbf{Y} \mathbf{R}(\tau) \mathbf{Y}^\top \end{aligned} \quad (6.268)$$

The determination of $\tilde{\mathbf{P}}$ involves at first sight inversion of a $(m-n) \times (m-n)$ matrix; however, by reorganizing the rows and columns of the Hessian matrix to comply with

$$\begin{bmatrix} \mathbf{\Xi} \mathbf{J} \mathbf{\Xi}^\top & \mathbf{\Xi} \mathbf{J} \mathbf{Y}^\top \\ \mathbf{Y} \mathbf{J} \mathbf{\Xi}^\top & \mathbf{Y} \mathbf{J} \mathbf{Y}^\top \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix},$$

the results concerning the inversion of partitioned matrices [Seber & Wild 89, App. A3] can be used to achieve the following result:

$$\tilde{\mathbf{P}} = \mathbf{Y} \mathbf{P} \mathbf{Y}^\top - \mathbf{Y} \mathbf{P} \mathbf{\Xi}^\top \left(\mathbf{\Xi} \mathbf{P} \mathbf{\Xi}^\top \right)^{-1} \mathbf{\Xi} \mathbf{P} \mathbf{Y}^\top. \quad (6.269)$$

Obviously, merely the inversion of an $n \times n$ matrix is required. Hence, $GEN_{\mathcal{H}}$ becomes:

$$GEN_{\mathcal{H}} = S_N(\hat{\mathbf{w}}) + \Delta S + \frac{2}{N} \cdot \text{tr} \left[\left(\tilde{\mathbf{S}} + \tilde{\mathbf{R}}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} \left(\tilde{\mathbf{R}}(\tau) + \tilde{\mathbf{R}}^\top(\tau) \right) \right) \tilde{\mathbf{P}} \right]. \quad (6.270)$$

Generalization Error based Pruning Algorithm Choose a filter architecture parameterized by the m -dimensional weight vector, \mathbf{w} . Calculate the estimated weights, $\hat{\mathbf{w}}$, the Hessian matrix $\mathbf{J}_N(\hat{\mathbf{w}})$, and the inverse Hessian $\mathbf{J}_N^{-1}(\hat{\mathbf{w}})$. Select a WDV-algorithm cf. Sec. 6.7.2.1–6.7.2.3 of form a set of optimal WDV's

$$\mathcal{Q} = \left\{ \mathbf{d}_1^*, \mathbf{d}_2^*, \dots, \mathbf{d}_Q^* \right\} \quad (6.271)$$

and corresponding restriction matrices. Calculate the generalization error estimate, $GEN_{\mathcal{H}_r}$, for each hypothesis, \mathcal{H}_r , $r = 1, 2, \dots, Q$, imposed by \mathcal{Q} according to Th. 6.2–6.9 and the results stated above. Determine:

$$n = \arg \min_r GEN_{\mathcal{H}_r}. \quad (6.272)$$

If $n = 0$ conclude that no weights can be deleted. Otherwise, reestimate the weights within a second order cost function approximation, cf. Sec. 6.6.5, i.e.,

$$\hat{\mathbf{w}}_{\mathcal{H}_n} = \hat{\mathbf{w}} - \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \mathbf{\Xi}^\top \left(\mathbf{\Xi} \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \mathbf{\Xi}^\top \right)^{-1} \mathbf{\Xi} \hat{\mathbf{w}} \quad (6.273)$$

where $\widehat{\mathbf{w}}_{\mathcal{H}_n}$ is the reestimated weight vector under the hypothesis given by \mathbf{d}_n^* . In addition the second order approximation of the training cost yields:

$$\widehat{C}_N(\widehat{\mathbf{w}}_{\mathcal{H}_n}) = C_N(\widehat{\mathbf{w}}) + \widehat{\mathbf{w}}^\top \boldsymbol{\Xi}^\top \left(\boldsymbol{\Xi} \mathbf{J}_N^{-1}(\widehat{\mathbf{w}}) \boldsymbol{\Xi}^\top \right)^{-1} \boldsymbol{\Xi} \widehat{\mathbf{w}}. \quad (6.274)$$

6.7.5 Optimal Brain Damage

The diagonal Hessian approach Sec. 6.7.2.1 is in the Optimal Brain Damage (OBD) pruning procedure [Le Cun et al. 90] exploited for both forming the WDV's, and for deciding which weights to delete. The procedure deals with the usual LS cost function and runs as follows:

The OBD-Procedure Choose a filter architecture parameterized by an m -dimensional weight vector, $\mathbf{w}_m = [w_{m,1}, w_{m,2}, \dots, w_{m,m}]^\top$. Initialize the number of deleted weights, $n = 0$. Estimate the weights, i.e., determine $\widehat{\mathbf{w}}_{m-n}$, and the diagonal elements of the Hessian matrix, H_{ii} . Compute saliencies, ϱ_i according to sa:saliency and perform the ranking (cf. sa:obdrank):

$$\varrho_{\ell_1} \leq \varrho_{\ell_2} \leq \dots \leq \varrho_{\ell_{m-n}}. \quad (6.275)$$

Delete r weights $w_{m-n,\ell_1}, w_{m-n,\ell_2}, \dots, w_{m-n,\ell_r}$ with low saliencies. Update $n \leftarrow n + r$. Go to *Step 2*.

The predominant benefit of the OBD scheme is that the computation of the saliencies is relatively simple. This is, of course, important when dealing with large networks, as in [Le Cun et al. 90]. However, the OBD procedure is afflicted with some drawbacks:

- Step 4**
- The off-diagonal elements of the Hessian, which are typically important, are not taken into account.
 - A complete model assumption is implicitly done since the coherence between the increase in cost, ΔS , cf. sa:deltas and the test statistic is valid only when the model is complete.
 - In order to avoid the deletion of essential weights, r has to be relatively small compared to m . Consequently, if many weights eventually can be deleted several re-training phases are required. No precise criterion for selecting, r , and secondly for stopping the algorithm is provided. However, such criteria can simply be provided by interpreting OBD as an statistical test. The reject region of the statistical test based on the approximation:

$$\delta S(\mathbf{d}_r) = \sum_{i=1}^r \widehat{\mathbf{w}}_{d_r,i}^2 \cdot H_{d_r,i,d_r,i} \quad (6.276)$$

is according to sa:rrpru given by

$$\delta S(\mathbf{d}_r) > \chi_{1-\alpha}^2(r) \frac{S_N(\widehat{\mathbf{w}}_{m-n})}{N - m - n}. \quad (6.277)$$

The factor $\chi_{1-\alpha}^2(r)/(N-m)$ represents the acceptable percentage increase in the cost when deleting the weights. A straight forward modification of the OBD procedure is thus to delete r weights if

$$\delta S(\mathbf{d}_r) < \frac{\chi_{1-\alpha}^2(r)}{N-m-n} S_N(\hat{\mathbf{w}}_{m-n}). \quad (6.278)$$

Moreover, this provides a simple stop criterion since the algorithm is stopped when it is not possible to fulfill sa:rjobd anymore. In this form the OBD-procedure becomes a special case of the Statistical Pruning Algorithm given in Sec. 6.7.3.

6.7.6 Optimal Brain Surgeon

The Optimal Brain Surgeon (OBS) pruning procedure [Hassibi & Stork 93] is a recent modification of the OBD-procedure in order to avoid the restrictive diagonal Hessian assumption. The procedure is based on an individual deletion approach and focus on using the LS cost function. It runs as follows:

The OBS-Procedure Choose a filter architecture parameterized by an m -dimensional weight vector, $\mathbf{w}_m = [w_{m,1}, w_{m,2}, \dots, w_{m,m}]^\top$. Initialize $n = 0$. Estimate the weights, i.e., determine $\hat{\mathbf{w}}_{m-n}$. Calculate the $(m-n) \times (m-n)$ Hessian matrix, $\mathbf{H}_N(\hat{\mathbf{w}}_{m-n})$ and the inverted: $\mathbf{H}_N^{-1}(\hat{\mathbf{w}}_{m-n}) = \{P_{ij}\}$, $i, j \in [1; m-n]$. Calculate the modified saliencies according to sa:modsal, i.e.,

$$\tilde{\varrho}_i^{(n)} = \frac{\hat{w}_{m-n,i}^2}{P_{i,i}}. \quad (6.279)$$

If $\tilde{\varrho}_\ell^{(n)} \ll S_N(\hat{\mathbf{w}}_{m-n})$, where $\ell = \arg \min_i \tilde{\varrho}_i^{(n)}$ then delete the weight $w_{m-n,\ell}$ and proceed to *Step 5*; otherwise proceed to *Step 6*. Reestimate all remaining weights within a second order approximation which according to sa:whyp reads⁶⁷:

$$\hat{\mathbf{w}}_{\mathcal{H}} = \hat{\mathbf{w}}_{m-n} - \frac{\hat{w}_{m-n,\ell}}{P_{\ell,\ell}} \cdot [P_{1,\ell}, P_{2,\ell}, \dots, P_{m-n,\ell}]^\top. \quad (6.280)$$

Pick out the weights which are not deleted, i.e.,

$$\hat{\mathbf{w}}_{m-n-1} = [\hat{w}_{\mathcal{H},1}, \dots, \hat{w}_{\mathcal{H},\ell-1}, \hat{w}_{\mathcal{H},\ell+1}, \dots, \hat{w}_{\mathcal{H},m-n}]^\top. \quad (6.281)$$

Increment n and go to *Step 3*. No more weights can be deleted within the second order approximation. Go to *Step 2*; alternatively, stop.

The OBS-procedure may be interpreted as a special variant of the Statistical Pruning Algorithm cf. Sec. 6.7.3. First, it should be noticed that the deletion procedure is based

⁶⁷The restriction matrix is in this case an $m-n$ dimensional row vector:

$$\Xi = [\Xi_1, \Xi_2, \dots, \Xi_{m-n}]$$

where $\Xi_\ell = 1$ and $\Xi_i = 0$, $i \neq \ell$. Furthermore, $\mathbf{a} = \mathbf{0}$.

on a second order approximation of the cost function and furthermore the weights are in *Step 5*. retrained within this approximation. Consequently, these assumptions lead to

$$\mathbf{H}(\mathbf{w}_{\mathcal{H}}) \approx \mathbf{H}(\hat{\mathbf{w}}_m) \quad (6.282)$$

where $\mathbf{w}_{\mathcal{H}}$ is the retrained weight vector with dimension m and n components equal to zero. In [Hassibi & Stork 93] the determination of the Hessian in *Step 3*. is not paid any particular attention; however, using the above fact leads to that (except for $n = 0$) $\mathbf{H}_N(\hat{\mathbf{w}}_{m-n})$ is determined from $\mathbf{H}_N(\hat{\mathbf{w}}_{m-n+1})$ simply by removing the row and column corresponding to the weight currently deleted. Moreover, with the above fact in mind no direct matrix inversion (except for $n = 0$) is required in *Step 3*.. Define the $(m-n) \times (n-m)$ inverse Hessian

$$\mathbf{P} = \mathbf{H}_N^{-1}(\hat{\mathbf{w}}_{m-n}), \quad (6.283)$$

and the $(m-n+1) \times (n-m+1)$ inverse Hessian⁶⁸

$$\tilde{\mathbf{P}} = \{\tilde{P}_{ij}\} = \mathbf{H}_N^{-1}(\hat{\mathbf{w}}_{m-n+1}). \quad (6.284)$$

Furthermore, recall that ℓ is the number of the weight in the $m-n+1$ -dimensional weight vector which currently is deleted. Now using the result given by sa:invhesalt:

$$\mathbf{P} = \begin{bmatrix} \tilde{P}_{1,1} & \cdots & \tilde{P}_{1,\ell-1} & \tilde{P}_{1,\ell+1} & \cdots & \tilde{P}_{1,m-n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \tilde{P}_{\ell-1,1} & \cdots & \tilde{P}_{\ell-1,\ell-1} & \tilde{P}_{\ell-1,\ell+1} & \cdots & \tilde{P}_{\ell-1,m-n+1} \\ \tilde{P}_{\ell+1,1} & \cdots & \tilde{P}_{\ell+1,\ell-1} & \tilde{P}_{\ell+1,\ell+1} & \cdots & \tilde{P}_{\ell+1,m-n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \tilde{P}_{m-n+1,1} & \cdots & \tilde{P}_{m-n+1,\ell-1} & \tilde{P}_{m-n+1,\ell+1} & \cdots & \tilde{P}_{m-n+1,m-n+1} \end{bmatrix} - \tilde{P}_{\ell,\ell}^{-1} \begin{bmatrix} \tilde{P}_{\ell,1} \\ \vdots \\ \tilde{P}_{\ell,\ell-1} \\ \tilde{P}_{\ell,\ell+1} \\ \vdots \\ \tilde{P}_{\ell,m-n+1} \end{bmatrix} \cdot [\tilde{P}_{\ell,1}, \dots, \tilde{P}_{\ell,\ell-1}, \tilde{P}_{\ell,\ell+1}, \dots, \tilde{P}_{\ell,m-n+1}]. \quad (6.285)$$

Due to the fact that all weights are reestimated *every time* a weight has been deleted then

$$\Delta S = \sum_{n=1}^Q \tilde{\sigma}_{\ell}^n \quad (6.286)$$

where Q is the number of deleted weights⁶⁹. That is, the OBS-procedure uses the test statistic valid for complete models in order to decide if the weights can be deleted. In addition, if the increase in error is much smaller than the current training cost (cf. *Step 4*.) then the weight is deleted. This is in keeping with using the reject region of the

⁶⁸Since the Hessian is symmetric, $\tilde{P}_{ij} = \tilde{P}_{ji}$.

⁶⁹Recall that ΔS is the “true” change in cost when imposing the hypothesis.

statistical test cf. e.g., `sa:ineqtest` except that no exact definition of what is meant by “much smaller” is provided. This is, however, clear when using the statistical framework.

`sa:delsobs` furthermore enables an interpretation of the WDV-algorithm which is embedded in the OBS-procedure. At first sight one might think that this WDV-algorithm equals the WDV-algorithm based on individual deletion cf. Sec. 6.7.2.2. However, this is not true due to the fact that the modified saliencies are estimated recursively with an intermediate retraining. Let \mathbf{d}_n^* denote the optimal WDV with dimension n and $\Delta S(\mathbf{d}_n)$ the change in cost when employing a particular WDV, \mathbf{d}_n cf. `sa:deltas`. The set of WDV’s $\mathcal{Q} = \{\mathbf{d}_1^*, \mathbf{d}_2^*, \dots, \mathbf{d}_Q^*\}$ formed by the OBS-procedure is given then given by:

$$\mathbf{d}_1^* = \arg \min_i \tilde{\varrho}_i^1 \quad (6.287)$$

$$\mathbf{d}_n^* = \left\{ \mathbf{d}_n = [(\mathbf{d}_{n-1}^*)^\top, d_{n,n}]^\top \mid \min_{\mathbf{d}_n} \Delta S(\mathbf{d}_n) \right\}, \quad n = 2, 3, \dots, Q. \quad (6.288)$$

That is, the $n - 1$ first components of \mathbf{d}_n^* are fixed at the optimal $(n - 1)$ -dimensional WDV, \mathbf{d}_{n-1}^* , while the last component, $d_{n,n}$, is adjusted so that $\Delta S(\mathbf{d}_n)$ is minimized. This search strategy is obviously local since it goes directly towards the nearest minimum without having the option of changing a previously made choice, i.e., once a weight is chosen for deletion it remains so.

6.8 Procedures for Expanding the Architecture

Fundamentally it is impossible to predict how the filter architecture should be expanded so that the generalization error is diminished without providing assumptions on the system which generated the data, as mentioned in 6.4. The only reliable strategy for a procedure which gradually expands the architecture is therefore to use the basic architecture synthesis algorithm cf. Sec. 6.5.1, i.e.,

Basic Expansion Algorithm
 Select an initial architecture. Choose a number of possible expansions. Determine the expansion under considerations which yields the maximum decrease in the generalization error (according to a chosen generalization error estimator) and accept that particular expansion. Go to *Step 2*.

The obvious inconvenience is that at each stage a – possibly large – number of expansions have to be tested, i.e., estimation of the weights and the generalization error have to be performed. On the other hand, it may still be possible to give suboptimal, consolidated proposals.

In [Hertz et al. 91, Ch. 6.6] different algorithms for expanding a multi-layer feed-forward perceptron neural network (MFPNN) when dealing with binary classification tasks are discussed; however, it does not seem obvious how these algorithms should be modified in order to accomplish the continuous output case under consideration in this Thesis.

Below we focus on various methods applicable for the continuous output case.

6.8.1 Stepwise Forward Inclusion

In the statistical literature concerning linear regression a common method for expanding the architecture is known as Stepwise Forward Inclusion (SFI) [Draper & Smith 81, Ch.

6.11]. SFI is a part of a more general algorithm called Stepwise Regression [Draper & Smith 81, Ch. 6.11] in which the architecture gradually is expanded and pruned. Pruning is done by using Stepwise Backward Elimination which is in its original form⁷⁰ based on hypothesis testing cf. Sec. 6.6.

SFI deals with LX-models, i.e.,

$$\hat{y}(k) = \mathbf{w}_m^\top \mathbf{v}_m(k) + e(k) \quad (6.289)$$

where $\mathbf{v}_m(k)$, \mathbf{w}_m are the m -dimensional regression and weight vector, respectively. It is assumed that a pool of potential regressors exist in advance. Define the vector containing the pool of q possible regressors as $\mathbf{v}_q(k) = [v_1(k), v_2(k), \dots, v_q(k)]^\top$. For instance, consider the case where the p -dimensional input vector $\mathbf{z}(k)$ is determined and $\mathbf{v}_q(k)$ is a l 'th order multidimensional polynomial expansion of $\mathbf{z}(k)$ with $q = (l+p)!/(l!p!)$ cf. Sec. 3.2.1.

Suppose that m regressors already are in the model, i.e., $\mathbf{v}_m = [v_{r_1}, v_{r_2}, \dots, v_{r_m}]^\top$, where $r_i \in [1; q]$ are different indices. The issue under consideration is the inclusion of an extra regressor. In the SFI this issue is solved by comparing the partial correlation coefficients of the regressors which are not included yet. The partial correlation coefficients are defined by:

$$\rho_{y, v_j | \mathbf{v}_m} = \frac{E \{ (y(k) - \langle y(k) \rangle) \cdot (v_j(k) - \langle v_j(k) \rangle) | \mathbf{v}_m(k) \}}{\sqrt{E \{ (y(k) - \langle y(k) \rangle)^2 | \mathbf{v}_m \} E \{ (v_j(k) - \langle v_j(k) \rangle)^2 | \mathbf{v}_m \}}} \quad (6.290)$$

where $j \neq r_i, \forall i$ and $\langle \cdot \rangle$ denotes the time-average.

$\rho_{y, v_j | \mathbf{v}_m}$ measures how much of the variations in the output $y(k)$ which can be explained (linearly) be the regressor $v_j(k)$ when the variation explained by the present regressors is deducted. Consequently, the novel regressor is chosen as: $v_j = \arg \max |\rho_{y, v_j | \mathbf{v}_m}|$. Since only estimates of the partial correlation coefficient are available we furthermore have to check if the correlation is significantly non-zero. This can be formulated as a statistical test cf. [Draper & Smith 81].

An interpretation of this procedure is possible. Suppose that we employ the usual LS cost without regularization and obtain the weight estimate $\hat{\mathbf{w}}_m$ with corresponding cost, $S_N(\hat{\mathbf{w}}_m)$. Now by adding an extra regressor v_j and reestimating then the cost $S_N(\hat{\mathbf{w}}_{m+1})$ is obtained. Maximizing the absolute partial correlation coefficient corresponds to maximizing the decrease in cost, viz. $S_N(\hat{\mathbf{w}}_m) - S_N(\hat{\mathbf{w}}_{m+1})$. To test whether this change is significant one can use `saineqtest` since this is equivalent to testing if the weight associated with the novel regressor is equal to zero.

To sum up, SFI is equivalent to the following procedure:

Stepwise Forward Inclusion Algorithm
Consider the model with m regressors \mathbf{v}_m . Estimate the $q - m$ possible models with one extra regressor added. If any of the potential regressors provides a significant improvement of the cost – i.e., the associated weight is non-zero – then pick the regressor which has the largest improvement.

The real drawback of SFI is that it involves a lot of retraining; however, a scheme based on the partial correlation coefficient which reduces the required computations is mentioned in [Billings & Voon 86].

⁷⁰A Stepwise Regression algorithm based on estimating the generalization error has also been proposed [Leontaritis & Billings 87].

6.8.2 Cascade-Correlation

An expansion procedure which works for MFPNN with linear (continuous) output is The Cascade-Correlation Algorithm [Fahlman & Lebiere 90]. The idea is to add the neurons (perceptrons) one by one so that the output of the most recent neuron has the highest possible correlation with the the error signal. Let us represent the MFPNN by the canonical representation: $\hat{y}(k) = \boldsymbol{\alpha}^\top \mathbf{v}(k)$, where $\mathbf{v}(k) = [v_1(k), v_2(k), \dots, v_q(k)]^\top$, $\boldsymbol{\alpha}$ are the state and weight vectors of the output layer, respectively. The algorithm runs as follows:

Cascade Correlation Algorithm Start with the empty network, i.e., \mathbf{v} has the dimension 0. and set the counter $q = 0$. Add a new neuron which receives input from all previous neurons and from the input vector $\mathbf{z}(k)$. That is, let

$$v_{q+1}(k) = h\left(\mathbf{w}_1^\top \mathbf{z}(k) + \mathbf{w}_2^\top \mathbf{v}(k)\right) \quad (6.291)$$

where $\mathbf{w}_1, \mathbf{w}_2$ represent weights associated with the neuron and $h(\cdot)$ is the activation function. Estimate \mathbf{w}_1 and \mathbf{w}_2 (see [Fahlman & Lebiere 90]) so that

$$[\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2] = \arg \max |\hat{\gamma}| \quad (6.292)$$

where $\hat{\gamma}$ is the estimated crosscovariance function between $v_{q+1}(k)$ and the error signal $e(k) = y(k) - \hat{\boldsymbol{\alpha}}^\top \mathbf{v}(k)$ given by

$$\hat{\gamma} = \frac{1}{N} \sum_{k=1}^N (v_{q+1}(k) - \langle v_{q+1}(k) \rangle) \cdot (e(k) - \langle e(k) \rangle). \quad (6.293)$$

Here $\langle \cdot \rangle$ is the time-average. Expand the \mathbf{v} and $\boldsymbol{\alpha}$ vectors, i.e.,

$$\mathbf{v}(k) \leftarrow \left[\mathbf{v}^\top(k), v_{q+1}(k) \right]^\top, \quad (6.294)$$

$$\boldsymbol{\alpha} \leftarrow \left[\boldsymbol{\alpha}^\top(k), \alpha_{q+1} \right]^\top, \quad (6.295)$$

and increment the counter q . Estimate the $\boldsymbol{\alpha}$ according to the cost function, i.e.,

$$\hat{\boldsymbol{\alpha}} = \arg \min C_N(\boldsymbol{\alpha}). \quad (6.296)$$

Stop if the cost is acceptably small; otherwise, go to *Step 2*.

The number of layers in the network constructed by this algorithm is thus equal to the number of neurons and each neuron is connected with the output. Notice (cf. *Step 3*) that the weights associated with a particular neuron are estimated only once while the α -weights are reestimated each time a neuron is added.

The idea behind Cascade Correlation resembles the Stepwise Forward Inclusion procedure mentioned above since the new regressor, i.e., the new neuron, is added so that it describes as much as possible of the variation in the output which is not already described by the present neurons. This is explicitly done by maximizing the correlation with the error signal which expresses the lack of description. However, to implement the idea of SFI completely one should include a novel neuron and then estimate *all weights* in the network and finally check if the cost drops significantly. Since only the weights in the output

layer are reestimated every time a new neuron is added then one may expect the resulting network to be suboptimal and typically too large. Consequently, the Cascade Correlation algorithm may be viewed as a particular training algorithm which estimates the weight successively. This is possibly important when dealing with large and deep networks which are hard to train due to the fact that the weights are highly interacting.

6.8.3 Statistical Expansion Test

In [White 89b] the problem of expanding a 2-layer MFPNN by adding hidden neurons is given a statistical formulation. The 2-layer $[p, m_1, 1]$ -network ($m_2 = 1$) model obeys:

$$\begin{aligned} y(k) &= f_n(\mathbf{z}(k); \mathbf{w}_m) + e(k; \mathbf{w}_m) \\ &= \mathbf{W}^{(2)} \mathbf{h} \left(\mathbf{W}^{(1)} \mathbf{s}^{(0)}(k) \right) + e(k; \mathbf{w}_m) \end{aligned} \quad (6.297)$$

where

- Step 4** • $\mathbf{W}^{(r)}$ are weight matrices with dimension $m_r \times (m_{r-1} + 1)$.
- $\mathbf{h}(\cdot)$ is the vector activation function.
 - $\mathbf{s}^{(0)}(k) = [1, \mathbf{z}^\top(k)]^\top$ is the $p + 1$ dimensional augmented input vector.
 - \mathbf{w}_m is the m -dimensional weight vector, $m = m_1(p + 2) + 1$, containing all weights in the network.
 - $e(k; \mathbf{w}_m)$ is the error signal.

The problem under consideration is whether the generalization error can be reduced by adding extra hidden neurons, i.e., by increasing m_1 . Formally we want to test if the current model is complete when using the optimal weights, \mathbf{w}_m^* , i.e., cf. Def. 6.3 the hypothesis, \mathcal{H}_1 and the alternative $\mathcal{H}_{1,a}$ become:

$$\begin{aligned} \mathcal{H}_1 : \quad & \forall \mathbf{z}, g_n(\mathbf{z}) = f_n(\mathbf{z}; \mathbf{w}_m^*) \\ \mathcal{H}_{1,a} : \quad & \forall \mathbf{z}, g_n(\mathbf{z}) \neq f_n(\mathbf{z}; \mathbf{w}_m^*) \end{aligned} \quad (6.298)$$

In the rest of the section the statistical framework presented in Sec. 6.6 is employed rather than proceeding as in [White 89b]; however, the final result is the same.

If \mathcal{H}_1 is true then adding extra neurons will not cause any changes. That is, the optimal weights attached to the added neurons equal zero. However, the hypothesis concerning a number of irrelevant neurons is irregular cf. Sec. 6.6.4. Consequently, we draw on the methodology presented in Sec. 6.6.4.

Form Q fixed hidden neurons with the Q -dimensional response vector

$$\boldsymbol{\zeta}(k) = \mathbf{h} \left(\boldsymbol{\Omega} \mathbf{s}^{(0)}(k) \right) \quad (6.299)$$

where $\boldsymbol{\Omega}$ is a $Q \times (p+1)$ weight matrix chosen so that the weight space is properly covered⁷¹. For instance, one may chose $\boldsymbol{\Omega}$ randomly. Next consider the expanded model

$$\begin{aligned} y(k) &= f_n(\mathbf{z}(k); \mathbf{w}_q) + e(k; \mathbf{w}_q) \\ &= \mathbf{W}^{(2)} \mathbf{h} \left(\mathbf{W}^{(1)} \mathbf{s}^{(0)}(k) \right) + \boldsymbol{\omega}^\top \boldsymbol{\zeta}(k) + e(k; \mathbf{w}_q) \end{aligned} \quad (6.300)$$

⁷¹Recall according to Sec. 6.6.4 that a principal component analysis of $\boldsymbol{\zeta}$ may reduce the dimension (i.e., it becomes less than Q) and thereby improve the power of the test.

where $\boldsymbol{\omega}$ is a Q -dimensional weight vector which determines the contributions from the fixed neurons. All weights are assembled in the $q = m + Q$ dimensional weight vector $\boldsymbol{w}_q = [\boldsymbol{w}_m^\top, \boldsymbol{\omega}^\top]$. The hypothesis (and alternative) to be tested is⁷²:

$$\begin{aligned} \mathcal{H}_2 : \boldsymbol{\omega}^* &= \mathbf{0} \\ \mathcal{H}_{2,a} : \boldsymbol{\omega}^* &\neq \mathbf{0} \end{aligned} \quad (6.301)$$

Since the original hypothesis \mathcal{H}_1 impose that the model is complete then two facts should be noted:

- No regularization is possible, i.e., the usual LS cost function, $S_N(\cdot)$, is employed.
- The reject region given by sa:ineqtest is applicable.

The hypothesis is in terms of \boldsymbol{w}_q given by:

$$\boldsymbol{\Xi} \boldsymbol{w}_q^* = \mathbf{0} \quad (6.302)$$

where $\boldsymbol{\Xi} = [\mathbf{0}_{Q,m}, \mathbf{I}_{Q,Q}]$.

Now according to sa:ineqtest the reject region becomes:

$$\Delta S > \chi_{1-\alpha}^2(Q) \frac{S_N(\hat{\boldsymbol{w}}_q)}{N - q} \quad (6.303)$$

where $\Delta S = S_N(\hat{\boldsymbol{w}}_m) - S_N(\hat{\boldsymbol{w}}_q)$ and $\chi_{1-\alpha}^2(Q)$, $\hat{\boldsymbol{w}}_m$, $\hat{\boldsymbol{w}}_q$ are the estimated weight vectors of the original and the expanded model, respectively. Furthermore, $\chi_{1-\alpha}^2(Q)$ is the $(1 - \alpha)$ -fractile of the chi-square distribution with Q degrees of freedom.

The estimated weights of the original model, $\hat{\boldsymbol{w}}_m$, are supposed to be estimated already. Further the weight estimates of the expanded model, $\hat{\boldsymbol{w}}_q$, is required. It is possible to find $\hat{\boldsymbol{w}}_q$ without a fully retraining due to the fact that the test procedure is based on a second order Taylor series expansion of the cost function. That is, define

$$\hat{\boldsymbol{w}}_{q,\mathcal{H}} = [\hat{\boldsymbol{w}}_m^\top, \mathbf{0}^\top]^\top, \quad (6.304)$$

then the second order approximation of the cost function regarding the expanded model becomes:

$$S_N(\boldsymbol{w}_q) = S_N(\hat{\boldsymbol{w}}_{q,\mathcal{H}}) + \frac{\partial S_N(\boldsymbol{w}_{q,\mathcal{H}})}{\partial \boldsymbol{w}_q^\top} (\boldsymbol{w}_q - \hat{\boldsymbol{w}}_{q,\mathcal{H}}) + (\boldsymbol{w}_q - \hat{\boldsymbol{w}}_{q,\mathcal{H}})^\top \mathbf{H}_{q,q}(\hat{\boldsymbol{w}}_{q,\mathcal{H}}) (\boldsymbol{w}_q - \hat{\boldsymbol{w}}_{q,\mathcal{H}}) \quad (6.305)$$

where $\mathbf{H}_{q,q}(\hat{\boldsymbol{w}}_{q,\mathcal{H}})$ is the Hessian defined in sa:jndef and $S_N(\hat{\boldsymbol{w}}_{q,\mathcal{H}}) = S_N(\hat{\boldsymbol{w}}_m)$. The estimated weights, $\hat{\boldsymbol{w}}_q$, result by solving the equation

$$\frac{\partial S_N(\hat{\boldsymbol{w}}_q)}{\partial \boldsymbol{w}_q} = \mathbf{0}. \quad (6.306)$$

⁷²Note that \mathcal{H}_2 is an implication of \mathcal{H}_1 . Consequently, if \mathcal{H}_2 is rejected then also \mathcal{H}_1 is rejected. On the other hand, accept of \mathcal{H}_2 does not necessarily imply accept of \mathcal{H}_1 . Intuitively this is clear: We only test if inclusion of extra hidden neurons does not improve the model. However, the model may still be improved by including other types of nonlinear terms, e.g., the consideration of another type of hidden neurons.

Simple linear algebra then gives:

$$\begin{aligned}\hat{\mathbf{w}}_q &= \hat{\mathbf{w}}_{q,\mathcal{H}} - \frac{1}{2} \mathbf{H}_{q,q}^{-1}(\hat{\mathbf{w}}_{q,\mathcal{H}}) \cdot \frac{\partial S_N(\mathbf{w}_{q,\mathcal{H}})}{\partial \mathbf{w}_q} \\ &= \hat{\mathbf{w}}_{q,\mathcal{H}} + \mathbf{H}_{q,q}^{-1}(\hat{\mathbf{w}}_{q,\mathcal{H}}) \cdot \nabla(\hat{\mathbf{w}}_{q,\mathcal{H}})\end{aligned}\quad (6.307)$$

where the gradient is given by

$$\nabla(\hat{\mathbf{w}}_{q,\mathcal{H}}) = -\frac{1}{2} \frac{\partial S_N(\hat{\mathbf{w}}_{q,\mathcal{H}})}{\partial \mathbf{w}_q} = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}_{q,\mathcal{H}}) e(k; \hat{\mathbf{w}}_{q,\mathcal{H}}), \quad (6.308)$$

and the instantaneous gradient vector by

$$\boldsymbol{\psi}(k; \mathbf{w}_{q,\mathcal{H}}) = \frac{\partial f_n(\mathbf{z}(k), \hat{\mathbf{w}}_{q,\mathcal{H}})}{\partial \mathbf{w}_q} = \left[\frac{\partial f_n(\mathbf{z}(k), \hat{\mathbf{w}}_m)}{\partial \mathbf{w}_m^\top}, \boldsymbol{\zeta}^\top(k) \right]^\top. \quad (6.309)$$

Thus all terms involved depend only on quantities which are derived from the estimation of $\hat{\mathbf{w}}_m$. The inverse Hessian may be calculated with a minimum of computations. Consider the following partition:

$$\mathbf{H}_{q,q}(\hat{\mathbf{w}}_{q,\mathcal{H}}) = \begin{bmatrix} \mathbf{H}_{m,m}(\hat{\mathbf{w}}_m) & \mathbf{H}_{m,Q}(\hat{\mathbf{w}}_{q,\mathcal{H}}) \\ \mathbf{H}_{m,Q}^\top(\hat{\mathbf{w}}_{q,\mathcal{H}}) & \mathbf{H}_{Q,Q}(\hat{\mathbf{w}}_{q,\mathcal{H}}) \end{bmatrix}. \quad (6.310)$$

In particular, $\mathbf{H}_{Q,Q}(\hat{\mathbf{w}}_{q,\mathcal{H}}) = \boldsymbol{\zeta}(k) \boldsymbol{\zeta}^\top(k)$, and $\mathbf{H}_{m,m}(\hat{\mathbf{w}}_m)$ is the Hessian of the original model. When using a second order minimization procedure the inverse Hessian, $\mathbf{H}_{m,m}^{-1}(\hat{\mathbf{w}}_m)$, is an immediate spinoff. According to [Seber & Wild 89, App. A3] $\mathbf{H}_{q,q}^{-1}$ (the dependence on the weights is omitted for simplicity) is given by:

$$\mathbf{H}_{q,q}^{-1} = \begin{bmatrix} \mathbf{H}_{m,m}^{-1} + \mathbf{B}_{m,Q} \mathbf{B}_{Q,Q}^{-1} \mathbf{B}_{m,Q}^\top & -\mathbf{B}_{m,Q} \mathbf{B}_{Q,Q}^{-1} \\ -\mathbf{B}_{Q,Q}^{-1} \mathbf{B}_{m,Q}^\top & \mathbf{B}_{Q,Q}^{-1} \end{bmatrix} \quad (6.311)$$

where the auxiliary matrices are defined by:

$$\mathbf{B}_{m,Q} = \mathbf{H}_{m,m}^{-1} \mathbf{H}_{m,Q}, \quad (6.312)$$

$$\mathbf{B}_{Q,Q} = \mathbf{H}_{Q,Q} - \mathbf{H}_{m,Q}^\top \mathbf{H}_{m,m}^{-1} \mathbf{H}_{m,Q}. \quad (6.313)$$

That is, the only extra matrix inversion involved is the calculation of $\mathbf{B}_{Q,Q}^{-1}$.

Now, what remains in order to calculate the rejection region `sa:rejttest` is the evaluation of ΔS and $S_N(\hat{\mathbf{w}}_q)$. Due to the relation: $S_N(\hat{\mathbf{w}}_q) = S_N(\hat{\mathbf{w}}_m) - \Delta S$ we focus on the evaluation of ΔS only. Substituting `sa:expanest` into `sa:expanctst` and performing simple algebraic manipulations yield:

$$\Delta S = \nabla^\top(\hat{\mathbf{w}}_{q,\mathcal{H}}) \mathbf{H}_{q,q}^{-1}(\hat{\mathbf{w}}_{q,\mathcal{H}}) \nabla(\hat{\mathbf{w}}_{q,\mathcal{H}}). \quad (6.314)$$

To sum up, we now have:

Statistical Expansion AlgorithmStart by estimating a filter with one neuron⁷³. Generate a random $Q \times (p + 1)$ weight matrix $\boldsymbol{\Omega}$, and form the response vector, $\boldsymbol{\zeta}(k)$ of Q fixed neurons. Perform

⁷³In [White 89b] the first neuron is supposed to be linear.

a PCA-analysis of $\zeta(k)$ and let $\zeta'(k)$ denote the Q' -dimensional ($Q' \leq Q$) vector of significant principal components. Test the hypothesis that the fixed neurons (i.e., the vector $\zeta'(k)$) contribute significantly, according to sa:rejtest. If the hypothesis is rejected then include an extra neuron, reestimate all weights and go to *Step 2.* Otherwise, stop.

Finally, the close relationship with the SFI-algorithm (see above) should be noted. In fact, the Q fixed neurons acts as a pool of potential regressors.

6.8.4 Partition Function Filters

Consider the partition function filter (PFF) which in the canonical filter representation, cf. Sec. 3.1.2, is given by⁷⁴

$$\hat{y}(k) = \sum_{i=1}^q \alpha_i \cdot b_i(\mathbf{z}(k); \beta_i) \cdot D_i(\mathbf{z}(k)) \quad (6.315)$$

$$= \sum_{i=1}^q \hat{y}_i(k; \mathbf{w}_i) \quad (6.316)$$

where

Step 4 • $\mathbf{z}(k) \in \mathcal{D}$ is the p -dimensional input vector.

- $b_i(\mathbf{z}(k); \beta_i)$ is the i 'th basis function.
- α_i and β_i are the weights associated with $b_i(\cdot)$.
- $D_i(\mathbf{z})$ is the domain function given by:

$$D_i(\mathbf{z}(k)) = \begin{cases} 1 & , \mathbf{z}(k) \in \mathcal{D}_i \\ 0 & , \text{otherwise} \end{cases} \quad (6.317)$$

where \mathcal{D}_i is the i 'th partition. The partitions are supposed to be disjoint and consequently, the input domain is given by: $\mathcal{D} = \bigcup_{i=1}^q \mathcal{D}_i$.

- $\hat{y}_i(k; \mathbf{w}_i)$ is the output of the i 'th *local filter* parameterized by $\mathbf{w}_i = [\alpha_i, \beta_i^\top]^\top$.

Let \mathbf{w} be the vector of all weights in the PFF, i.e., $\mathbf{w} = [\mathbf{w}_1^\top, \mathbf{w}_2^\top, \dots, \mathbf{w}_q^\top]^\top$. Moreover, let $p_{\mathbf{z}}(\mathbf{z})$, $p_\varepsilon(\varepsilon)$ denote the p.d.f.'s of the input and the inherent noise, respectively. Now the generalization error cf. sa:gentrue of the PFF is given by:

$$\begin{aligned} G(\hat{\mathbf{w}}) &= E_{\mathbf{z}, \varepsilon} \left\{ (y - \hat{y})^2 \right\} \\ &= \int_{\mathcal{D}} \left(y - \sum_{i=1}^q \hat{\alpha}_i \cdot b_i(\mathbf{z}(k); \hat{\beta}_i) \cdot D_i(\mathbf{z}(k)) \right)^2 p_{\mathbf{z}}(\mathbf{z}) p_\varepsilon(\varepsilon) d\mathbf{z} d\varepsilon \\ &= \sum_{i=1}^q \int_{\mathcal{D}_i} \left(y - \hat{\alpha}_i \cdot b_i(\mathbf{z}(k); \hat{\beta}_i) \right)^2 p_{\mathbf{z}}(\mathbf{z}) p_\varepsilon(\varepsilon) d\mathbf{z} d\varepsilon. \end{aligned} \quad (6.318)$$

⁷⁴Here the bias weight α_0 is excluded.

The generalization error is thus simply the sum of the local generalization errors, $G_i(\hat{\mathbf{w}}_i)$, defined by⁷⁵:

$$G_i(\hat{\mathbf{w}}_i) = \int_{\mathcal{D}_i} \left(y - \hat{\alpha}_i \cdot b_i(\mathbf{z}(k); \hat{\boldsymbol{\beta}}_i) \right)^2 p_{\mathbf{z}}(\mathbf{z}) p_{\varepsilon}(\varepsilon) d\mathbf{z} d\varepsilon. \quad (6.319)$$

The average generalization error $\Gamma = E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\}$ is the average of the generalization error w.r.t. the training set. That is, we average over fluctuation in the weight estimate. Now suppose that the LS cost function $S_N(\mathbf{w})$ is employed. Evaluation of the cost function gives:

$$\begin{aligned} S_N(\mathbf{w}) &= \frac{1}{N} \sum_{k=1}^N (y(k) - \hat{y}(k))^2 \\ &= \frac{1}{N} \sum_{k=1}^N \left(y(k) - \sum_{i=1}^q \hat{\alpha}_i \cdot b_i(\mathbf{z}(k); \hat{\boldsymbol{\beta}}_i) \cdot D_i(\mathbf{z}(k)) \right)^2 \\ &= \sum_{i=1}^q \frac{1}{N} \sum_{\mathcal{K}_i} \left(y(k) - \hat{\alpha}_i \cdot b_i(\mathbf{z}(k); \hat{\boldsymbol{\beta}}_i) \right)^2 \end{aligned} \quad (6.320)$$

where

$$\mathcal{K}_i = \{k : \mathbf{z}(k) \in \mathcal{D}_i\} \quad (6.321)$$

Obviously, $\bigcup_{i=1}^q \mathcal{K}_i = \{1, 2, \dots, N\}$ and $\mathcal{K}_{r_1} \cap \mathcal{K}_{r_2} = \emptyset$. Now define the local cost function:

$$S_{N,i}(\mathbf{w}_i) = \frac{1}{N} \sum_{\mathcal{K}_i} \left(y(k) - \hat{\alpha}_i \cdot b_i(\mathbf{z}(k); \hat{\boldsymbol{\beta}}_i) \right)^2. \quad (6.322)$$

Clearly, the estimated weights $\hat{\mathbf{w}}_i = \arg \min_{\mathbf{w}_i} S_{N,i}(\mathbf{w}_i)$ are identical to the corresponding part of the vector $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} S_N(\mathbf{w})$. That is, there is no dependence among the estimates, $\hat{\mathbf{w}}_i$, $i = 1, 2, \dots, q$. In consequence the average generalization error also splits into a sum

$$\Gamma = \sum_{i=1}^q E_{\mathcal{T}} \{G_i(\hat{\mathbf{w}}_i)\} = \sum_{i=1}^q \Gamma_i. \quad (6.323)$$

Two circumstances should be noted:

- The above result rely on employing the LS cost function. However, it is still possible to include a regularization term of the form:

$$R_N(\mathbf{w}) = \sum_{i=1}^q R_{N,i}(\mathbf{w}_i) D_i(\mathbf{z}(k)). \quad (6.324)$$

That is, the regularization is required to be local.

- The shape of the partitions \mathcal{D}_i is, in principle, assumed to be fixed. However, in practice the shapes are – to some extend – dependent on the training data. That means, the sum of local average generalization errors does not equal Γ . Studying the effects of this dependence is extremely complicated and is left for future studies.

⁷⁵If we deal with a localized covering architecture in which the the partitions, \mathcal{D}_i , are not disjoint we may expect that the decomposition of the generalization error into local generalization errors is a valid approximation provided that the interactions among the domain functions are small. That is, if the value of $D_i(\cdot)$ is large all the other domain functions are supposed to have small values.

In principle, the PFF acts like an ensemble of local, independent filters, \hat{y}_i which collaborate in order to form the filter output $\hat{y}(k)$. Furthermore, each local filter is encumbered with a local average generalization error. The task is now to suggest a strategy for gradually expanding the PFF. We focus on adding local filters one by one. A straight forward strategy thus consists in comparing the individual Γ_i 's, and subsequently, divide the local filter with maximum Γ_i into two local filters.

Let us elaborate on various facts which motivate this strategy. Recall cf. Sec. 6.3.4 that the average generalization error obeys the decomposition:

$$\Gamma = \sigma_\varepsilon^2 + MSME + WFP \quad (6.325)$$

where σ_ε^2 is the inherent noise variance, $MSME$ is the mean square model error, and WFP is the weight fluctuation penalty.

σ_ε^2 is normally assumed to be independent of the input, \mathbf{z} . Hence, the inherent noise contributes uniformly to the individual Γ_i and consequently, the inherent noise does not affect the suggested strategy.

The $MSME$ describes the systematic error done when employing an infinite training and decreases with increasing model complexity, i.e., increasing q . The WFP explains the effects of estimating the model from a finite training set. Normally it increases with increasing model complexity; however, the increase is not necessarily monotonous.

When q is small the $MSME$ – compared to the WFP – will be the dominating term, i.e., a large Γ_i indicates a large model error. A natural strategy is thus to divide the local filter, say $\hat{y}_r(k)$, where $r = \arg \max \Gamma_i$ into two new filters, say $\hat{y}_{r_1}(k)$, $\hat{y}_{r_2}(k)$. That is, the partition \mathcal{D}_r is divided into the disjoint partitions, \mathcal{D}_{r_1} , \mathcal{D}_{r_2} so that $\mathcal{D}_{r_1} \cup \mathcal{D}_{r_2} = \mathcal{D}_r$. It is ensured that $\Gamma_{r_1} + \Gamma_{r_2} \leq \Gamma_r$ if $\exists \mathbf{w}_{r_1}, \mathbf{w}_{r_2}$ so that $\hat{y}_r(k; \hat{\mathbf{w}}_r) \equiv \hat{y}_{r_1}(k; \mathbf{w}_{r_1}) + \hat{y}_{r_2}(k; \mathbf{w}_{r_2})$. Here $\hat{\mathbf{w}}_r$ are the estimated weights of the local filter $\hat{y}_r(k)$. This requirement may be met by employing the same basis function, $b(\cdot)$ within all local filters, i.e., $b_i(\cdot) = b(\cdot)$.

The next problem is how the division of \mathcal{D}_i should be performed. According to the discussion in Sec. 6.4 an optimal solution to this problem requires knowledge of the true system; consequently, we resort to a heuristic method. Consider the case where the partitions are separated by a number of hyperplanes cf. Sec. 3.1.2. The partitions, \mathcal{D}_{r_1} , \mathcal{D}_{r_2} , result from dividing \mathcal{D}_r by a hyperplane

$$\mathcal{H}: \quad \mathbf{a}^\top \mathbf{z}(k) - t = 0 \quad (6.326)$$

where \mathbf{a} is the p -dimensional normal vector and t the threshold. The normal vector is – for lack of anything better – selected according to a random procedure. The threshold is next adjusted so that the number of data samples within the new partitions is equal, q.e., $\mathcal{K}_{r_1} = \mathcal{K}_{r_2} = \mathcal{K}_r/2$. This ensures that none of the partitions contains an inexpedient low number of data samples which gives rise to high WFP . Secondly, the area of a particular partition will become inversely proportional to the probability that an input vector belongs to the partition. Hence, data are not wasted on achieving a close fit in regions of the input space where the p.d.f. takes small values⁷⁶. In order to accomplish the above claim, t , is set according to:

$$t = \frac{1}{|\mathcal{K}_r|} \sum_{\mathcal{K}_r} \mathbf{a}^\top \mathbf{z}(k). \quad (6.327)$$

⁷⁶Note that the mean square error is the integral of the squared error *weighted* with the p.d.f. of the input.

Here $|\mathcal{K}_r|$ is the number of data samples in \mathcal{K}_r . $\zeta(k) = \mathbf{a}^\top \mathbf{z}(k)$ defines the projection onto the one-dimensional space pointed out by the normal vector \mathbf{a} . The threshold is determined as the time-average of $\zeta(k)$ which ensures that the partitions contain approximately the same number of data samples. The determination of t is depicted in Fig. 6.8.

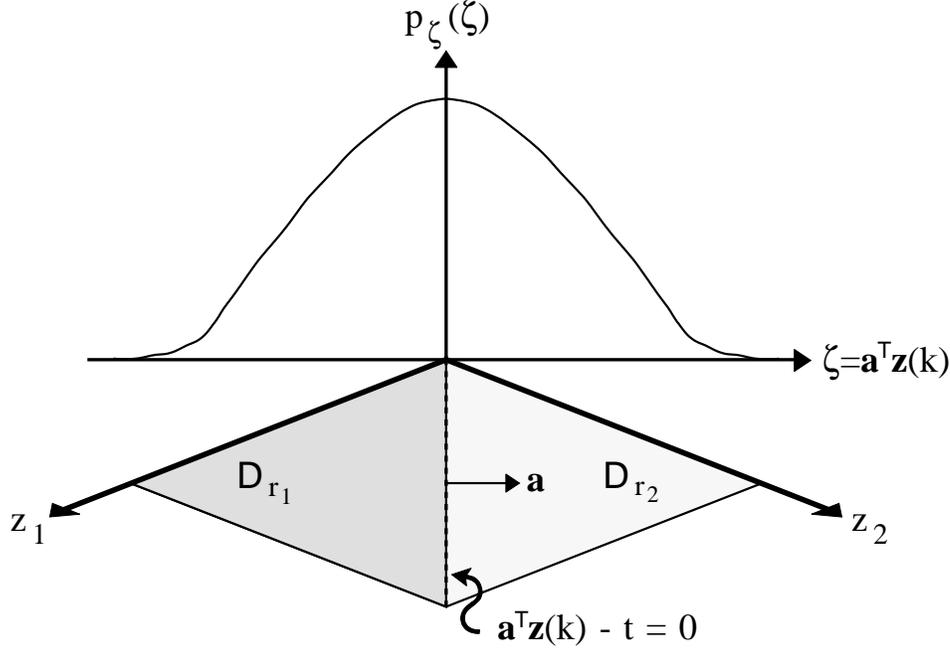


Figure 6.8: Determination of the threshold t . $p_\zeta(\zeta)$ is the p.d.f. of $\zeta = \mathbf{a}^\top \mathbf{z}$.

When q gets larger Γ will be dominated by the *WFP* term. That is, variations in the *MSME* over the input space will be less noticeable; consequently, it becomes more doubtful whether the procedure results in a proper expansion. It is obvious that the expansion procedure should be attended by a reestimation⁷⁷ of Γ so that expansions which result in increasing average generalization error are ignored.

Let us summarize the suggested procedure by formulating an algorithm concerning the gate function filter given by:

$$\hat{y}(k) = \sum_{i=1}^q \alpha_i b(\mathbf{z}(k); \boldsymbol{\beta}_i) \cdot D_i(\mathbf{z}(k)). \quad (6.328)$$

The gates, \mathcal{D}_i , are delimited by hyperplanes which are parallel to co-ordinate axes, i.e., the i 'th hyperplane is given by the equation:

$$z_{j_i} - t_i = 0 \quad (6.329)$$

where j_i , t_i , are the splitting direction and the threshold, respectively.

⁷⁷Note that the change in the average generalization error equals: $\Gamma_r - (\Gamma_{r_1} + \Gamma_{r_2})$.

Gate Function Filter Synthesis Algorithm

- Step 1*
- a. Select a basis function, $b(\cdot)$, and choose a method for estimating the generalization error.
 - b. Initialize the counter, $q = 1$. Further set the first gate equal to the input domain, i.e., $\mathcal{D}_1 = \mathcal{D}$.
 - c. Estimate the weights, $\hat{\mathbf{w}}_1$, and calculate the generalization error estimate, \hat{G}_1 .
- Step 2* Rank the generalization error estimates:

$$\hat{G}_{i_q} \geq \hat{G}_{i_{q-1}} \geq \dots \geq \hat{G}_{i_1} \quad (6.330)$$

where $i_{s_1} \neq i_{s_2}$, $i_s \in [1; q]$.

- Step 3* **for** $r = i_q, i_{q-1}, \dots, i_1$
- a. Select the splitting direction, j_r , randomly among the numbers $1, 2, \dots, p$. Next calculate the threshold, t_r , as the time-average of $z_{j_r}(k)$ within the gate \mathcal{D}_r according to `sa:threshold`.
 - b. Add the novel local filters, $\hat{y}_{r_1}(k)$, $\hat{y}_{r_2}(k)$ and estimate the weights, \mathbf{w}_{r_1} , \mathbf{w}_{r_2} .
 - c. Calculate the generalization error estimates: \hat{G}_{r_1} , \hat{G}_{r_2} . If $\hat{G}_{r_1} + \hat{G}_{r_2} < \hat{G}_r$ then accept the expansion, increment q , and go to *Step 2*. Otherwise, if not all p splitting directions have been tried then goto *Step 3a*.

Two circumstances should be emphasized in connection with this algorithm:

- In a practical implementation this algorithm is efficiently implemented according to the tree-structured approach discussed in Sec. 3.1.2.
- A pruning procedure is in fact embedded in the expansion procedure since (cf. *Step 3c*.) the expansion is accepted only when the generalization error decreases. A possible extension consists in employing other pruning procedures cf. Sec. 6.7.

In [Hoffmann 92a, Ch. 5] similar algorithms are suggested. These algorithms deal with the so-called zero and first order gate function filters which are characterized by possessing the basis functions, $b(\mathbf{z}(k)) = 1$, $b(\mathbf{z}(k)) = \mathbf{z}(k)$, respectively. The approach in [Hoffmann 92a] is based on the decomposition of the cost function, cf. `sa:costdecomp`, rather than using the decomposition of the average generalization error in `sa:moderr`. However, recall that the training cost is a simple estimate of the generalization error.

6.9 Reducing Generalization Error by Regularization

Both experimental [Bishop 91], [Weigend et al. 90] and theoretical [Geman et al. 92], [Krogh & Hertz 91], [Seber & Wild 89, Ch. 9.5.2] studies have shown that the generalization ability can be increased by adding a regularization term to the cost function. Recall cf. Ch. 5 that the cost function, $C_N(\mathbf{w})$, in this case becomes:

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa R_N(\mathbf{w}) \quad (6.331)$$

where $S_N(\mathbf{w})$ is the usual cost function – in this work – the LS cost function, $\kappa \geq 0$ is the regularization parameter, and $R_N(\mathbf{w})$ is the regularization term.

In the discussion concerning the model error decomposition Sec. 6.3.4 it was demonstrated that the use of a $\kappa > 0$ implies that the *MSME* is increased. On the other hand, the *WFP* may decrease. Consider the case of a large κ and $R_N(\mathbf{w}) = r(\mathbf{w})$, i.e., a regularization term which is independent on the training set. Now, the weights will – to a large extent – be adjusted so that $r(\mathbf{w})$ becomes small. Hence, as $r(\mathbf{w})$ does not depend on the training set the fluctuations in the estimates will be small, and consequently, the *WFP* is reduced. This result is a trade off between *MSME* and *WFP*; hence, an optimal value of κ exists. In that respect κ can be regarded as an architecture parameter which adjusts the effective number of degrees of freedom since the weights are not allowed to move freely in the weights space; they are restricted to the region where $r(\mathbf{w})$ is small. In general it is, of course, impossible state any result concerning the optimal κ since it depends on the structure of the system and the architecture of the model. Consequently, one may rely on suboptimal methods. A straight forward recipe is to vary κ so that some generalization error estimate (see Sec. 6.5) is minimized.

Below a simple case – inspired by [Krogh & Hertz 91] – is treated in order to substantiate the mentioned trade off, and to clarify the parameters entering the expression for the optimal value of κ . In addition, the example may provide some guidelines when dealing with more complicated situations.

Consider the linear system:

$$y(k) = \mathbf{z}^\top(k) \mathbf{w}^\circ + \varepsilon(k) \quad (6.332)$$

where $\mathbf{z}(k)$ is the i.i.d. m -dimensional stochastic input vector signal, \mathbf{w}° is the m -dimensional true weight vector, and $\varepsilon(k)$ is the i.i.d. zero mean inherent noise with variance σ_ε^2 . It is assumed that $\varepsilon(k)$ is independent of $\mathbf{z}(k)$. The model is complete and given by:

$$y(k) = \mathbf{z}^\top(k) \mathbf{w} + e(k; \mathbf{w}) \quad (6.333)$$

where $e(k; \mathbf{w})$ is the error signal and \mathbf{w} the m -dimensional weight vector. In order to estimate the model from the training set, $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, we employ the LS cost function, $S_N(\mathbf{w})$, with a *weight decay regularizer*, i.e., the cost function becomes:

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa \mathbf{w}^\top \mathbf{w}. \quad (6.334)$$

The average generalization error, Γ , obeys the model error decomposition (see Sec 6.3.4)

$$\Gamma = \sigma_\varepsilon^2 + MSME(\kappa) + WFP(\kappa) \quad (6.335)$$

where the dependence on κ is explicitly emphasized. In App. G a complete derivation of $MSME(\kappa)$ $WFP(\kappa)$ is provided. The results are:

$$MSME(\kappa) = \sum_{i=1}^m \frac{(\omega_i^\circ)^2 \kappa^2 \lambda_i}{(\lambda_i + \kappa)^2} \quad (6.336)$$

$$WFP(\kappa) = \frac{1}{N} \cdot \text{tr} \left[\kappa^2 \mathbf{J}^{-1} \mathbf{K} \mathbf{J}^{-1} \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \right] + \frac{\sigma_\varepsilon^2}{N} \sum_{i=1}^m \frac{\lambda_i^2}{(\lambda_i + \kappa)^2} \quad (6.337)$$

where

- λ_i is the i 'th eigenvalue of the (by assumption) positive definite Hessian matrix $\mathbf{H} = E\{\mathbf{z}(k)\mathbf{z}(k)\}$.
- ω_i° is the i 'th transformed true weight defined by:

$$\boldsymbol{\omega}^\circ = [\omega_1^\circ, \omega_2^\circ, \dots, \omega_m^\circ]^\top = \mathbf{Q}^\top \mathbf{w}^\circ \quad (6.338)$$

where \mathbf{Q} is the unitary matrix of normalized eigenvectors of \mathbf{H} ,

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m] \quad (6.339)$$

with \mathbf{q}_i being the i 'th eigenvector.

- \mathbf{K} is the fourth order statistic:

$$\mathbf{K} = E_{\mathcal{T}} \left\{ \left(\mathbf{z}(k)\mathbf{z}^\top(k) - \mathbf{H} \right) \mathbf{J}^{-1} \mathbf{H} \mathbf{J}^{-1} \left(\mathbf{z}(k)\mathbf{z}^\top(k) - \mathbf{H} \right) \right\}. \quad (6.340)$$

In order to find κ_{opt} which minimizes the average generalization error and thus trades off *MSME* and *WFP* the sum of the partial derivatives w.r.t. κ is set equal to zero, i.e.,

$$\frac{\partial MSME}{\partial \kappa} + \frac{\partial WFP}{\partial \kappa} = 0. \quad (6.341)$$

Note that $WFP \rightarrow 0$ as $N \rightarrow \infty$ while the $MSME(\kappa) \neq 0$ unless $\kappa = 0$. That is, the optimal value of κ , $\kappa_{\text{opt}} = 0$ as $N \rightarrow \infty$. Since the results above are based on an $o(1/N)$ approximation it seems natural to consider κ_{opt} to $o(1/N)$, i.e., $\kappa_{\text{opt}} \propto 1/N$. It is found that

$$\kappa_{\text{opt}} = \frac{\sigma_\varepsilon^2 \text{tr} \mathbf{H}^{-1}}{N (\mathbf{w}^\circ)^\top \mathbf{H}^{-1} \mathbf{w}^\circ} + o(1/N). \quad (6.342)$$

Notice certain facts concerning this optimal value:

- It depends on the noise level, σ_ε . Clearly if no noise is present then $\kappa_{\text{opt}} = 0$ since the data in the training set perfectly describes the system as we note the model is complete. On the other hand, if noise is present⁷⁸ there will be a significant *WFP* which can be lowered by introducing a non-zero κ and thus introducing a *MSME*.
- To first order in $1/N$ κ_{opt} is independent of the fourth order statistic of the input vector signal, i.e., independent of \mathbf{K} .
- It is – loosely speaking – inversely proportional to the square of the optimal weight values. This is due to the fact that we regularize against the zero weight vector⁷⁹.

It is important to point out that κ_{opt} can not be calculated in practical applications since it depends on the optimal weights which are of course unknown; however even when $\kappa \neq \kappa_{\text{opt}}$ there may still be a reduction in the average generalization error. In fact, the average generalization error is reduced within the interval $\kappa \in]0; 2\kappa_{\text{opt}}]$ as demonstrated in App. G.

⁷⁸Notice that the noise variance enters the *WFP* only.

⁷⁹The regularizing term is zero when employing $\mathbf{w} = \mathbf{0}$.

6.10 Summary

The topic of this chapter was the construction of filter architecture synthesis algorithms. The goal of filter architecture synthesis is to select the filter architecture with maximum quality. As a quality measure the concept of generalization ability was introduced. In particular, we define the generalization error as the mean square error of an input-output sample which is independent of the training set consisting of N related input-output data. The generalization error depends on the system which is modeled and the training set through the weight estimate, $\hat{\mathbf{w}}$. In order to circumvent the dependence on the current training set we define the average generalization error which is the average of the generalization error over all possible training sets with size N .

Various properties of the average generalization error was discussed in terms of the model error decomposition. This leads, in particular, to a clarification of the fundamental limitations in searching for the optimal filter architecture.

The possibility of estimating the generalization error forms the basis of a simple architecture synthesis algorithm. The idea is to estimate the generalization error of various filters and then pick the one with minimal generalization error. A number of classical generalization error estimates was reviewed and a novel estimator, *GEN*, is presented. The *GEN*-estimator handles models which are:

- Nonlinear in the weights.
- Incomplete, i.e., models which do not perfectly model the system at hand.

A statistical framework for reducing the generalization error was presented. This framework lead to various algorithms for filter architecture synthesis which falls into two classes: Algorithms which gradually expand or prune the architecture. In addition, some of the synthesis algorithms suggested in the neural network literature were reviewed, interpreted, and extended.

Finally, we demonstrated that the use of regularization may reduce the generalization error. Consequently, regularization may be viewed as a tool for synthesizing a proper filter architecture.

CHAPTER 7

VALIDATION OF ARCHITECTURE SYNTHESIS METHODS

In this chapter the performance of various filter architecture synthesis methods is studied by numerical simulation. This is about:

- Validation and comparison of generalization error estimators, including:
 - The Generalization Error Estimator for Incomplete, Nonlinear Models (*GEN*-estimator) cf. Sec. 6.5.8.
 - The Cross-Validation estimator (*C*-estimator) cf. Sec. 6.5.3.
 - Leave-One-Out Cross-Validation estimator (*L*-estimator) cf. Sec. 6.5.4.
 - The Final Prediction Error estimator (*FPE*-estimator) cf. Sec. 6.5.6.
- Validation of statistically based algorithms for optimizing the filter architecture cf. Ch. 6.

7.1 Validation of the *GEN*-estimate

Recall that the ability to estimate the generalization error serves two purposes:

1. It provides the tool for evaluating the quality of an estimated model.
2. It enables the construction of algorithms for architecture synthesis (see Sec. 6.5.1).

An immediate validation strategy is obviously to compare the architectures which are synthesized by using the *GEN*-estimate relative to other techniques. However, it may be cumbersome to draw firm general conclusions from this comparison due to the following arguments:

- The resulting architecture will to some extent depend on the available training data. In consequence, one has to repeat the architecture synthesis several times and with different of training set sizes, N , thus resulting in some “mean” performance which depends on N . Furthermore, this task involves an enormous computational burden.

- In order to decide which of the architecture synthesis algorithms (ASA’s) one should prefer, a huge independent cross-validation set has to be available so that the “true” generalization error can be calculated¹ afterwards the ASA’s have been run.
- The result depends on the chosen algorithm for estimating the filter parameters (weights) since
 - The algorithm may converge to a local minimum.
 - The algorithm does not minimize the cost function perfectly. For instance, applying the SG-algorithm a perfect local minimization requires an infinite number of iterations and secondly, that the step-size is diminished according to a certain scheme as the number of iterations increase, cf. Sec. 5.4.
- Typically the ASA involves a number of steps where the architecture gradually changes. The path which is followed by a certain ASA through the space of architectures is highly dependent on the initial conditions and parameter settings. Consequently, numerous trials with the particular ASA is urgent in order to substantiate e.g., the mean performance of the ASA.
- The nature of the nonlinear system which is being modeled will influence on the results. One may imagine that ASA’s which turn out to be applicable on some examples will be overruled on other examples.

In the light of these arguments it is appraised that much effort is required in order to set up a secure validation procedure. This is therefore left for future research. Instead, we will employ a statistical approach aiming at eliminating some of the obstacles mentioned above. The approach consists in considering various generalization error estimates as statistical estimates of the *average generalization error*, $\Gamma = E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\}$, cf. sa:gamdef. The *GEN*-estimate is then compared with alternative generalization error estimates in order to elucidate how well the estimates predict the value of Γ .

7.1.1 Simulation Setup

The simulations performed use the conventional LS cost function solely. Simulations of the *GEN*-estimate applicable for the LS cost function with a simple regularizer (see further Sec. 6.5.8) is thus subject to future studies .

In the performed simulations the *GEN*-estimate is compared to the following estimates:

1. The Final Prediction Error estimate (*FPE*-estimate) which cf. Sec. 6.5.6, Th. 6.6 is given by:

$$FPE(\mathcal{T}) = \frac{N + m}{N - m} S_N(\hat{\mathbf{w}}), \quad N > m \quad (7.1)$$

where N is the size of the training set, $\hat{\mathbf{w}}$ is the estimated weights, $m = \dim(\hat{\mathbf{w}})$, and $S_N(\cdot)$ is the LS cost function based on N training samples. Recall that *FPE* is the estimator of Γ which appears by restricting the model to be complete².

Notice that *FPE* depends on the actual training set, \mathcal{T} , through the estimated weights, $\hat{\mathbf{w}}$.

¹Recall cf. Sec. 6.5.3 that the cross-validation estimate (excluding leave-one-out cross-validation) is an unbiased and consistent estimator of the generalization error in the limit of an infinite cross-validation set.

²Notice furthermore the assumptions given in Th. 6.6.

2. The cross-validation estimate (C -estimate) which cf. Sec. 6.5.3 is given by:

$$C_\nu(\mathcal{T}) = \frac{1}{N_c} \sum_{k=N-N_c+1}^N e^2(k; \widehat{\mathbf{w}}_{N-N_c}) \quad (7.2)$$

where $e(k; \widehat{\mathbf{w}}_{N-N_c})$ is the error obtained when using $\widehat{\mathbf{w}}_{N-N_c}$ estimated from $N - N_c$ samples. ν denotes the percentage of the total number of training samples used for estimating the weights, i.e.,

$$\nu = \frac{N - N_c}{N} \cdot 100\% \quad (7.3)$$

where N_c is the number of samples used to calculate the cross-validation estimate.

Note that the dependence on the actual training set is pointed out explicitly.

3. The leave-one-out cross-validation estimate (L -estimate) which cf. Sec. 6.5.4 yields:

$$L(\mathcal{T}) = \frac{1}{N} \sum_{j=1}^N e^2(j; \widehat{\mathbf{w}}_{(j)}) \quad (7.4)$$

where $\widehat{\mathbf{w}}_{(j)}$ is the weight estimate obtained by training on the training set: $\{\mathbf{x}(k); y(k)\}$, where

$$\begin{aligned} k &= 2, 3, \dots, N & , j &= 1 \\ k &= 1, 2, \dots, j-1, j+1, \dots, N & , j &\in [2; N-1] \\ k &= 1, 2, \dots, N-1 & , j &= N \end{aligned} \quad (7.5)$$

That is, for every j we train on $N - 1$ samples and calculate the squared error on the remaining single sample.

Aiming at an empirical comparison of the above generalization error estimates with the GEN -estimator we form Q independent training sets with sizes:

$$N = N_{\min}, N_{\min} + 1, \dots, N_{\max}. \quad (7.6)$$

The s 'th training set with size N , $\mathcal{T}_N^{(s)}$, is given by:

$$\mathcal{T}_N^{(s)} = \left\{ \mathbf{x}^{(s)}(k); y^{(s)}(k) \right\} \quad (7.7)$$

where $s \in [1; Q]$, $N \in [N_{\min}; N_{\max}]$, and $k \in [1; N]$. The weight estimate obtained by using the training set, $\mathcal{T}_N^{(s)}$, is denoted by $\widehat{\mathbf{w}}^{(s)}$.

In order to eliminate a source of error (the second item of the list mentioned on page 235) we assume that the model, the system, and the joint distribution of the input, \mathbf{x} , and the inherent error, ε , are selected so that it is possible to calculate the generalization error (cf. sa:gentrue):

$$G(\widehat{\mathbf{w}}) = E_{\mathbf{x}_t, \varepsilon_t} \left\{ e_t^2(\widehat{\mathbf{w}}) \right\} \quad (7.8)$$

where $[\mathbf{x}_t; \varepsilon_t]$ denotes an independent test sample. Note that $\widehat{\mathbf{w}}$ and the test sample are independent.

The average generalization error, Γ , is now estimated as:

$$\hat{\Gamma}_G = \langle G(\hat{\mathbf{w}}^{(s)}) \rangle = \frac{1}{Q} \sum_{s=1}^Q G(\hat{\mathbf{w}}^{(s)}) \quad (7.9)$$

where $\langle \cdot \rangle$ denotes the average w.r.t. the Q independent training sets, $\mathcal{T}_N^{(s)}$. If $G(\hat{\mathbf{w}}^{(s)})$ forms a mean-ergodic sequence w.r.t. to the Q realizations then [Papoulis 84a, Ch. 9-5] $\lim_{Q \rightarrow \infty} \hat{\Gamma}_G = \Gamma$ since $\Gamma = E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\}$. When N is small it is evident that the fluctuations in the weight estimates (for instance, as given by the variance of the weights) will be large. Consequently, the smaller N is, the larger Q is required in order to get a reliable estimate of Γ . It is worth noting that $\hat{\Gamma}_G$ is one of the best estimates of Γ one can hope for unless we deal with trivial models and systems³.

Let $\hat{\Gamma}(\mathcal{T}_N^{(s)})$ denote one of the estimators: *GEN*, *FPE*, C_ν , and L . The impending issues are:

- Clarification of the quality of the particular estimator, viz. defining suitable quantities which measure the “distance” between the estimator and the “true” value Γ . In a statistical perspective, this is about characterizing the distribution of the particular estimator.
- The comparison of the estimates, i.e., the comparison of the distributions.

It should be emphasized that no objective guidelines are available in order to settle these issues. In consequence, one has to argue in favor of a proper choice. In this work we consider three different quality measures:

- Normalized bias, *NB*.
- Mean squared error, *MSE*.
- Probability of proximity, Π .

7.1.1.1 Normalized Bias

The normalized bias of a particular estimator is defined by:

$$NB = \frac{\hat{\Gamma}_G - \langle \hat{\Gamma}(\mathcal{T}_N^{(s)}) \rangle}{\hat{\Gamma}_G} \quad (7.10)$$

where $\hat{\Gamma}_G$ is given by sv:gammahat. Sometimes the association with the estimator, $\hat{\Gamma}$, is denoted explicitly by: $NB(\hat{\Gamma})$. The normalized bias measures the average percentage distance of the particular estimator, $\hat{\Gamma}$, from the average generalization error, $\hat{\Gamma}_G$. It should be emphasized (cf. sv:gammahat) that $\hat{\Gamma}_G$ always is calculated due to the weights, $\hat{\mathbf{w}}^{(s)}$, which are estimated on all available training data, i.e., N – no matter how many data used for estimating, $\hat{\Gamma}$. For instance, when calculating the cross-validation and the leave-one-out cross-validation estimators the weights are estimated on fewer than N samples⁴. Consequently, this introduces a bias relative to the *GEN* and *FPE* estimators in which all N samples are used for estimating the weights. However, this bias seems – as regards

³For instance when the output, $y(k)$, is an uncorrelated stochastic signal with unknown mean value.

⁴The weights used to calculate the C -estimate are estimated on $N - N_c$ samples while $N - 1$ samples are used in the L -estimate.

the comparison – quite fair as the goal is to exploit the present N training optimally, i.e., simultaneously ensuring proper weight and the generalization error estimates.

In order to substantiate whether the GEN -estimator is closer to $\widehat{\Gamma}_G$ on the average than another estimator, say $\widehat{\Gamma}$, several statistical tests can be designed. The basic assumption is that it is possible to formulate the tests as Gaussian tests. Consider the scalar stochastic variable, Z , with mean value, μ_Z and variance σ_Z^2 . The hypotheses, \mathcal{H}_i , and alternatives, $\mathcal{H}_{a,i}$, $i = 1, 2, 3$, under consideration are:

$$\mathcal{H}_1 : \mu_z = 0 \quad \mathcal{H}_{a,1} : \mu_z \neq 0 , \quad (7.11)$$

$$\mathcal{H}_2 : \mu_z \leq 0 \quad \mathcal{H}_{a,2} : \mu_z > 0 , \quad (7.12)$$

$$\mathcal{H}_3 : \mu_z \geq 0 \quad \mathcal{H}_{a,3} : \mu_z < 0 . \quad (7.13)$$

Suppose that Q independent observations, Z_s , $s \in [1; Q]$ are provided then the hypotheses can be tested using the test statistic:

$$T = \frac{\sqrt{Q} \langle Z_s \rangle}{S} \quad (7.14)$$

where S is the empirical variance given by:

$$S = \frac{Q}{Q-1} \langle [Z_s - \langle Z_s \rangle]^2 \rangle = \frac{1}{Q-1} \sum_{s=1}^Q [Z_s - \langle Z_s \rangle]^2 . \quad (7.15)$$

The reject regions, \mathcal{C}_i , are given by (see e.g., [Anderson 84]):

$$\mathcal{C}_1 = \left\{ T < \tau_{\frac{\alpha}{2}}(Q-1) \vee T > \tau_{1-\frac{\alpha}{2}}(Q-1) \right\} \quad (7.16)$$

$$\mathcal{C}_2 = \{ T > \tau_{1-\alpha}(Q-1) \} \quad (7.17)$$

$$\mathcal{C}_3 = \{ T < \tau_{\alpha}(Q-1) \} \quad (7.18)$$

where α is the significance level⁵ and $\tau_{\alpha}(Q)$ is the α -fractile of the t-distribution with Q degrees of freedom. In principle it is assumed that Z is Gaussian distributed; however, due to the central limit theorem it is possible to relax this assumption provided that Q is sufficiently large [Anderson 84, Ch. 5.4]. In the limit $Q \rightarrow \infty$ the t-distribution converges to the Gaussian distribution in probability, i.e., the $\tau_{\alpha}(Q)$ fractiles are properly replaced by fractiles of the standard Gaussian distribution, \mathcal{N}_{α} . Let $t(Q)$ denote a t-distributed variable with Q degrees of freedom and $\mathcal{N}(0, 1)$ a standard Gaussian variable. The following equations then give the relation between the fractiles:

$$\alpha = \text{Prob} \{ t(Q) < \tau_{\alpha}(Q) \} \quad (7.19)$$

⇓

$$\alpha = \text{Prob} \left\{ \frac{t(Q)}{\sqrt{\frac{Q}{Q-2}}} < \frac{\tau_{\alpha}(Q)}{\sqrt{\frac{Q}{Q-2}}} \right\} \quad (7.20)$$

⁵That is, the probability of rejecting a true hypothesis.

↓ ($V\{t(Q)\} = Q/(Q - 2)$ and $Q \rightarrow \infty$)

$$\alpha = \text{Prob} \left\{ \mathcal{N}(0, 1) < \frac{\tau_\alpha(Q)}{\sqrt{\frac{Q}{Q-2}}} \right\}. \quad (7.21)$$

That is, since $\alpha = \text{Prob}\{\mathcal{N}(0, 1) < \mathcal{N}_\alpha\}$ then

$$\tau_\alpha(Q) = \mathcal{N}_\alpha \sqrt{\frac{Q}{Q-2}}, \quad Q \rightarrow \infty. \quad (7.22)$$

The result of testing the hypotheses listed above is summarized in Fig. 7.1. On the basis

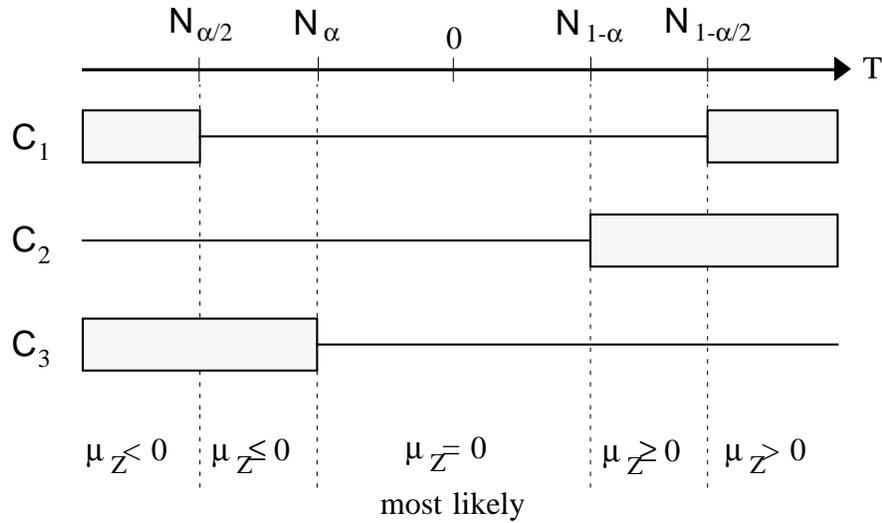


Figure 7.1: The result of testing the hypotheses in sv:h1-(7.13) depending on the magnitude of the test statistic T .

of the test of the Z variable it is possible to design tests which decide if GEN is better than another estimator, $\hat{\Gamma}$, on the average. Define the following stochastic variables:

$$Z_1 = \hat{\Gamma}_G - GEN, \quad (7.23)$$

$$Z_2 = \hat{\Gamma}_G - \hat{\Gamma}. \quad (7.24)$$

Now by using the tests listed above on each of the variables, Z_i , $i = 1, 2$, allow us to state a precedence of $\hat{\Gamma}_G$, $\langle GEN(\mathcal{T}_N^{(s)}) \rangle$, and $\langle \hat{\Gamma}(\mathcal{T}_N^{(s)}) \rangle$. Two fundamentally different cases occur:

1. Either both estimators, GEN and $\hat{\Gamma}$, are greater (lower) than $\hat{\Gamma}_G$.

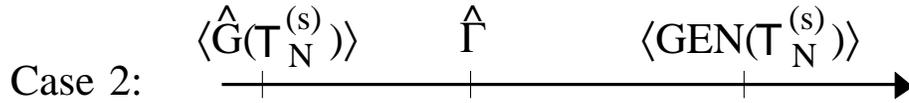


Figure 7.2: Two fundamentally different precedences of $\hat{\Gamma}_G$, $\langle GEN(\mathcal{T}_N^{(s)}) \rangle$, and $\langle \hat{\Gamma}(\mathcal{T}_N^{(s)}) \rangle$.

2. One is greater (lower) than $\hat{\Gamma}_G$ while the other is lower (greater).

This is shown in Fig. 7.2. If the tests show that case 1 occurs then an additional test on the mean value of the variable

$$Z_3 = GEN - \hat{\Gamma} \quad (7.25)$$

decides which of the estimators one should prefer. On the other hand, if case 2 occurs then it is necessary to test the mean value of the variable:

$$Z_4 = GEN + \hat{\Gamma} - 2\hat{\Gamma}_G. \quad (7.26)$$

The motivation for testing this variable is due to the inequalities:

$$GEN - \hat{\Gamma}_G < \hat{\Gamma}_G - \hat{\Gamma} \quad (7.27)$$

\Downarrow

$$GEN + \hat{\Gamma} - 2\hat{\Gamma}_G < 0. \quad (7.28)$$

Thus if $\mu_{Z_4} < 0$ then GEN is closer to $\hat{\Gamma}_G$ than $\hat{\Gamma}$ on the average⁶.

7.1.1.2 Mean Square Error

$$MSE = \left\langle \left[\hat{\Gamma}(\mathcal{T}_N^{(s)}) - \hat{\Gamma}_G \right]^2 \right\rangle. \quad (7.29)$$

⁶If the test of Z_i , $i = 1, 2$, resulted in that $E\{\hat{\Gamma}(\mathcal{T}_N^{(s)})\} > E\{GEN(\mathcal{T}_N^{(s)})\}$ then the interpretation of Z_4 should be negated.

The mean square error measures the average squared distance from $\widehat{\Gamma}_G$ ⁷. The *MSE* is in terms of bias and variance decomposed as:

$$MSE = \underbrace{\langle \widehat{\Gamma}^2(\mathcal{T}_N^{(s)}) \rangle - \langle \widehat{\Gamma}(\mathcal{T}_N^{(s)}) \rangle^2}_{\text{Variance}} + \underbrace{\left[\widehat{\Gamma}_G - \langle \widehat{\Gamma}(\mathcal{T}_N^{(s)}) \rangle \right]^2}_{\text{Squared Bias}}. \quad (7.30)$$

The *MSE* is thus a measure which combines first and second order moments of the estimator distribution. However, notice that indeed one has made a choice when forcing the variance and the squared bias to enter the *MSE with equal weight*. Of course one may argue in favor of other weightings. The statistical tests presented in the previous paragraph are in principle applicable when comparing the *MSE* of various estimators; however, Q is typically required to be extremely large in order to ensure any significance at all. This is due to the fact that it is hard to get a reliable estimate of the variance of the *MSE*'s. Consequently, such tests are omitted.

7.1.1.3 Probability of Proximity

Define the probability of proximity:

$$\begin{aligned} \Pi &= \text{Prob} \left\{ |GEN - \Gamma| < |\widehat{\Gamma} - \Gamma| \right\} \\ &\approx \left\langle \mu \left(\left| \widehat{\Gamma}(\mathcal{T}_N^{(s)}) - \widehat{\Gamma}_G \right| - \left| GEN(\mathcal{T}_N^{(s)}) - \widehat{\Gamma}_G \right| \right) \right\rangle \end{aligned} \quad (7.31)$$

where $\mu(\cdot)$ is the step function

$$\mu(n) = \begin{cases} 1 & , n \geq 0 \\ 0 & , n < 0 \end{cases} . \quad (7.32)$$

The probability of proximity measures the probability that the *GEN*-estimator is closer to $\widehat{\Gamma}_G$ than another estimator $\widehat{\Gamma}$ ⁸. This measure thus takes all higher order moments of the estimator distributions along with the dependence between the estimators into account. Note that $\Pi = 0.5$ corresponds to the case where *GEN* and $\widehat{\Gamma}$ are equally good in predicting Γ .

7.1.2 Simulated Systems

In the sections below three different simple systems and models are considered. The models under consideration are all *incomplete* since the novelty of the *GEN*-estimator is that it is designed to handle such models. Furthermore, recall that

1. Incomplete models are the common case since we rarely are provided with perfect knowledge of the system which has to be modeled.
2. The *GEN*-estimator coincides in most cases with the *FPE*-estimator when the model becomes complete.

⁷Sometimes the association with the estimator, $\widehat{\Gamma}$ is denoted explicitly by: $MSE(\widehat{\Gamma})$.

⁸Sometimes the association with the estimator, $\widehat{\Gamma}$ is denoted explicitly by: $\Pi(\widehat{\Gamma})$.

7.1.2.1 Volterra System

Consider the simple second order Volterra system:

$$y(k) = y^\circ(k) + \varepsilon(k) = [x(k), x^2(k)]\mathbf{w}^\circ + \varepsilon(k) \quad (7.33)$$

where

- $y^\circ(k)$ is the noiseless output the Volterra system.
- Two types of input signals, $x(k)$, are considered: White and colored Gaussian signals. In the white (i.i.d.) case $x(k) \in \mathcal{N}(0, 1)$. In the colored case,

$$x(k) = b(k) * \xi(k) = \sum_{n=0}^M b(n)\xi(k-n) \quad (7.34)$$

where $\xi(k) \in \mathcal{N}(0, 1)$ is an i.i.d. sequence and $b(k)$ is a M 'th order FIR-filter designed to implement a low-pass filter with normalized cutoff frequency $f_c = 0.01$. The design is performed with the MATLAB [MATLAB 94] routine "fir1" which uses a Hamming windowed ideal low-pass filter impulse response given by (see e.g., [Oppenheim & Schaffer 75, Ch. 7.4]):

$$b(k) = \begin{cases} \frac{\sin\left(2\pi\left(k - \left\lceil\frac{M}{2}\right\rceil\right)f_c\right)}{\pi\left(k - \left\lceil\frac{M}{2}\right\rceil\right)} \cdot \left(0.54 - 0.46\cos\left(\frac{2\pi k}{M}\right)\right) & , 0 \leq k \leq M \\ 0 & , \text{otherwise} \end{cases} \quad (7.35)$$

where $\lceil \cdot \rceil$ denotes rounding to the nearest integer towards infinity. In the present case we used $M = 15$. Note that M equals the dependence lag (defined in As. 6.5) since⁹

$$E\{x(k)x(k+\tau)\} = \sum_{n_1=0}^M \sum_{n_2=0}^M b(n_1)b(n_2)E\{\xi(k-n_1)\xi(k+\tau-n_2)\} = 0, \quad |\tau| > M. \quad (7.36)$$

- The (true) weights are given by: $\mathbf{w}^\circ = [1, 1]^\top$.
- $\varepsilon(k) \in \mathcal{N}(0, \sigma_\varepsilon^2)$ is an inherent i.i.d. noise sequence which is independent of $x(k)$. The variance, σ_ε^2 is determined so that the signal-to-noise ratio, $SNR = 5 \approx 7\text{dB}$, i.e.,

$$\sigma_\varepsilon^2 = \frac{E\{(y^\circ(k))^2\}}{SNR}. \quad (7.37)$$

According to sv:volsys and moment relations concerning Gaussian variables (see e.g., [Bendat & Piersol 86, Ch. 3.3.3] and App. B ih:moment) we get:

$$E\{(y^\circ(k))^2\} = (\mathbf{w}^\circ)^\top \mathbf{H} \mathbf{w}^\circ \quad (7.38)$$

⁹This is due to the fact that $\xi(k)$ is an i.i.d. sequence, i.e., only when

$$k - n_1 = k + \tau - n_2 \Leftrightarrow \tau = n_2 - n_1$$

the expectation in sv:corx takes non-zero values. Since $n_1, n_2 \in [0, M]$ this can only be accomplished for $-M \leq \tau \leq M$.

where \mathbf{H} is the expected Hessian

$$\mathbf{H} = \begin{bmatrix} E\{x^2(k)\} & E\{x^3(k)\} \\ E\{x^3(k)\} & E\{x^4(k)\} \end{bmatrix}. \quad (7.39)$$

Using moment relations concerning Gaussian variables (see e.g., [Bendat & Piersol 86, Ch. 3.3.3] and App. B, ih:moment):

$$\mathbf{H} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & 3\sigma_x^4 \end{bmatrix} \quad (7.40)$$

where $\sigma_x^2 = V\{x(k)\}$. In the white noise case $\sigma_x^2 = 1$, whereas in the colored noise case (cf. sv:corx and the fact that $E\{\xi(k)\} = 0$):

$$\sigma_x^2 = E\{x^2(k)\} = E\{\xi^2(k)\} \sum_{n=0}^M b^2(n). \quad (7.41)$$

The Volterra system in sv:volsys is modeled by the LL-model:

$$y(k) = w \cdot x(k) + e(k; w) \quad (7.42)$$

where w is a scalar weight and $e(k; w)$ is the error signal. The model is obviously incomplete as no $x^2(k)$ term enters this model. The modeling of the Volterra system is shown in Fig. 7.3. The weight, w , is estimated by minimizing the LS cost function, $S_N(w)$. According to Sec. 5.2 the estimate, $\hat{w}^{(s)}$ w.r.t. the training set, $\mathcal{T}_N^{(s)}$ (see sv:trainset) becomes:

$$\hat{w}^{(s)} = \left[\sum_{k=1}^N (x^{(s)}(k))^2 \right]^{-1} \cdot \sum_{k=1}^N x^{(s)}(k) y^{(s)}(k) \quad (7.43)$$

An important feature of the present system and model is the possibility of calculating the generalization error cf. sv:gentrue. Let $E\{\cdot\}$ denote the expectation w.r.t. an independent test sample $[x_t, \varepsilon_t]$. Using sv:volsys, (7.42) and the moment relations concerning Gaussian variables (see above) we get¹⁰:

$$\begin{aligned} G(\hat{w}) &= E\{e_t^2(\hat{w})\} \\ &= E\left\{ \left[w_1^\circ x_t + w_2^\circ x_t^2 + \varepsilon_t - \hat{w} x_t \right]^2 \right\} \\ &= (w_1^\circ - \hat{w})^2 E\{x_t^2\} + 3(w_2^\circ E\{x_t^2\})^2 + \sigma_\varepsilon^2 \end{aligned} \quad (7.44)$$

Since the model is an LL-model the *GEN*-estimators are given by Th. 6.3 and Th. 6.5:

$$GEN\left(\mathcal{T}_N^{(s)}\right) = S_N(\hat{w}^{(s)}) + \frac{2 \left[R(0) + 2 \sum_{\tau=1}^{\tau_{\max}} \frac{N-\tau}{N} R(\tau) \right]}{\sum_{k=1}^N (x^{(s)}(k))^2} \quad (7.45)$$

¹⁰The superscript $^{(s)}$ on the weight is omitted for simplicity.

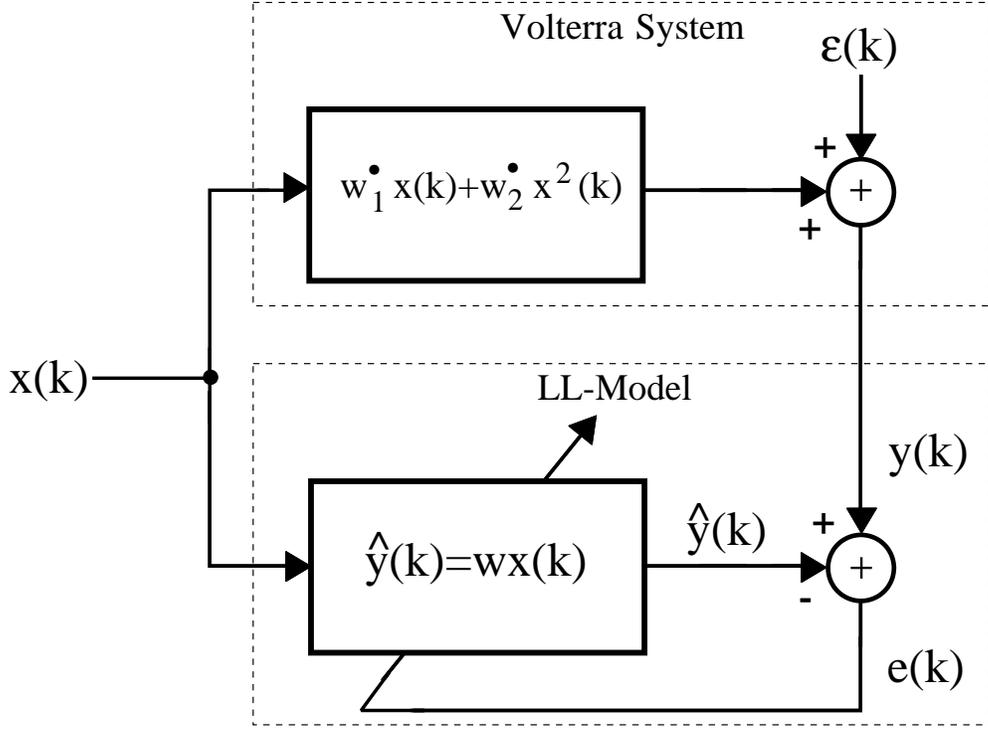


Figure 7.3: Incomplete modeling of a simple second order Volterra system.

where

$$R(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} x^{(s)}(k)x^{(s)}(k+\tau)e(k;\hat{w}^{(s)})e(k+\tau;\hat{w}^{(s)}). \quad (7.46)$$

In principle: $\tau_{\max} = M$. That is, dealing with white input $\tau_{\max} = 0$. However, dealing with colored input the equality can not be maintained when N is small compared to M as $R(\tau)$ becomes an unreliable estimator¹¹ of $\hat{\Phi}(\tau)$ (see gend:phihat). In general we therefore choose¹²:

$$\tau_{\max} = \min \left\{ M, \left\lfloor \frac{N}{c} \right\rfloor \right\} \quad (7.47)$$

where $\lfloor \cdot \rfloor$ denotes rounding to the nearest integer towards minus infinity and c is a suitable factor, typically $10 \leq c \leq 100$. The choice of c thus provides a trade off between; on the one hand demanding $R(\tau)$ to be reliable, and on the other hand forcing τ_{\max} to be as close as possible to M so that all essential terms are included.

7.1.2.2 Simple Neural Network

The concerned neural network system consists of a single nonlinear neuron with two inputs as follows:

$$y(k) = y^\circ(k) + \varepsilon(k) = h(\mathbf{z}^\top \mathbf{w}^\circ) + \varepsilon(k) \quad (7.48)$$

¹¹That is, high bias and variance cf. App. A and [Papoulis 84b, Sec. 11-2].

¹²It should be emphasized that other schemes for selecting τ_{\max} are definitely workable.

where

- $\mathbf{z}(k) = [z_1(k), z_2(k)]^\top$ is a 2-dimensional input vector signal. Let $\boldsymbol{\xi}(k) \in \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\boldsymbol{\xi})$ be a 2-dimensional i.i.d. Gaussian sequence where

$$\boldsymbol{\Sigma}_\boldsymbol{\xi} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}. \quad (7.49)$$

As previously, two types of input signals are considered. Either the input is white, i.e., $\mathbf{z}(k) = \boldsymbol{\xi}(k)$, or colored

$$z_j(k) = \xi_j(k) * b(k), \quad j = 1, 2 \quad (7.50)$$

where $b(k)$ is the filter given by sv:bk. In the colored case the covariance matrix of $\mathbf{z}(k)$ becomes¹³:

$$\boldsymbol{\Sigma}_\mathbf{z} = \boldsymbol{\Sigma}_\boldsymbol{\xi} \cdot \sum_{n=0}^M b^2(n). \quad (7.51)$$

- The (true) weights¹⁴ are in the white input case given by $\mathbf{w}^\circ = [0.8, 0.8]^\top$, and in the colored case by $\mathbf{w}^\circ = [3, 3]^\top$.
- The activation function, $h(u)$ takes the form:

$$h(u) = \exp\left(-\frac{(u - \nu)^2}{\eta^2}\right) - \exp\left(-\frac{(u + \nu)^2}{\eta^2}\right) \quad (7.52)$$

where ν is a position parameter and η is a scale parameter. The reason for employing this activation function is that it allows for calculation of the generalization error, as we shall see below. $\nu = 2$ and $\eta = 1$ were employed which result in the shape depicted in Fig. 7.4.

- The inherent noise is an i.i.d. Gaussian sequence, $\varepsilon(k) \in \mathcal{N}(0, \sigma_\varepsilon^2)$, independent of $\mathbf{z}(k)$, and with variance

$$\sigma_\varepsilon^2 = \frac{E\{(y^\circ(k))^2\}}{SNR} \quad (7.53)$$

where $SNR = 10 = 10\text{dB}$. The evaluation of $E\{(y^\circ(k))^2\}$ is treated in App. D and corresponds to the term denoted G_1 .

The employed NN-model of the neural network system in sv:neusys is parameterized by a single weight, w , and obeys:

$$y(k) = h(w \cdot z_1(k)) + e(k; w). \quad (7.54)$$

The model is incomplete since $z_2(k)$ does not enter the model. The modeling is shown in Fig. 7.5. For each training set, $\mathcal{T}_N^{(s)}$ the weight, w , is estimated based on a LS cost function, $S_N(w)$, using an off-line Modified Gauss-Newton algorithm (MGN) cf. Sec. 5.5. Note that an infinite number of iterations in principle is necessary in order to ensure that

¹³To see this one may use sv:corx. For a further reference see App. B p. 363.

¹⁴The true weight values are chosen so that the system essentially becomes nonlinear. Note that if $u(k) = \mathbf{z}^\top \mathbf{w}^\circ$ is too close to zero then the system is quasi-linear as $h(u) \approx cu$ where c is a constant.

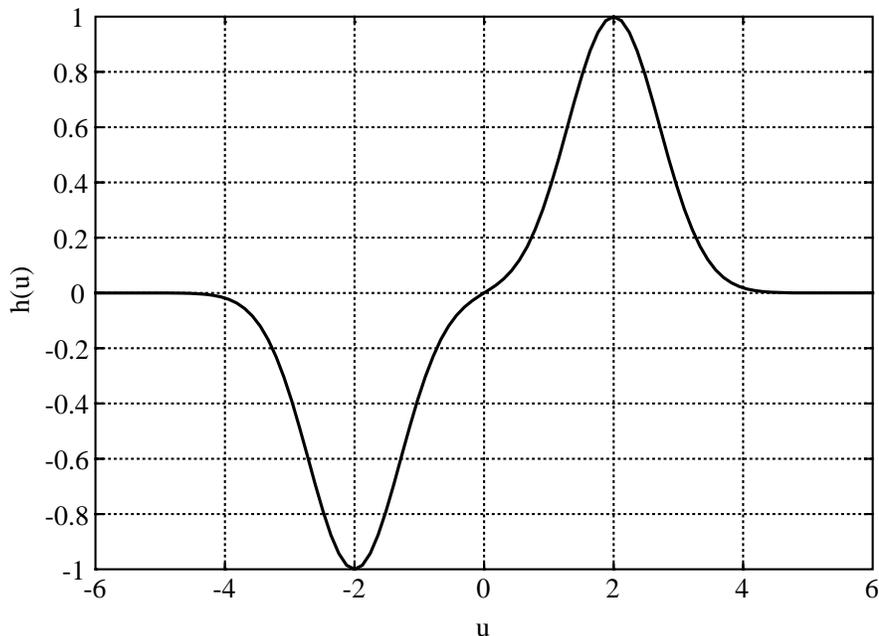


Figure 7.4: The activation function $h(u)$ when $\nu = 2$ and $\eta = 1$.

\hat{w} really minimizes the LS cost function and thereby eliminates the effect of the algorithm (see Ch. 6 for a more thorough discussion). Let $w_{(i)}$ denote the the weight estimate at iteration i . The MGN-algorithm was stopped when the stopping criterion

$$\frac{S_N(w_{(i)}) - S_N(w_{(i-1)})}{S_N(w_{(i-1)})} < 10^{-12}$$

was met or if the number of iterations exceeded 100. These settings are supposed to eliminate the algorithm effect appropriately.

Due to the special shape of the activation function, the simplicity of the neural network, and the fact that the input is Gaussian distributed it is possible to calculate the generalization error¹⁵

$$G(\hat{w}) = E \left\{ e^2(w) \right\} = \sigma_\varepsilon^2 + E \left\{ [h(w_1^\circ z_1 + w_2^\circ z_2) - h(\hat{w} z_1)]^2 \right\}. \quad (7.55)$$

The details are given in App. D.

Recall from Ch. 6 that the *GEN*-estimate measures the effect of having a finite training set, i.e., the average increase in the expected squared error due to fluctuations in the estimates, $\hat{w}^{(s)}$, around the optimal weight, w^* , (which minimizes the generalization error). In general w^* may not be unique; however this does not affect the usefulness of the *GEN*-estimate, since the averaging implicitly is done w.r.t. estimates, $\hat{w}^{(s)}$, which are close to a particular optimal weight, w^* . When simulating this averaging explicitly by generating a

¹⁵The expectation is w.r.t. to the joint distribution of \mathbf{z} and ε . Note that usual subindex t , denoting a test sample, is omitted.

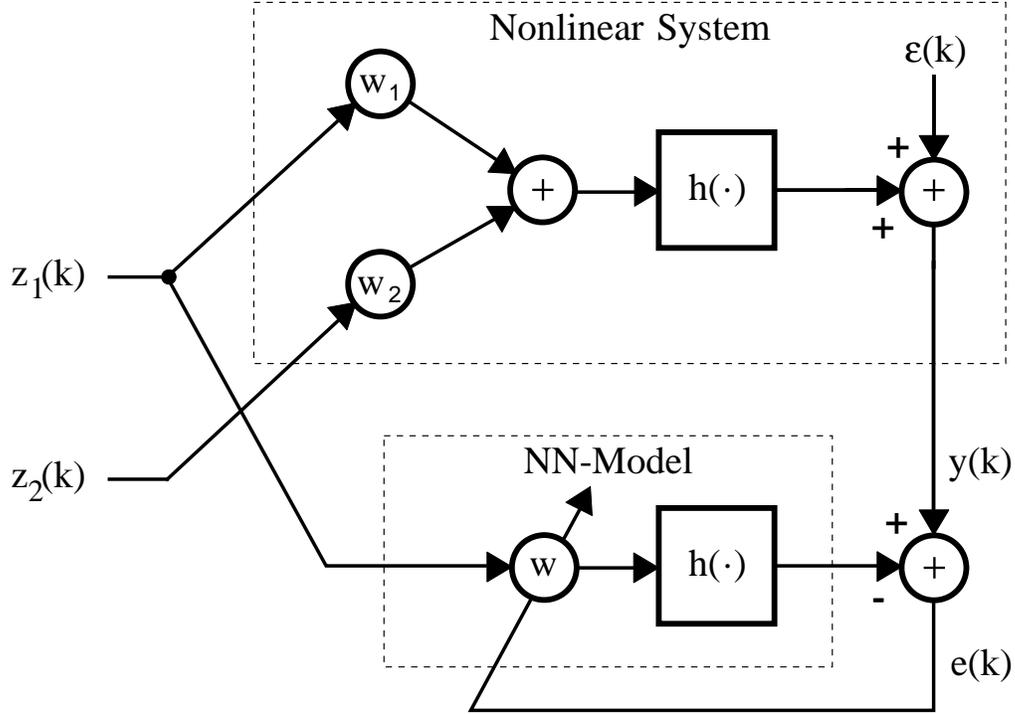


Figure 7.5: Incomplete modeling of a simple neural network consisting of a single nonlinear neuron with two inputs.

large number of training sets, one should carefully attend to that all estimates indeed are perturbations of *the same optimum*, w^* . In the present case this problem never occurs since the optimal weight is unique. This is experienced by inspecting $G(w)$ depicted in Fig. 7.6.

As the model in `sv:neumod` is an NN-model the *GEN*-estimators are calculated due to Th. 6.2 and Th. 6.4. We get:

$$GEN\left(\mathcal{T}_N^{(s)}\right) = S_N(\hat{w}^{(s)}) + \frac{2}{N} \frac{R(0) + 2 \sum_{\tau=1}^{\tau_{\max}} \frac{N-\tau}{N} R(\tau)}{H_N(\hat{w}^{(s)})} \quad (7.56)$$

where the involved quantities are given by:

$$R(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \psi(k; \hat{w}^{(s)}) \psi(k+\tau; \hat{w}^{(s)}) e(k; \hat{w}^{(s)}) e(k+\tau; \hat{w}^{(s)}) \quad (7.57)$$

$$\psi(k; \hat{w}^{(s)}) = h'(\hat{w}^{(s)} \cdot z_1^{(s)}(k)) z_1^{(s)}(k) \quad (7.58)$$

$$h'(u) = -2 \frac{x-\nu}{\eta^2} \exp\left(-\frac{(u-\nu)^2}{\eta^2}\right) + 2 \frac{x+\nu}{\eta^2} \exp\left(-\frac{(u+\nu)^2}{\eta^2}\right), \quad (7.59)$$

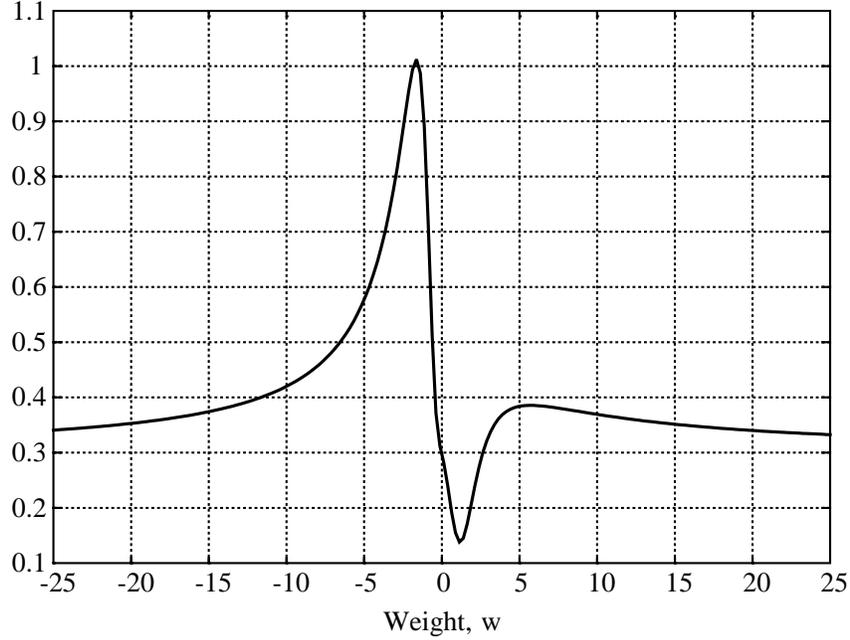


Figure 7.6: Expected cost function, $G(w)$ (cf. sv:gtruenau), when modeling the neural system in sv:neusys by the model in sv:neumod. Obviously, only one (global) minimum is present.

$$H_N(\hat{w}^{(s)}) = \frac{1}{N} \sum_{k=1}^N \psi^2(k; \hat{w}^{(s)}) - h''(u) \left(z_1^{(s)}(k) \right)^2 e(k; \hat{w}^{(s)}), \quad (7.60)$$

$$h''(u) = -2h(u) + 4 \frac{(x - \nu)^2}{\eta^4} \exp\left(-\frac{(u - \nu)^2}{\eta^2}\right) - 4 \frac{(x + \nu)^2}{\eta^4} \exp\left(-\frac{(u + \nu)^2}{\eta^2}\right). \quad (7.61)$$

The value of τ_{\max} is discussed in the preceding paragraph (see sv:taumax).

7.1.2.3 Polynomial System

In order to substantiate the quality of the *GEN*-estimate within a multi-parameter model we consider a 4'th ($l = 4$) order the Chebyshev system (see Sec. 3.2.1.3) given by the canonical representation:

$$\begin{aligned} y(k) &= y^\circ(k) + \varepsilon(k) \\ &= g_n(\mathbf{z}(k)) + \varepsilon(k) \\ &= \left(\mathbf{w}_q^\circ\right)^\top \mathbf{v}_q(k) + \varepsilon(k) \end{aligned} \quad (7.62)$$

where

- $\mathbf{z}(k)$ is a two-dimensional ($p = 2$) input vector signal. Let $\boldsymbol{\xi} \in \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\xi}})$ be an i.i.d. 2-dimensional Gaussian sequence with

$$\boldsymbol{\Sigma}_{\boldsymbol{\xi}} = \begin{bmatrix} 0.2 & 0.04 \\ 0.04 & 0.2 \end{bmatrix}. \quad (7.63)$$

As before we consider two types of input: The white input case corresponding to $\mathbf{z} = \boldsymbol{\xi}$ and the colored case:

$$z_j(k) = \xi_j(k) * \tilde{b}(k), \quad j = 1, 2 \quad (7.64)$$

where

$$\tilde{b}(k) = \frac{b(k)}{\sqrt{\sum_{k=0}^M b^2(k)}}, \quad (7.65)$$

and $b(k)$ is the filter given by sv:bk with $M = 10$ and $f_c = 0.01$.

- \mathbf{w}_q° is the true weight vector with dimension $q = C_{l+p,l} = C_{6,4} = 15$ which is given by:

$$\mathbf{w}_q^\circ = [1, 1, 3, -0.5, 0.5, 1.5, -1, 0.5, 0.2, -0.7, 0.2, 0.2, -0.5, 0.1, 0.1]^\top. \quad (7.66)$$

The subindex q indicates that the dimension of the weight vector is equal to q .

- $\mathbf{v}_q(k)$ contains the q polynomial terms – in this case Chebyshev polynomials (time index k omitted):

$$\begin{aligned} \mathbf{v}_q &= [1, T_1(z_1), T_2(z_1), \dots, T_l(z_1), \\ &T_1(z_2), T_1(z_2)T_1(z_1), \dots, T_1(z_2)T_{l-1}(z_1), \\ &T_1(z_3), T_1(z_3)T_1(z_1), \dots, T_1(z_3)T_{l-1}(z_1), \\ &\vdots \\ &T_{l-1}(z_{p-1})T_1(z_p), \\ &T_l(z_p)]^\top \end{aligned} \quad (7.67)$$

where $T_r(\cdot)$ is the r 'th order Chebyshev polynomial.

- The inherent noise, $\varepsilon(k) \in \mathcal{N}(0, \sigma_\varepsilon^2)$, is an i.i.d. Gaussian sequence, independent of $\mathbf{z}(k)$, and with

$$\sigma_\varepsilon^2 = \frac{E \left\{ (y^\circ(k))^2 \right\}}{SNR} \quad (7.68)$$

where $SNR = 20 \approx 13\text{dB}$. $E \left\{ (y^\circ(k))^2 \right\}$ is, cf. sv:chesys, evaluated as

$$\begin{aligned} E \left\{ (y^\circ(k))^2 \right\} &= E \left\{ \left(\mathbf{w}_q^\circ \right)^\top \mathbf{v}_q(k) \mathbf{v}_q^\top(k) \mathbf{w}_q^\circ \right\} \\ &= \left(\mathbf{w}_q^\circ \right)^\top \mathbf{H} \mathbf{w}_q^\circ \end{aligned} \quad (7.69)$$

where the expectation is w.r.t. \mathbf{z} and $\mathbf{H} = E \left\{ \mathbf{v}_q(k) \mathbf{v}_q^\top(k) \right\}$ is the expected Hessian. The evaluation of \mathbf{H} is treated in App. E.

The LN-model of sv:chesys is given by:

$$\begin{aligned} y(k) &= f_n(\mathbf{z}(k); \mathbf{w}_m) + e(k; \mathbf{w}_m) \\ &= \mathbf{w}_m^\top \mathbf{v}_m(k) + e(k; \mathbf{w}_m) \end{aligned} \quad (7.70)$$

where \mathbf{w}_m is the weight vector with dimension $m = 6$. $\mathbf{v}_m(k)$ is an restriction of $\mathbf{v}_q(k)$ corresponding to that some (i.e., $q - m = 9$) of the polynomial terms are excluded. Consequently, the model becomes incomplete. The restriction is formally expressed by:

$$\mathbf{v}_m(k) = \mathbf{\Xi} \mathbf{v}_q(k) \quad (7.71)$$

where $\mathbf{\Xi}$ is an $m \times q$ *restriction matrix* with only one non-zero element in each row corresponding to that one of the components in \mathbf{v}_m is paired with one of the components in \mathbf{v}_q .

In the present case the restricted vector $\mathbf{v}_m(k)$ yields (time index k omitted):

$$\mathbf{v}_m = [1, T_1(z_1), T_2(z_1), T_1(z_2), T_1(z_1)T_1(z_2), T_2(z_2)]^\top. \quad (7.72)$$

That is, all 3'rd and 4'th order terms in the $\mathbf{v}_q(k)$ vector are removed. Consequently, the model becomes a 2'nd order Chebyshev polynomial model. The modeling is shown in Fig. 7.7.

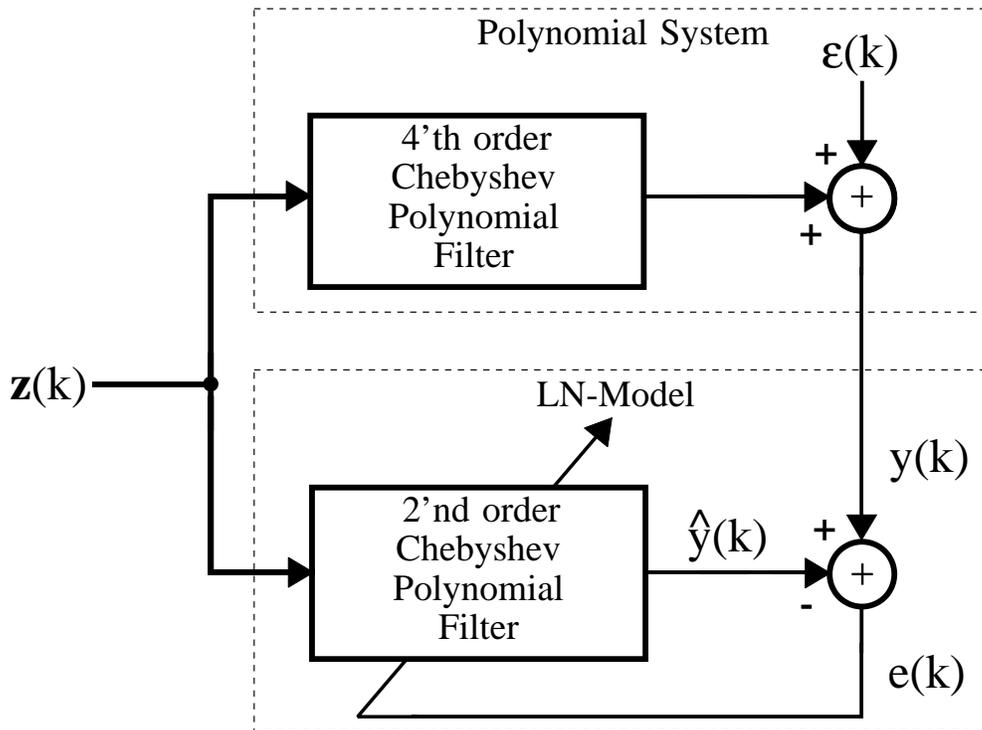


Figure 7.7: Incomplete modeling of a 4'th order Chebyshev polynomial system by a 2'nd order Chebyshev polynomial filter.

The weight vector is estimated according to the LS cost function. Cf. Sec. 5.2 the estimate based on the training set, $\mathcal{T}_N^{(s)}$ becomes:

$$\hat{\mathbf{w}}_m^{(s)} = \mathbf{H}_N^{-1} \cdot \frac{1}{N} \sum_{k=1}^N y^{(s)}(k) \mathbf{v}_m^{(s)}(k) \quad (7.73)$$

where $y^{(s)}(k)$, $\mathbf{v}_m^{(s)}(k)$ are the output and the \mathbf{v} -vector of the s 'th training set, respectively. Furthermore, the Hessian, \mathbf{H}_N , is given by

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{v}_m^{(s)}(k) \left(\mathbf{v}_m^{(s)}(k) \right)^\top. \quad (7.74)$$

It should be noted that the inverse Hessian, \mathbf{H}_N^{-1} in general is evaluated as the *Moore-Penrose pseudo inverse* (e.g., [Seber & Wild 89, App. A]) in order to handle numerical problems when \mathbf{H}_N is ill-conditioned which may be the case when N is small. Consider a single value decomposition (eigenvalue decomposition) of the Hessian:

$$\mathbf{H}_N = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^\top \quad (7.75)$$

where $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m]$ is the matrix of normalized eigenvectors, \mathbf{q}_i , $i = 1, 2, \dots, m$ and $\mathbf{\Lambda} = \text{diag}\{[\lambda_1, \lambda_1, \dots, \lambda_m]\}$ is the eigenvalue matrix where λ_i denotes the i 'th eigenvalue. Suppose that the eigenvalues are sorted so that

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m \geq 0, \quad (7.76)$$

and assume that $\lambda_{m_{\text{eff}}+1} = \lambda_{m_{\text{eff}}+2} = \dots = 0$ ¹⁶ then the Moore-Penrose pseudo inverse, \mathbf{H}_N^+ , is given by:

$$\mathbf{H}_N^+ = \mathbf{Q} \cdot \begin{bmatrix} \lambda_1^{-1} & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \lambda_2^{-1} & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_{m_{\text{eff}}}^{-1} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \cdot \mathbf{Q}^\top. \quad (7.77)$$

Compared to the ordinary inverse, $\mathbf{H}_N^{-1} = \mathbf{Q} \mathbf{\Lambda}^{-1} \mathbf{Q}^\top$, the pseudo inverse appears by replacing the last $m - m_{\text{eff}}$ reciprocal eigenvalues in $\mathbf{\Lambda}^{-1}$ by zeros.

In order to calculate the generalization error at the estimated weights, $G(\hat{\mathbf{w}}_m)$, we define $\hat{\mathbf{w}}_q$ so that $\hat{\mathbf{w}}_m = \mathbf{\Xi} \hat{\mathbf{w}}_q$. Secondly, $\hat{w}_{q,i} = 0$, $i \in \{s \in [1; q] \mid \Xi_{r,s} \neq 1, \forall j \in [1; m]\}$ where $\Xi_{r,s}$ is the (r, s) 'th entry in matrix $\mathbf{\Xi}$. That is, the weights corresponding to terms which are present in the system but not in the model are set equal to zero. In consequence the model is rewritten as:

$$\begin{aligned} y(k) &= \hat{\mathbf{w}}_m^\top \mathbf{v}_m(k) + e(k; \hat{\mathbf{w}}_m) \\ &\triangleq \hat{\mathbf{w}}_q^\top \mathbf{v}_q(k) + e(k; \hat{\mathbf{w}}_q). \end{aligned} \quad (7.78)$$

¹⁶In practice, this means zero w.r.t. some numerical precision. Using double precision the precision is around $\epsilon = 10^{-15}$. Numerical error analysis shows (see e.g., [Madsen & Nielsen 72], [Madsen & Nielsen 75]) that an eigenvalue $\lambda_i \leq \epsilon \sqrt{m} \cdot \|\mathbf{H}_N\|_2$ is indistinguishable from a zero eigenvalue. $\|\cdot\|_2$ is the matrix 2-norm equal to the largest eigenvalue.

Now by substituting sv:chesys into sv:chemod¹⁷:

$$\begin{aligned}
G(\hat{\mathbf{w}}_q) &= E \left\{ \left[y - \hat{\mathbf{w}}_q^\top \mathbf{v}_q \right]^2 \right\} \\
&= E \left\{ \left[\varepsilon + (\mathbf{w}_q^\circ - \hat{\mathbf{w}}_q)^\top \mathbf{v}_q \right]^2 \right\} \\
&= \sigma_\varepsilon^2 + (\mathbf{w}_q^\circ - \hat{\mathbf{w}}_q)^\top \mathbf{H} (\mathbf{w}_q^\circ - \hat{\mathbf{w}}_q)
\end{aligned} \tag{7.79}$$

where $\mathbf{H} = E\{\mathbf{v}_q \mathbf{v}_q^\top\}$. Recall that the calculation of \mathbf{H} is treated in App. E.

Since the model is an LN-model the *GEN*-estimates are given by Th. 6.3 and Th. 6.5, i.e.,

$$GEN \left(\mathcal{T}_N^{(s)} \right) = S_N \left(\hat{\mathbf{w}}_m^{(s)} \right) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^{\tau_{\max}} \frac{N-\tau}{N} \left(\mathbf{R}(\tau) + \mathbf{R}^\top(\tau) \right) \right) \mathbf{H}_N^+ \right] \tag{7.80}$$

where

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \mathbf{v}_m(k) e(k; \hat{\mathbf{w}}_m^{(s)}) \mathbf{v}_m^\top(k+\tau) e(k+\tau; \hat{\mathbf{w}}_m^{(s)}) \tag{7.81}$$

The employed value of τ_{\max} is given by sv:taumax.

7.1.3 Simulation Results

Using the different systems and models mentioned above, this subsection presents numerical comparisons of *GEN* and various generalization error estimators.

7.1.3.1 Volterra System

Comparison of *GEN* and *FPE* The number of independent training sets were chosen as:

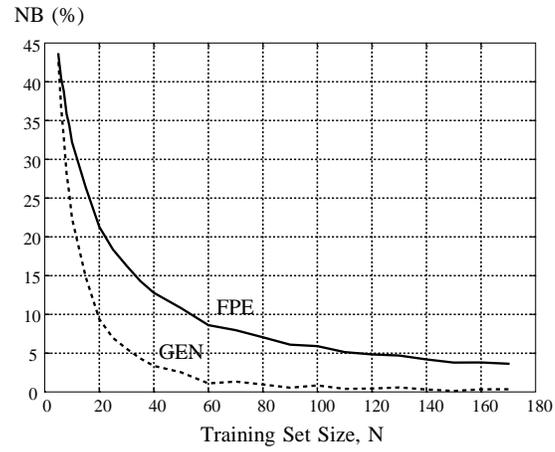
$$Q = \begin{cases} 30000 & 5 \leq N \leq 9 \\ 20000 & 10 \leq N \leq 170 \end{cases} \tag{7.82}$$

and we used $c = 20$ according to sv:taumax.

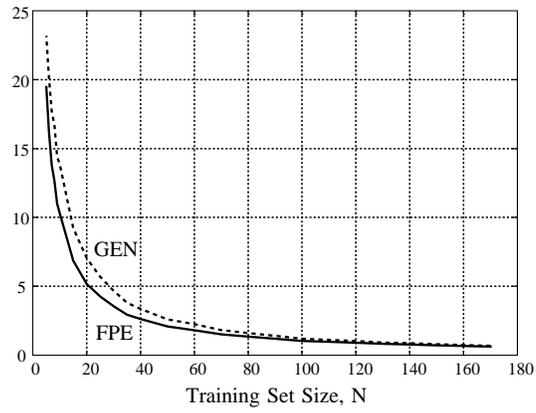
White Input Signal In Fig. 7.8 the result of comparing *GEN* and *FPE* when applying a white input signal is shown. On a $\alpha = 0.5\%$ significance level the tests described on pp. 239–241 result in that $|NB(GEN)| < |NB(FPE)|$ for all $5 \leq N \leq 170$. Particularly, in the range $20 \leq N \leq 120$ $|NB(GEN)|$ is less than the half of $|NB(FPE)|$. The mean squared error $MSE(GEN)$ is slightly higher than $MSE(FPE)$ – especially as $N < 80$. One could then argue that nothing speaks in favor of using the *GEN*-estimator since what is gained in lower bias is wasted in increased variance. However, recall that *MSE* is merely one particular measure which equally balances the first and second order moments (i.e., bias and variance) of the estimator distribution. Inspecting the probability of proximity, Π , it is seen that $\Pi \approx 0.6$ as $N > 15$. That is, with probability 0.6 *GEN* will be closer to Γ than *FPE* – even though *GEN* has a slightly higher *MSE*. Moreover, notice that Π peaks at $N = 20$ and then declines towards the asymptotic value 0.5¹⁸. In summary,

¹⁷The expectation below is w.r.t. to a test sample $[z_t, \varepsilon_t]$ independent of the training set.

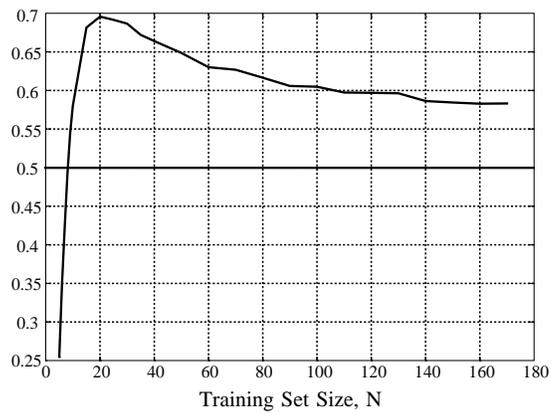
¹⁸This is due to the fact that both *GEN* and *FPE* are consistent estimators in the limit $N \rightarrow \infty$.



(a)



(b)



(c)

Figure 7.8: Comparison of *GEN* and *FPE* within the Volterra system Sec. 7.1.2.1 when applying a white input signal.

these arguments lead to the statement that one should prefer the *GEN*-estimator when $N > 15$ to the *FPE*-estimator at the expense of an increased computational burden cf. the discussion in Ch. 6.

Colored Input Signal Fig. 7.9 shows the comparison of *GEN* and *FPE* when applying a colored input signal. When $N > 15$ $|NB(GEN)| < |NB(FPE)|$ and vice versa when $N \leq 15$ on a $\alpha = 0.5\%$ significance level. When $N > 100$ the $|NB(GEN)|$ is less than or equal to one half of the $|NB(FPE)|$. However, the overall normalized bias performance of the *GEN*-estimator compared to the *FPE*-estimator is lower in this case than when applying a white input signal. The main reasons for this phenomenon are¹⁹:

- Employing a factor $c = 20$ in `sv:taumax` provides a tradeoff between reliable estimates of the correlation quantity, $R(\tau)$ and the the wish $\tau_{\max} = M$ – where $M = 15$ in the present case. Even when $N = 170$ only around half of the terms, $R(\tau)$, are included in the *GEN*-estimate since $\tau_{\max} = \min\{14, \lfloor 170/20 \rfloor\} = 8$. However, trials using a smaller c did not improve the performance in this case²⁰.
- Another reason stems from the approximation of the inverse Hessian matrix²¹, \mathbf{H}_N^{-1} . The approximation is definitely worse in the colored input case relative to dealing with a white input signal. It was shown in Th. B.4 that the error done when approximating the expectation of the inverse Hessian is $o((2M + 1)/N)$. Hence one expects that N has to be much larger in the colored case ($M > 0$) in order to obtain the same degree of accuracy. For numerical experiments on this topic the reader is referred to Ex. B.3.

Like the white input case the *MSE* of *GEN* is slightly higher than that of *FPE*. Never the less $\Pi > 0.5$ as $N \geq 25$, and furthermore $\Pi \approx 0.7$ in the interval $60 \leq N \leq 170$ indicating that *GEN* is closer to Γ than *FPE*. Hence, one should prefer *GEN* when $N \geq 25$.

Comparison of *GEN* and Cross-validation In the comparison with the cross-validation estimates the number of independent training sets was:

$$Q = 40000, \quad 6 \leq N \leq 100, \quad (7.83)$$

and $c = 10$ in `sv:taumax`.

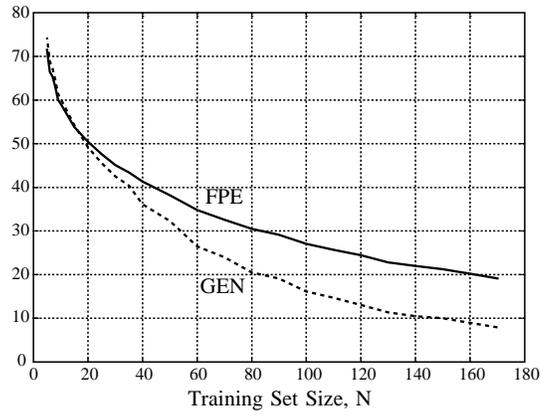
White Input Signal Fig. 7.10, Fig. 7.11 show the comparisons of *GEN* with C_{50} and C_{90} , respectively, when applying a white input signal. On a $\alpha = 20\%$ significance level²² the tests pp. 239–241 result in:

¹⁹Recall that the second order Taylor series expansion of the cost function is exact since we deal with an LL-model. Hence, no source of error is introduced hereof.

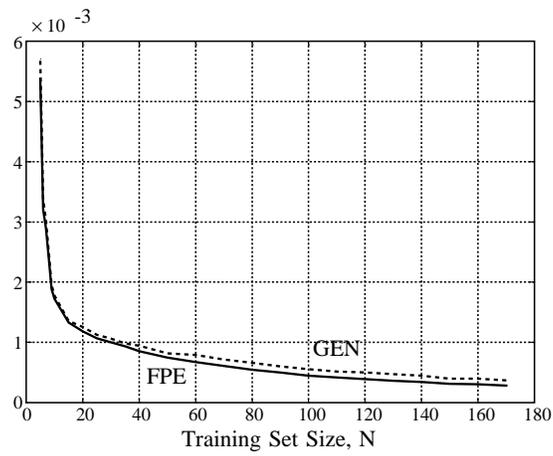
²⁰Notice that other schemes for selecting τ_{\max} possibly result in a slightly different performance. However, in practice it is of course not possible to optimize the strategy for selecting τ_{\max} as we do not know the true generalization error.

²¹This matrix reduces to a scalar in the present case.

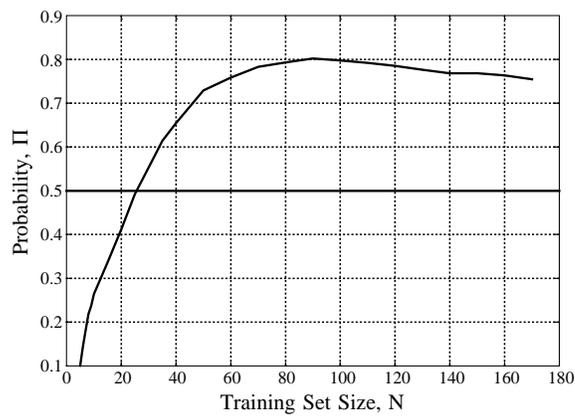
²²Due to the fact that cross-validation estimates are afflicted with very large variance it was not possible to obtain any significant statistical decisions on smaller significance levels.



(a)

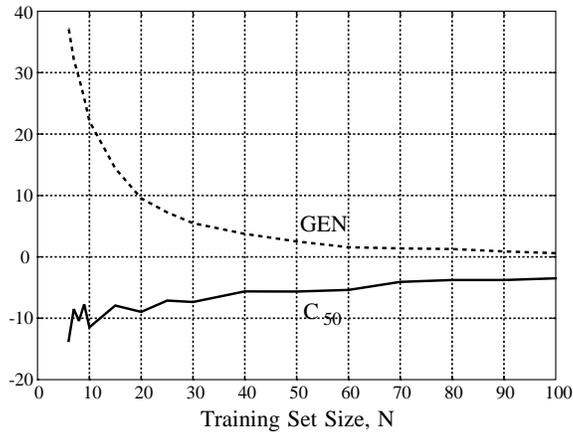


(b)

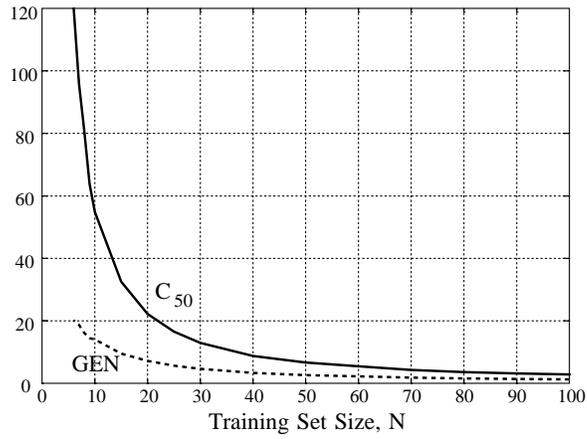


(c)

Figure 7.9: Comparison of *GEN* and *FPE* within the Volterra system Sec. 7.1.2.1 when applying a colored input signal.



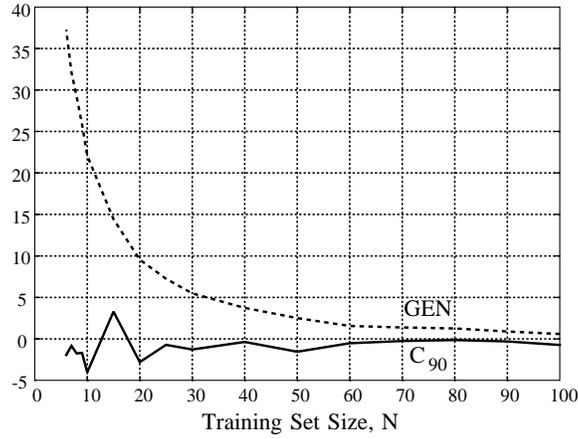
(a)



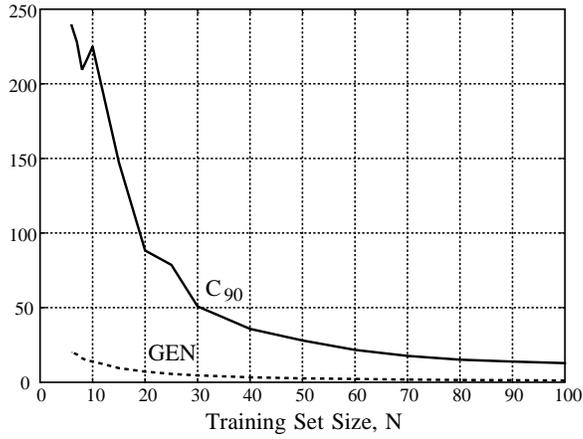
(b)

Figure 7.10: Comparison of GEN and C_{50} within the Volterra system Sec. 7.1.2.1 when applying a white input signal.

Relation	Training Set Size
$ NB(GEN) > NB(C_{50}) $	$6 \leq N \leq 9$
$ NB(GEN) \approx NB(C_{50}) $	$10 \leq N \leq 40$
$ NB(GEN) < NB(C_{50}) $	$41 \leq N \leq 100$
$ NB(GEN) > NB(C_{90}) $	$6 \leq N \leq 9$
$ NB(GEN) \approx NB(C_{50}) $	$10 \leq N \leq 100$



(a)



(b)

Figure 7.11: Comparison of GEN and C_{90} within the Volterra system Sec. 7.1.2.1 when applying a white input signal.

When $N > 40$ the $|NB(GEN)|$ is smaller than $|NB(C_{50})|$, and in addition, we notice that $|NB(GEN)|$ constitutes approximately 1/4 of $|NB(C_{50})|$ when $N \geq 60$. On the other hand, the normalized biases concerning GEN and C_{90} are indistinguishable when $N \geq 10$.

Concerning the mean squared errors it is seen that both $MSE(C_{50})$ and $MSE(C_{90})$ are huge compared to $MSE(GEN)$ for all $6 \leq N \leq 100$. In fact,

$$2 \lesssim \frac{MSE(C_{50})}{MSE(GEN)} \lesssim 6, \quad (7.84)$$

$$7 \lesssim \frac{MSE(C_{90})}{MSE(GEN)} \lesssim 12. \quad (7.85)$$

Finally, considering the probability of proximity depicted in Fig. 7.12 we notice that $\Pi > 0.5$ in for all $6 \leq N \leq 100$, particularly w.r.t. C_{90} . In conclusion, we may prefer

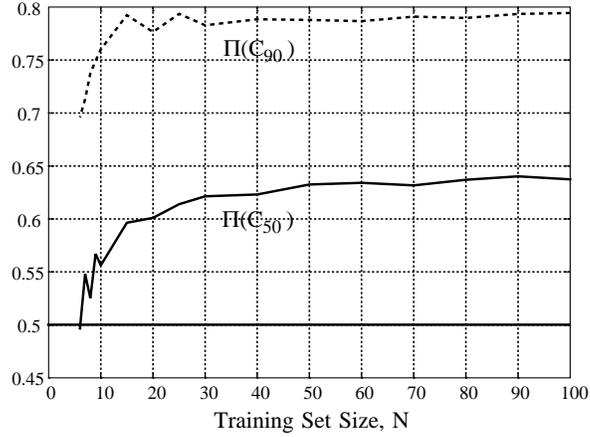


Figure 7.12: Probability of proximity concerning the C_{50} and C_{90} estimators within the Volterra system Sec. 7.1.2.1 when applying a white input signal.

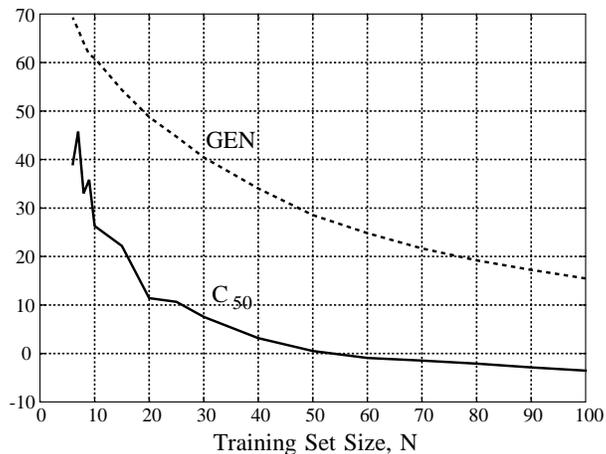
the GEN -estimator to the C_{50} -estimator as $N \geq 50$ since the $|NB|$ is lower, the MSE is substantially lower, and $\Pi \approx 0.63$. Even when $N < 50$ still the GEN -estimator is preferred due to tremendous MSE of the C_{50} -estimator. Furthermore, we judge that the C_{90} -estimator is outperformed by the GEN -estimator, in spite of the fact that it has not been possible to discover any significant improvement in normalized bias. On the other hand, $MSE(C_{90})$ is extremely large and in addition, $\Pi \approx 0.77$ for most values of N .

Colored Input Signal Fig. 7.13, Fig. 7.14 show the comparisons of GEN with C_{50} and C_{90} , respectively, when applying a colored input signal. The statistical tests show (on an $\alpha = 0.5\%$ significance level) that both cross-validation estimators have smaller normalized bias than the GEN -estimator. However, the mean squared errors are essentially higher, we observed:

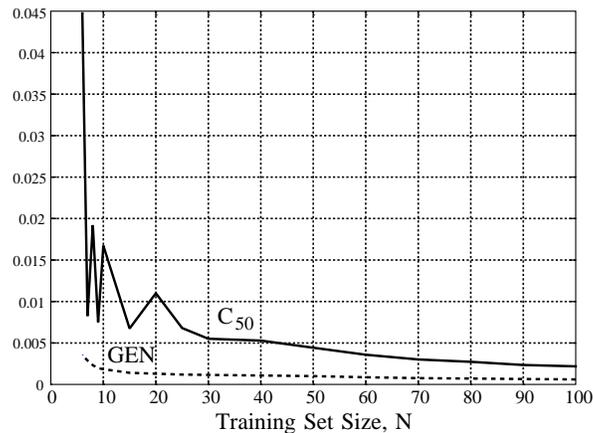
$$3 \lesssim \frac{MSE(C_{50})}{MSE(GEN)} \lesssim 13, \quad (7.86)$$

$$2 \lesssim \frac{MSE(C_{90})}{MSE(GEN)} \lesssim 9. \quad (7.87)$$

Finally, $\Pi(C_{90}) \approx 0.67$ while $\Pi(C_{50}) > 0.5$ as $N \geq 50$, cf. Fig. 7.15. The huge $MSE(C_{90})$ – relative to GEN – combined with the fact that the probability of proximity is nearly 70% for most N -values leads to that GEN is the most reliable estimator. The probability of proximity regarding the C_{50} -estimator seems not especially high. However, $MSE(C_{50})$ is still at least 3 times $MSE(GEN)$. These facts indicate that the high mean squared error (in particular, variance) is due to a severe deviation in a relatively small fraction of the Q replications (see also Fig. 7.25). It is really a matter of preference whether



(a)



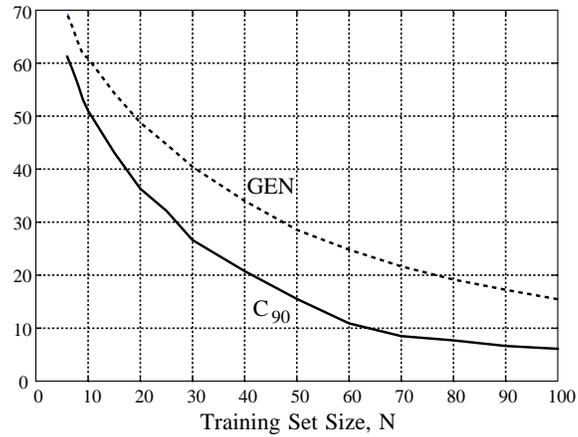
(b)

Figure 7.13: Comparison of GEN and C_{50} within the Volterra system Sec. 7.1.2.1 when applying a colored input signal.

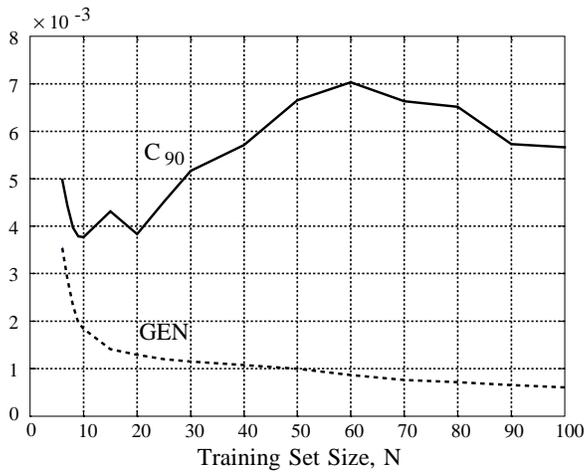
one would emphasize on mean squared error or probability of proximity – or eventually both quantities since they, to some extent, complement each other. It is appraised that huge MSE makes the C_{50} -estimator a feeble alternative to GEN . In addition, the matters discussed on p. 253 are viable in this case too.

Comparison of GEN and Leave-one-out Cross-validation When comparing the GEN and the L we used:

$$Q = \begin{cases} 10000 & 5 \leq N \leq 9 \\ 5000 & 10 \leq N \leq 100 \end{cases}, \quad (7.88)$$



(a)



(b)

Figure 7.14: Comparison of GEN and C_{90} within the Volterra system Sec. 7.1.2.1 when applying a colored input signal.

and

$$\tau = 1, 2, \dots, \min \{M, \lfloor N/10 \rfloor\}. \quad (7.89)$$

White Input Signal The results of the comparison is depicted in Fig. 7.16. On an $\alpha = 10\%$ significance level $|NB(GEN)| > |NB(L)|$ as $5 \leq N \leq 9$, while they are indistinguishable as $N \geq 10$. $MSE(L)$ is much larger than $MSE(GEN)$ as $N \lesssim 20$; otherwise they are comparable. Finally, $\Pi < 0.5$ for all N -values. Consequently, we may conclude that GEN is applicable as $N \lesssim 20$ due the fact that $MSE(L)$ is up to 3 times larger than $MSE(GEN)$. On the other hand, L is preferred otherwise. The benefit from using the

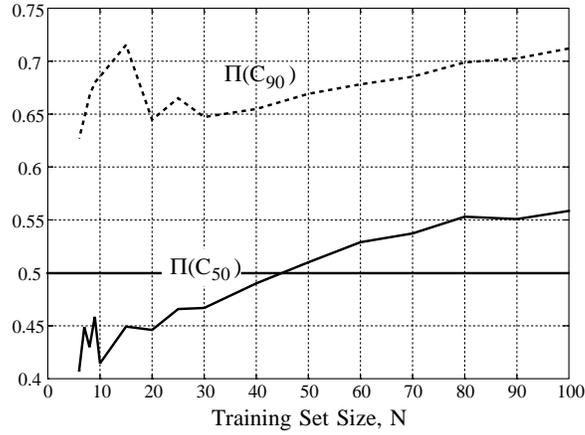


Figure 7.15: Probability of proximity concerning the C_{50} and C_{90} estimators within the Volterra system when applying a colored input signal.

L -estimator in this case is gained at the expense of significantly increased computational complexity (CP). According to sa:cpngen,

$$CP_{GEN} \approx Nm^2 \left(\frac{3}{2} itr + M + 1 \right) \quad (7.90)$$

where m is the number of weights (equal to one in the present case), itr is the number of training set replications when estimating the weights, $\mathbf{w}^{(s)}$, and M is the dependence lag (equal to zero when considered independent input) of the input vector. The complexity involved in calculating L is, however, cf. sa:cploofn

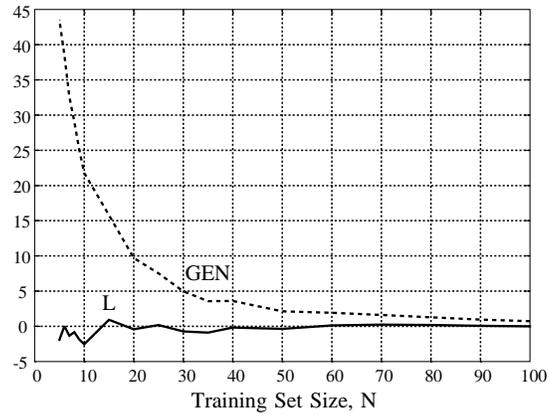
$$CP_L \approx \frac{3}{2} N^2 m^2 itr. \quad (7.91)$$

Consequently, in the present case:

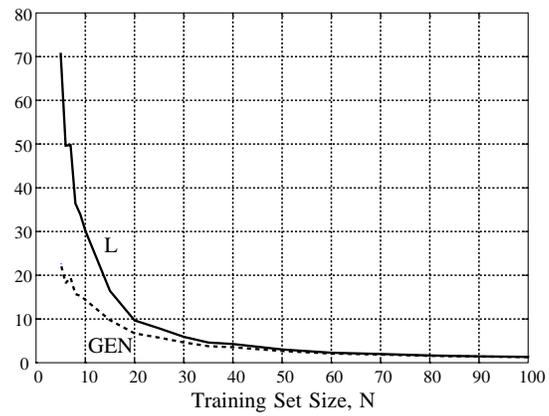
$$\frac{CP_L}{CP_{GEN}} \approx N. \quad (7.92)$$

Colored Input Signal Fig. 7.17 shows the comparison of the GEN and L estimators when employing a colored input signal. On an $\alpha = 0.5\%$ significance level we found that $|NB(GEN)| > |NB(L)|$ as $5 \leq N \leq 15$ and otherwise, vice versa. $|NB(GEN)|$ constitutes approximately the half of $|NB(L)|$ when $N = 100$. The mean squared error of the two estimators are fairly comparable for all N -values. Furthermore, $\Pi > 0.5$ as $N \geq 30$ and reaches a level at approx. 0.75 at $N = 100$. Hence, one may prefer GEN to L when $N \geq 30$. This is in spite of the fact that the computational complexity of the L -estimator is significantly larger (as mentioned above).

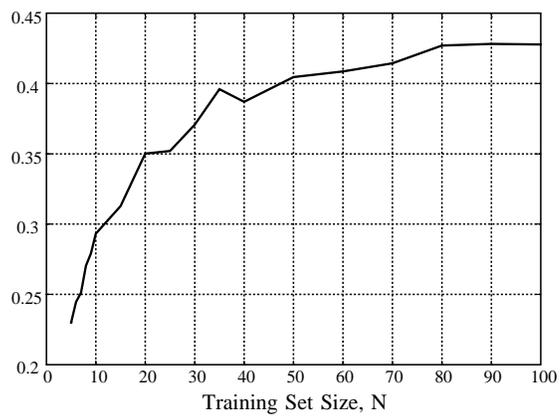
The reason for that the L -estimator works worse than when dealing with a white input signal is (cf. Sec. 6.5.4) that independence of the input signal, in fact, is an essential assumption in order to ensure low bias.



(a)

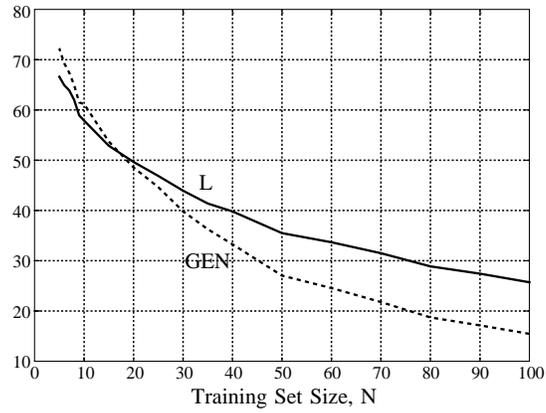


(b)

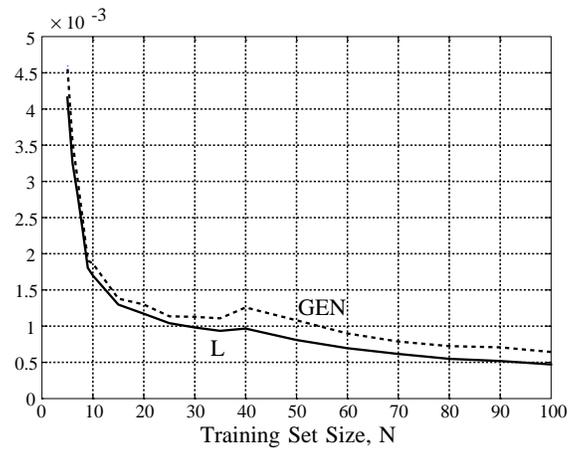


(c)

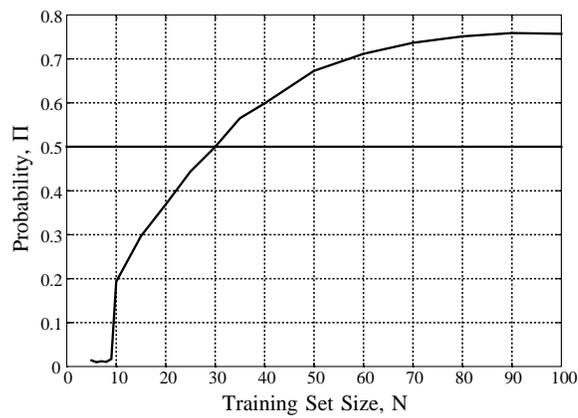
Figure 7.16: Comparison of GEN and L within the Volterra system Sec. 7.1.2.1 when applying a white input signal.



(a)



(b)



(c)

Figure 7.17: Comparison of GEN and L within the Volterra system Sec. 7.1.2.1 when applying a colored input signal.

7.1.3.2 Simple Neural Network

Let us next consider the simple neural network Sec. 7.1.2.2 containing only one weight. Contrary to the Volterra system the cost function is non-quadratic employing the neural network; however, the cost function possesses no local minima, according to Fig. 7.6. That means, an extra source of error enters the *GEN*-estimate in this case cf. Sec. 6.5.8.

We compared the *GEN*-estimator to the *FPE*-estimator using the parameters:

$$Q = 5000, \quad 5 \leq N \leq 100, \quad (7.93)$$

and

$$\tau = 1, 2, \dots, \min \{M, \lfloor N/10 \rfloor\}. \quad (7.94)$$

White Input Signal In Fig. 7.18 the comparison of *GEN* and *FPE* is shown when applying a white input signal. On an $\alpha = 0.5\%$ significance level $|NB(GEN)| < |NB(FPE)|$ as $N \geq 15$; otherwise, vice versa. At $N = 100$ the $NB(GEN)$ is approx. equal to $NB(FPE)/4$. The mean squared errors of the estimators are almost identical for all values of N considered. Furthermore, $\Pi > 0.5$ when $N \geq 20$ and around 0.56 as $40 \leq N \leq 100$. In consequence, one may prefer the *GEN*-estimator as $N \geq 20$. In addition, it should be noticed that $MSE(GEN)$ in fact is less than $MSE(FPE)$ as $N \leq 9$ which probably leads to a preference of *GEN*; however, the observed difference among the *MSE*'s is not necessarily statistical significant²³.

Colored Input Signal Fig. 7.19 shows the comparison using a colored input signal. $|NB(GEN)| < |NB(FPE)|$ as $N \geq 15$ on an $\alpha = 0.5\%$ significance level; otherwise, vice versa. The improvement in normalized bias is approximately a factor of 1.75 as $N = 100$. The mean squared errors are approximately identical; however, $MSE(GEN)$ tends to be smaller when N is small. Finally, $\Pi > 0.5$ as $N \geq 15$, and $\Pi \approx 0.65$ as $N \geq 20$. In conclusion, *GEN* may be preferred as $N \gtrsim 15$.

7.1.3.3 Polynomial System

The purpose of the polynomial system is to validate the *GEN*-estimator when considering models with more than one weight. First – of course – this is the realistic case; secondly, we may expect extra errors enter the *GEN*-estimator as mentioned in Sec. 6.5.8.4. When dealing with a white input signal we used:

$$Q = 32000, \quad 15 \leq N \leq 100; \quad (7.95)$$

however, when using colored input signal:

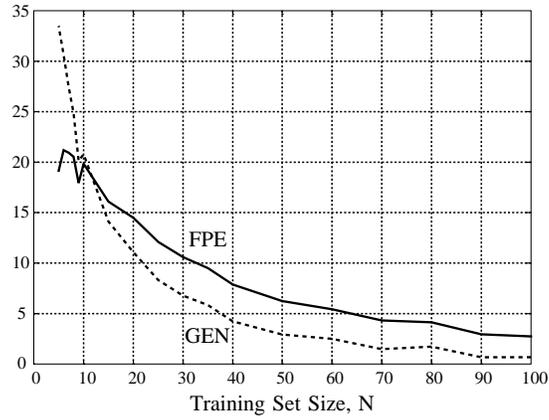
$$Q = \begin{cases} 40000 & 20 \leq N \leq 80 \\ 20000 & 81 \leq N \leq 149 \\ 10000 & 150 \leq N \leq 450 \end{cases}, \quad (7.96)$$

and

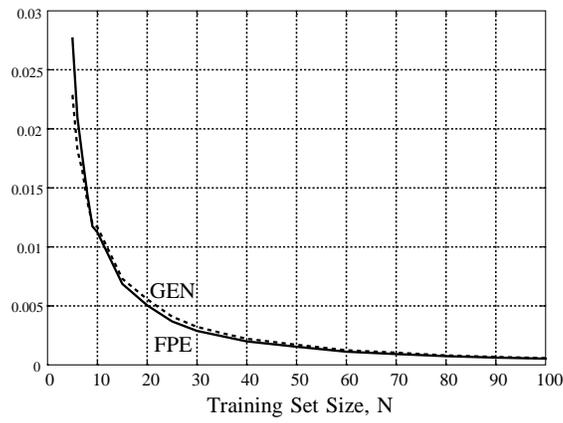
$$\tau = 1, 2, \dots, \min \{M, \lfloor N/10 \rfloor\} \quad (7.97)$$

where $M = 10$.

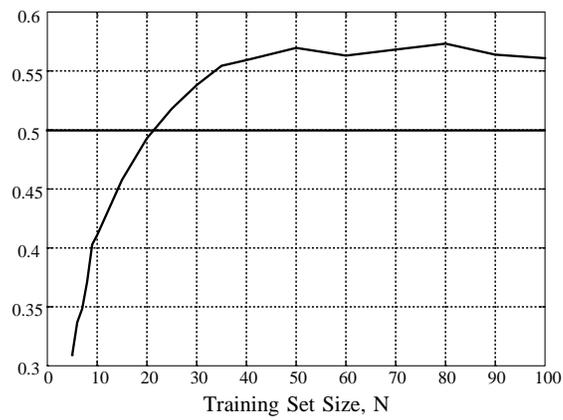
²³The large sample (i.e., $Q \rightarrow \infty$) power of testing equal variances of variables which not are Gaussian distributed is much less than testing equal means cf. e.g., [Lehmann 86, Sec. 5.3–5.5]. Consequently, such tests are omitted here (as mentioned earlier).



(a)

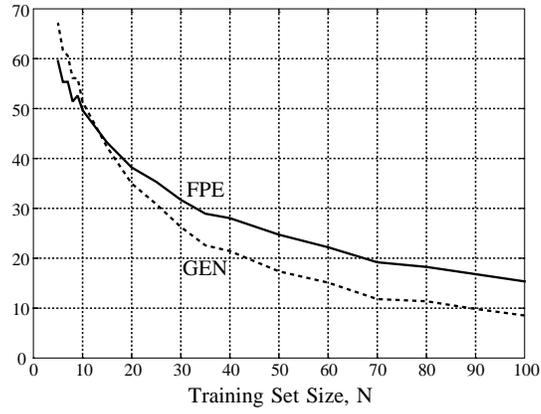


(b)

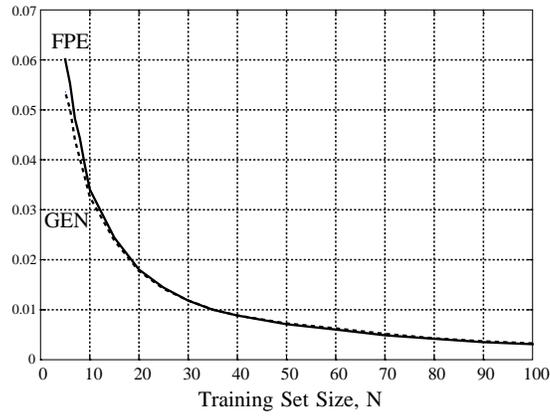


(c)

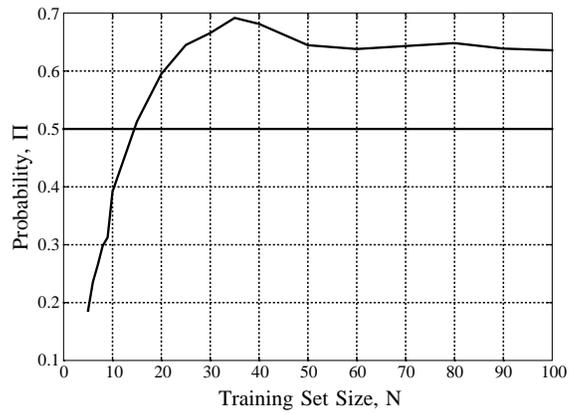
Figure 7.18: Comparison of *GEN* and *FPE* within the neural system Sec. 7.1.2.2 when applying a white input vector signal. 268



(a)



(b)

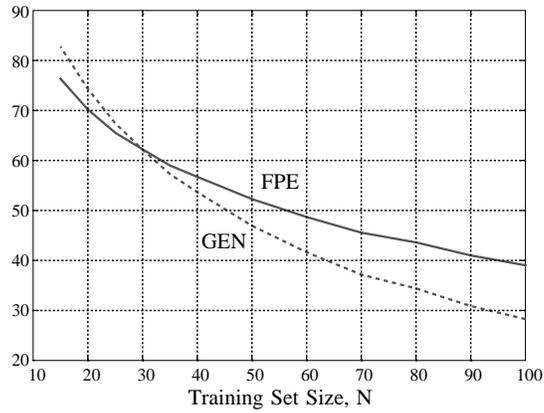


(c)

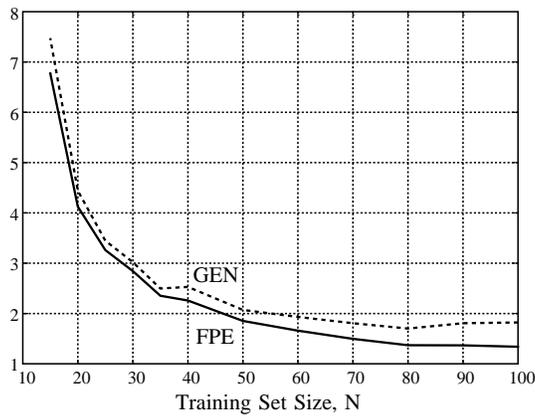
Figure 7.19: Comparison of *GEN* and *FPE* within the neural system Sec. 7.1.2.2 when applying a colored input vector signal. 269

Comparison of *GEN* and *FPE*

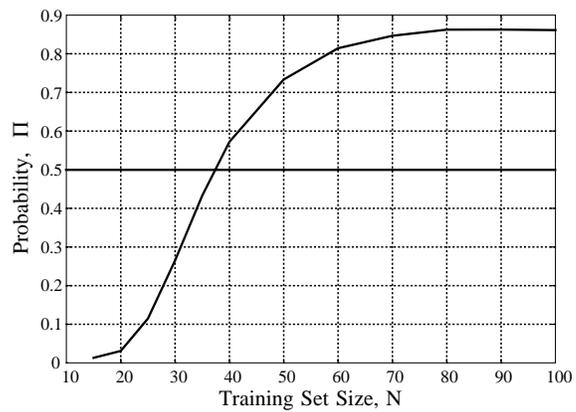
White Input Signal Fig. 7.20 shows the comparison of *GEN* and *FPE* when applying a white input signal. $|NB(GEN)| < |NB(FPE)|$ when $N \geq 30$ on an $\alpha = 0.5\%$ significance level, and vice versa otherwise. $MSE(GEN)$ is higher than $MSE(FPE)$ – up to a factor of approx. 2.5 when $N = 100$. However, $\Pi > 0.5$ as $N \geq 35$ and $\Pi \approx 0.85$ for high values of N . One may argue for a preference to *GEN* since the probability of proximity is really high, if – on the other hand – one tolerates the increased *MSE*.



(a)



(b)



(c)

Figure 7.20: Comparison of *GEN* and *FPE* within the polynomial system Sec. 7.1.2.3 when applying a white input signal.

Colored Input Signal Fig. 7.21 provides the comparison w.r.t. colored input signal. On an $\alpha = 0.5\%$ significance level $|NB(GEN)| > |NB(FPE)|$ as $20 \leq N \leq 40$; otherwise vice versa. The MSE 's are fairly comparable as $N \lesssim 250$; however, $MSE(GEN)$ tends to be approx. 30% larger as $N = 450$. $\Pi > 0.5$ as $N \gtrsim 75$ and reaches a level of approx. 0.85 as $N \geq 150$. Consequently, one may prefer the GEN -estimator as $N \gtrsim 75$ if the relatively small increase in MSE can be tolerated.

Comparison of GEN and Cross-validation

White Input Signal Fig. 7.22 and Fig. 7.23 show the comparison of GEN and C_{50} , C_{90} , respectively. Statistical testing resulted in the following relations among the normalized biases on an $\alpha = 10\%$ significance level:

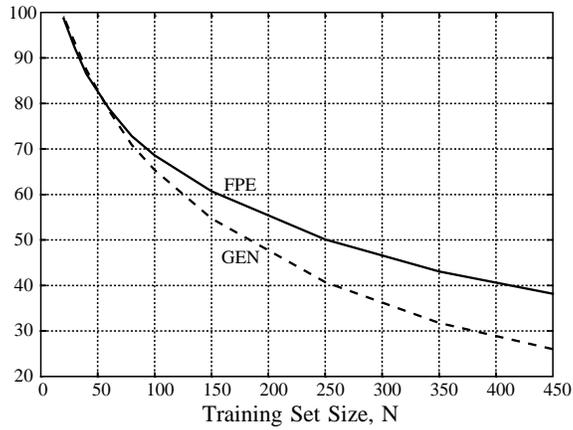
Relation	Training Set Size
$ NB(GEN) < NB(C_{50}) $	$15 \leq N \leq 20$
$ NB(GEN) \approx NB(C_{50}) $	$21 \leq N \leq 30$
$ NB(GEN) > NB(C_{50}) $	$31 \leq N \leq 100$
$ NB(GEN) > NB(C_{90}) $	$15 \leq N \leq 100$

Both $MSE(C_{50})$ and $MSE(C_{90})$ are tremendously high compared to $MSE(GEN)$ – proportions are approx. in the range $10-10^4$. The probability of proximities shown in Fig. 7.24. $\Pi(C_{50})$ never exceeds 0.4; however, due to huge MSE one would still prefer the GEN -estimator to the C_{50} -estimator. The fact that $\Pi(C_{50})$ is below 0.5 may be explained by considering the p.d.f.'s of the GEN and C_{50} estimators, say $p_{GEN}(\sigma^2)$, $p_{C_{50}}(\sigma^2)$, respectively. A sketch of the p.d.f.'s is shown in Fig. 7.25. $p_{C_{50}}(\sigma^2)$ peaks closer to $\widehat{\Gamma}_G$ ²⁴ than $p_{GEN}(\sigma^2)$ since $|NB(GEN)| > |NB(C_{50})|$. Furthermore, $p_{C_{50}}(\sigma^2)$ is more narrow than $p_{GEN}(\sigma^2)$ which result in a Π less than 0.5. On the other hand, $p_{C_{50}}(\sigma^2)$ has much longer tails which cause higher MSE .

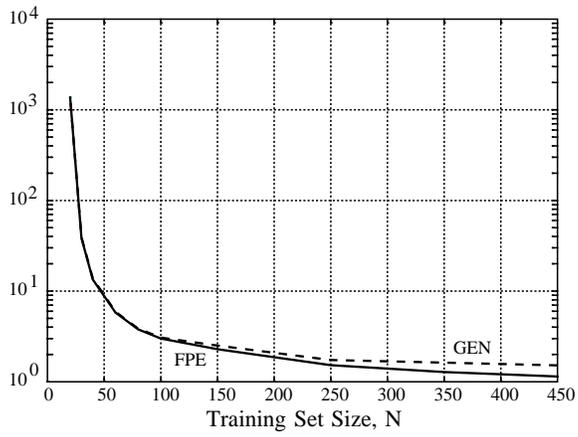
$\Pi(C_{90}) > 0.5$ for all values of N considered, and approx. at a level of 0.65 for larger N values. Hence, we conclude that GEN is superior to C_{90} in the present case.

Colored Input Signal Fig. 7.22 and Fig. 7.23 show the comparison of GEN and C_{50} , C_{90} , respectively, when applying a colored input signal. Testing the relations among normalized biases results on a $\alpha = 0.5\%$ significance level in:

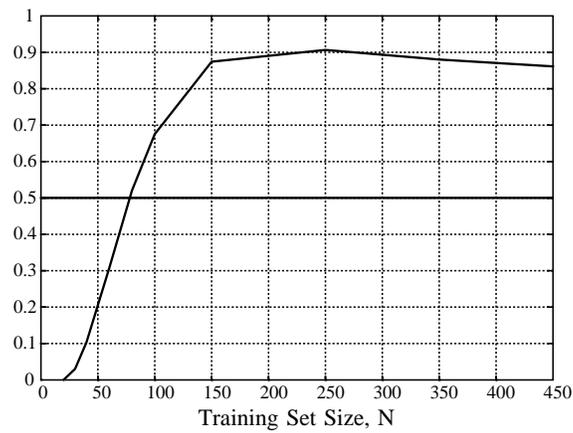
²⁴Recall that $\widehat{\Gamma}_G$ is the estimate of the average generalization error which appears by averaging the “true” generalization error over Q realizations cf. sv:gamahat.



(a)

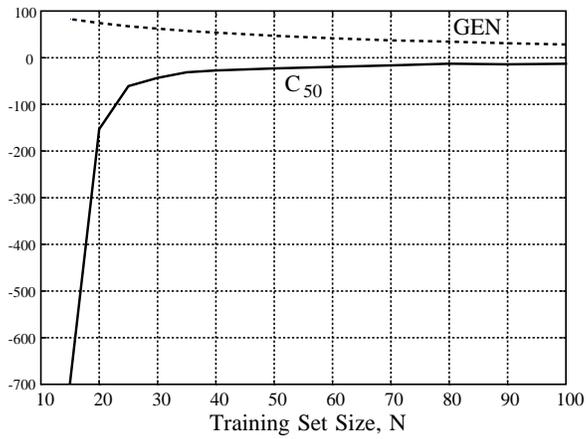


(b)

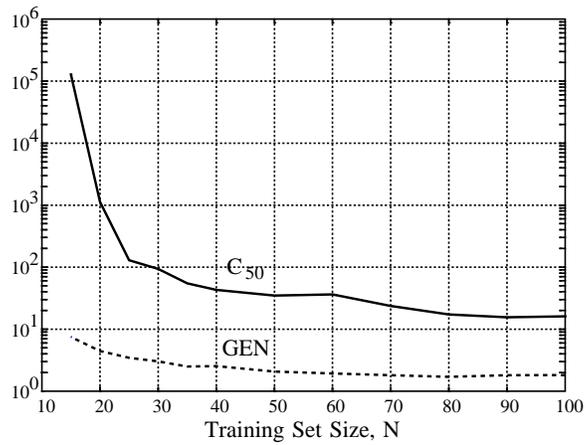


(c)

Figure 7.21: Comparison of GEN and FPE within the polynomial system Sec. 7.1.2.3 when applying a colored input signal. 273



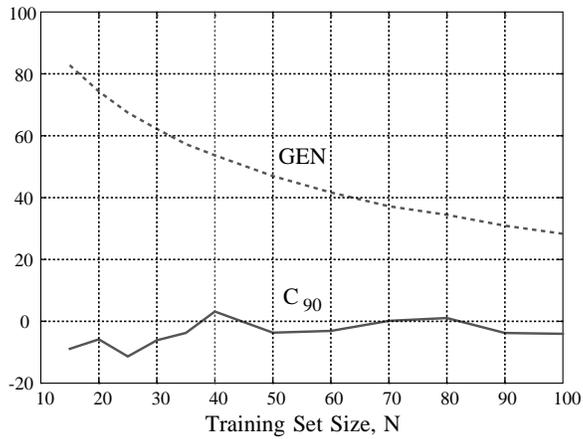
(a)



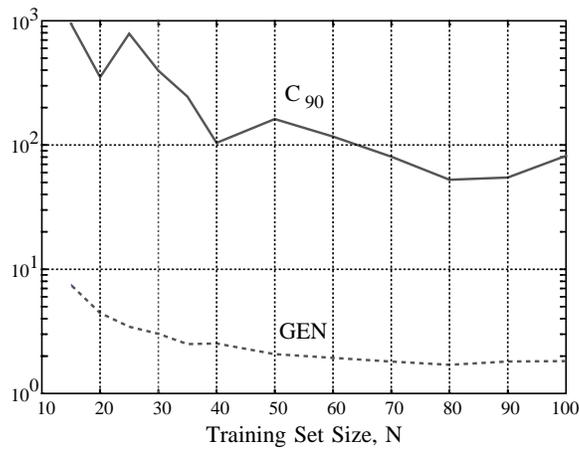
(b)

Figure 7.22: Comparison of GEN and C_{50} within the polynomial system Sec. 7.1.2.3 when applying a white input signal.

Relation	Training Set Size
$ NB(GEN) < NB(C_{50}) $	$20 \leq N \leq 60$
$ NB(GEN) > NB(C_{50}) $	$61 \leq N \leq 450$
$ NB(GEN) \approx NB(C_{90}) $	$20 \leq N \leq 30$
$ NB(GEN) > NB(C_{90}) $	$31 \leq N \leq 450$



(a)



(b)

Figure 7.23: Comparison of GEN and C_{90} within the polynomial system Sec. 7.1.2.3 when applying a white input signal.

The MSE 's of the cross-validation estimators are tremendous compared to the $MSE(GEN)$ (the proportions lie approx. in the range $10-10^7$). Fig. 7.28 show the probability of proximity. $\Pi(C_{90}) > 0.5$ as $N \gtrsim 60$ and reach a level of 0.7 for large N . As in the white input case, $\Pi(C_{50})$, never exceeds 0.5. However, the huge MSE afflicted with the cross-validation estimators leads to the conclusion that GEN is a better choice.

7.1.3.4 Discussion

In summary, from the numerical results we draw the following conclusions:

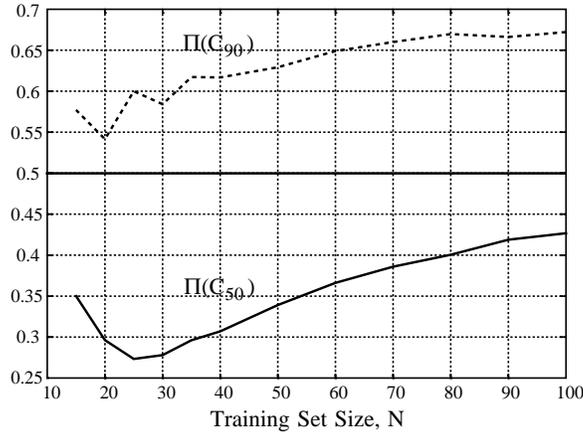


Figure 7.24: Probability of proximity concerning the C_{50} and C_{90} estimators within the polynomial system Sec. 7.1.2.3 when applying a white input signal.

- The *GEN*-estimator is tested under various model conditions which comprise:
 - Incomplete LX- and NN-models.
 - White and colored input.
 - No regularization²⁵.

In most cases quantitative arguments lead to that the *GEN*-estimator is a profitable alternative to the *FPE*-estimator, cross-validation estimators, and the leave-one-out cross-validation estimator.

- Comparing *GEN* and *FPE* we observed that the normalized bias of *GEN* is somewhat below the normalized bias of *FPE* as the size of the training set, N is large. When N is small the *GEN*-estimator typically has higher NB . This is due to the fact that *GEN*-estimator – in contrast to the *FPE*-estimator – uses several estimated quantities which consequently give rise to additional error; however, the error is for large N overshadowed by the fact that the *GEN*-estimator takes the matters concerning incomplete models into account. According to Sec. 6.5.8.4 the errors decrease with N ; on the other hand, they increase with both the number of weights, m , and the dependence lag (correlation length of the input), M . In particular, this phenomenon is pronounced when considering the polynomial system (see Fig. 7.20 and Fig. 7.21). When applying a white input the *GEN*-estimator is practicable as $N \geq 35$, while $N \gtrsim 75$ is required in the colored case.

The mean squared error of the *GEN*-estimator is comparable or slightly higher than that of *FPE*. Furthermore, the probability of proximity – i.e. the probability that *GEN* is closer to the “true” average generalization error than *FPE* – is in most cases somewhat above 0.5.

²⁵Testing the *GEN*-estimator when utilizing regularization is left for future work.

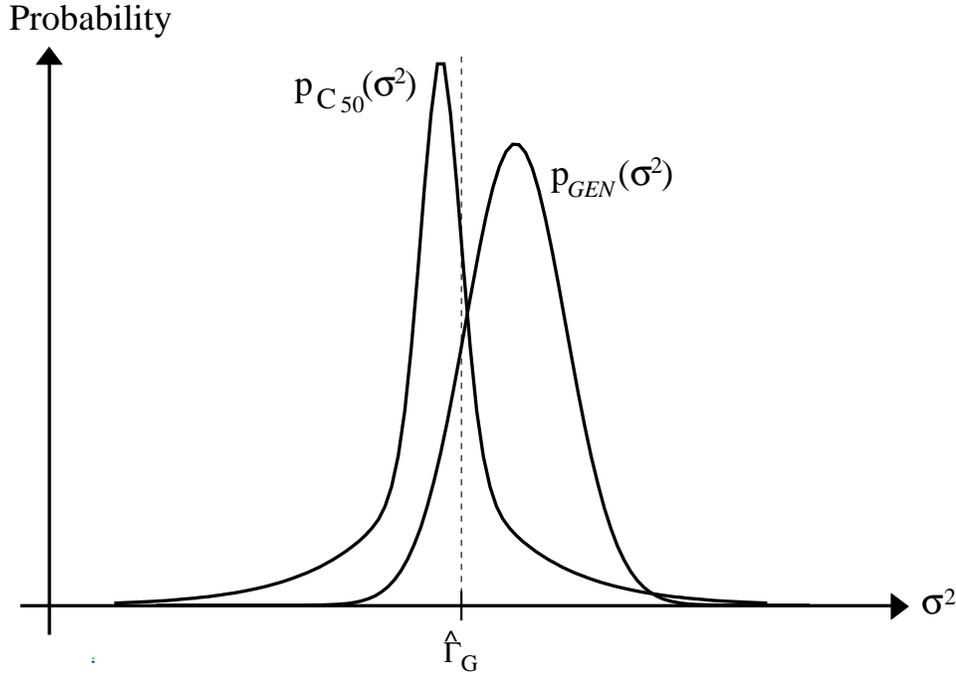
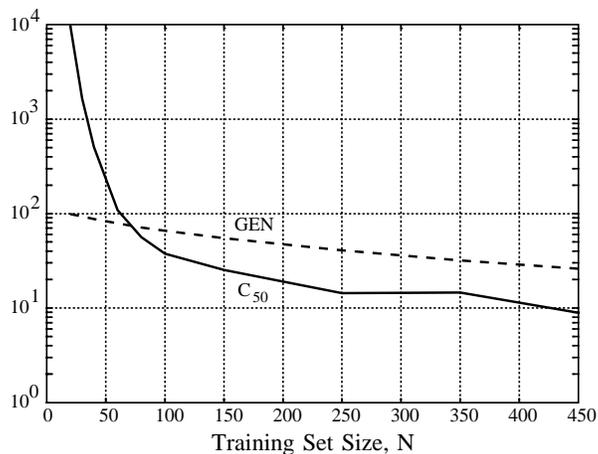
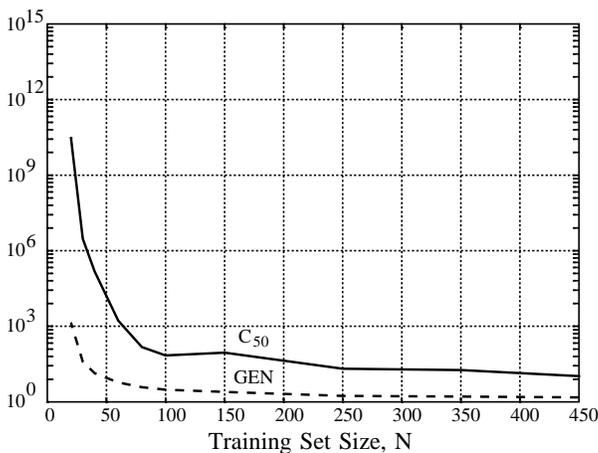


Figure 7.25: Sketch of $p_{GEN}(\sigma^2)$ and $p_{C_{50}}(\sigma^2)$.

- The comparison of *GEN* and the cross-validation estimators C_{50} , C_{90} is characterized by the fact that the (absolute) normalized bias of the cross-validation estimators are significantly lower than that of *GEN*. However, this advantage is completely blurred by the circumstance that the mean square error of the cross-validation estimators are tremendously higher compared to *GEN*. The probability of proximities, Π , seem – in particular regarding the C_{50} -estimator – relatively low, and sometimes even below 0.5. However, it is appraised that conclusions should not be drawn from the Π -measure solely – all measures have to be taken into consideration. Consequently, the huge mean square error discard the cross-validation estimators as alternatives to the *GEN*-estimator. It should be accentuated, however, that the conclusions are subjective – in fact – it is a matter of taste which particular measures one aims at.
- Concerning the comparison of *GEN* and the leave-one-out cross-validation estimators we noticed that when dealing with a white (independent) input signal the *L*-estimator outperformed the *GEN*-estimator, especially as N is large. However, considering small N -values the *L*-estimator had significantly higher *MSE*. On the other hand, dealing with colored input signal – which is the common case within signal processing – the normalized bias of the *GEN* is smaller for most N -values, the *MSE* was slightly higher, and Π somewhat above 0.5. This lead to a preference to *GEN*. Moreover, it was emphasized that the computational complexity of the *L*-estimator is essentially larger than that of *GEN*.



(a)

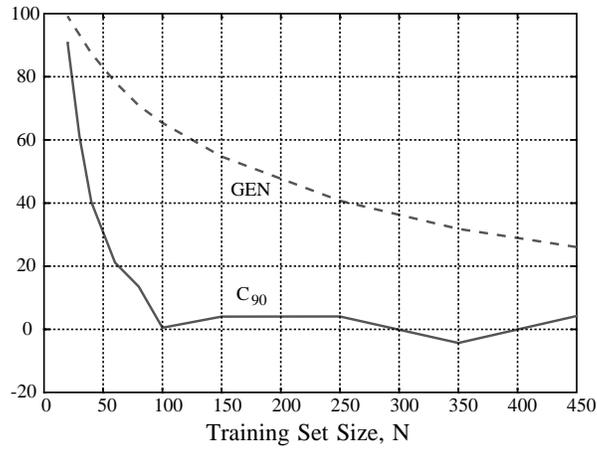


(b)

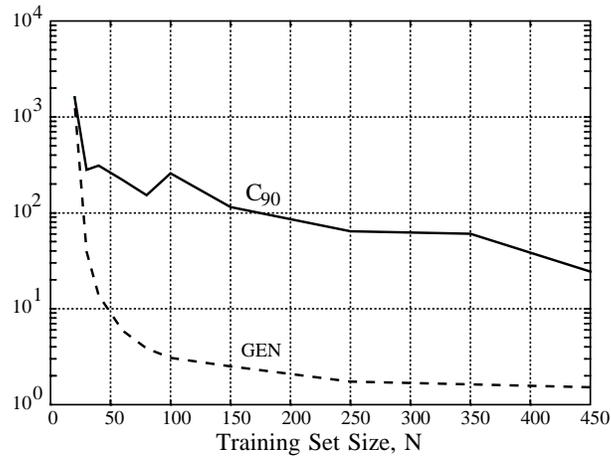
Figure 7.26: Comparison of GEN and C_{50} within the polynomial system Sec. 7.1.2.3 when applying a colored input signal.

Finally, notice that the conclusions merely are drawn from the study of the selected simulation examples. That is, there may still be cases which do not fulfil the assumptions which underlie the derivation of the GEN -estimator, e.g., the basic second order cost function approximation (see Sec. 6.5.8) may be violated.

In addition, recall that the essential application of the GEN -estimator is architecture synthesis. However, this topic is left for future research.



(a)



(b)

Figure 7.27: Comparison of *GEN* and *C₉₀* within the polynomial system Sec. 7.1.2.3 when applying a colored input signal.

7.2 Validation of Statistically based Pruning Algorithms

This section provides a validation of statistically based pruning procedures suggested in Sec. 6.7. Recall that the pruning procedure is divided into two parts:

1. Selection of a WDV-algorithm which results in a set of proper weight deletion vectors (WDV's). A particular WDV contains the indices of weights under consideration for deletion. In Sec. 6.7.2.1– Sec. 6.7.2.3 different WDV-algorithms are discussed. Here we merely consider:

- The approach based on individual deletion cf. Sec. 6.7.2.2 which is referred to

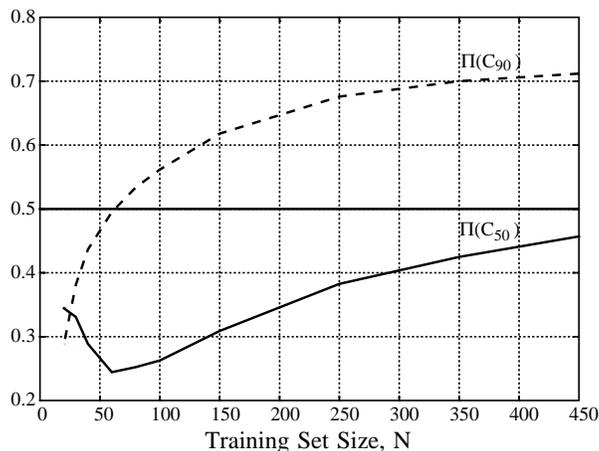


Figure 7.28: Probability of proximity concerning the C_{50} and C_{90} estimators within the polynomial system Sec. 7.1.2.3 when applying a colored input signal.

as WDV1.

- The approach based on a block Hessian structure cf. Sec. 6.7.2.3 which is referred to as WDV2.
2. Choice of method for testing if the weights contained in the WDV's can be removed from the filter architecture. We employ two different methods:
 - The *Statistical Pruning Algorithm* (SPA) cf. Sec. 6.7.3 which is based on a statistical testing framework.
 - The *Generalization Error based Pruning Algorithm* (GEPA) cf. Sec. 6.7.4 which is based on comparing the estimated generalization error before and after removal of the weights pointed out by the particular WDV.

The general short form of the pruning procedures is as follows:

1. Choose a filter architecture parametrized by an m -dimensional weight vector.
2. Estimate the filter weights, i.e., determine $\hat{\mathbf{w}}$.
3. Select a WDV-algorithm and form the set of optimal WDV's

$$\mathcal{Q} = \{\mathbf{d}_1^*, \mathbf{d}_2^*, \dots, \mathbf{d}_{m-1}^*\} \quad (7.98)$$

where \mathbf{d}_n^* is the optimal WDV containing n weights.

4. For all $1 \leq n \leq m - 1$ decide if the weights pointed out by \mathbf{d}_n^* can be deleted by using statistical testing or generalization error estimates.
5. Reestimate the weights of the pruned model, i.e., determine $\hat{\mathbf{w}}_{\mathcal{H}}$ by using the procedure given in Sec. 6.6.5.

7.2.1 Simulation Setup

Even though the suggested pruning procedures are viable for general incomplete NN-models we will merely provide preliminary results covering the case of a complete (over-parameterized) LN-models.

Data are, as in Sec. 7.1.2.3, generated by a $l = 4$ order Chebyshev system:

$$\begin{aligned} y(k) &= f_n(\mathbf{z}(k); \mathbf{w}) + \varepsilon(k) \\ &= (\mathbf{w}^\circ)^\top \mathbf{v}(k) + \varepsilon(k) \end{aligned} \quad (7.99)$$

where

- $\mathbf{z}(k) \in \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{z}})$ is a two-dimensional ($p = 2$) i.i.d. input vector signal where

$$\Sigma_{\mathbf{z}} = \begin{bmatrix} 0.2 & 0.1 \\ 0.1 & 0.2 \end{bmatrix}. \quad (7.100)$$

- $\mathbf{v}(k)$ contains the polynomial terms – in this case Chebyshev polynomials (time index k omitted):

$$\begin{aligned} \mathbf{v} &= [1, T_1(z_1), T_2(z_1), \dots, T_l(z_1), \\ &T_1(z_2), T_1(z_2)T_1(z_1), \dots, T_1(z_2)T_{l-1}(z_1), \\ &T_1(z_3), T_1(z_3)T_1(z_1), \dots, T_1(z_3)T_{l-1}(z_1), \\ &\vdots \\ &T_{l-1}(z_{p-1})T_1(z_p), \\ &T_l(z_p)]^\top \end{aligned} \quad (7.101)$$

where $T_r(\cdot)$ is the r 'th order Chebyshev polynomial (see Sec. 3.2.1.3).

- \mathbf{w}° is the true weight vector with dimension $m = C_{l+p,l} = C_{6,4} = 15$. We consider two cases:

System #1: All 3'rd and 4'th order terms in the $\mathbf{v}(k)$ vector are removed; consequently, the system is of 2'nd order. The true weight vector obeys:

$$\mathbf{w}^\circ = [1.5, 2, 1, 0, 0, 0.5, 0.25, 0, 0, 0.4, 0, 0, 0, 0, 0]^\top. \quad (7.102)$$

System #2: All terms in the $\mathbf{v}(k)$ which depends on $z_2(k)$ are removed and the true weight vector is given by:

$$\mathbf{w}^\circ = [1.5, 2, 0.25, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^\top. \quad (7.103)$$

- The inherent noise, $\varepsilon(k) \in \mathcal{N}(0, \sigma_\varepsilon^2)$, is an i.i.d. Gaussian sequence independent of $\mathbf{z}(k)$ and with $\sigma_\varepsilon^2 = 0.15$.

The complete LN-model of sv:prusys is given by:

$$y(k) = \mathbf{w}^\top \mathbf{v}(k) + e(k; \mathbf{w}) \quad (7.104)$$

where $e(k; \mathbf{w})$ is the error signal.

The weights are estimated by minimizing the LS cost function, $S_N(\mathbf{w})$, according to a training set: $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, $k = 1, 2, \dots, N$, where $N = 500$. According to Sec. 5.2 (pa:wdestlin) the estimated weights obey:

$$\hat{\mathbf{w}} = \mathbf{H}_N^{-1} \cdot \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)y(k) \quad (7.105)$$

where \mathbf{H}_N is the Hessian matrix:

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)\mathbf{z}^\top(k). \quad (7.106)$$

The WDV-algorithm based on individual deletion (WDV1) estimates the optimal WDV by (see Sec. 6.7.2.2):

$$\mathbf{d}_n^* = \arg \min_{\mathbf{d}_n} \left[\sum_{r=1}^n \tilde{\varrho}_{d_n, r} \right] \quad (7.107)$$

where $\tilde{\varrho}_{d_n, r}$ is the modified saliency defined by:

$$\tilde{\varrho}_{d_n, r} = \frac{\hat{w}_{d_n, r}^2}{P_{d_n, r, d_n, r}} \quad (7.108)$$

where $P_{i, i}$ is the i 'th diagonal element of $\mathbf{H}_N^{-1}(\hat{\mathbf{w}})$. The set of optimal WDV's, \mathcal{Q} , is simply obtained by ranking the modified saliencies.

In WDV2 based on a block Hessian approach the optimal WDV is found according to:

$$\mathbf{d}_n^* = \arg \min_{\mathbf{d}_n} \delta S(\mathbf{d}_n), \quad (7.109)$$

where

$$\delta S(\mathbf{d}_n) = \hat{\mathbf{w}}^\top \mathbf{\Xi}^\top \mathbf{\Xi} \mathbf{H}_N(\hat{\mathbf{w}}) \mathbf{\Xi}^\top \mathbf{\Xi} \hat{\mathbf{w}}, \quad (7.110)$$

and $\mathbf{\Xi}$ is the restriction matrix specified by the current WDV, \mathbf{d}_n . The optimization is solved by using the Stack Algorithm cf. p. 211 with stack-size $\varsigma = 4(m - n + 1)$.

The decision whether the weight specified by a particular optimal WDV \mathbf{d}_n^* should be removed from the model or not is in the SPA (cf. Sec. 6.7.3) based on the test statistic:

$$T = \hat{\mathbf{w}}^\top \mathbf{\Xi}^\top \left(\mathbf{\Xi} \hat{\mathbf{V}} \mathbf{\Xi}^\top \right)^{-1} \mathbf{\Xi} \hat{\mathbf{w}} \quad (7.111)$$

where $\hat{\mathbf{V}}$ is the estimated covariance matrix of the weight estimate. Since the model is complete Th. 6.16 gives:

$$\hat{\mathbf{V}} = \frac{S_N(\hat{\mathbf{w}})}{N - m} \cdot \mathbf{H}_N(\hat{\mathbf{w}})^{-1}. \quad (7.112)$$

The weights are removed if $T > \chi_{1-\alpha}^2(n)$ where $\chi_{1-\alpha}^2(n)$ is the $1 - \alpha$ fractile of the χ^2 -distribution with n degrees of freedom. We employed a significance level $\alpha = 10\%$.

The GEPA (cf. Sec. 6.7.4) is based on comparing the *GEN*-estimates of the models which appear by removing the weights suggested by the optimal WDV's. The architecture with minimal generalization error is finally selected. As the model is complete the particular variant of *GEN*-estimator employed here is the *FPE*-estimator.

The pruning procedures are validated by comparing the estimated average generalization error of the unpruned model, $\hat{\Gamma}_G$, with that of the pruned model, say $\hat{\Gamma}_{G,\mathcal{H}}$. In particular, we compare the *relative improvement* defined by:

$$\Delta\hat{\Gamma}_G = \frac{\hat{\Gamma}_G - \hat{\Gamma}_{G,\mathcal{H}}}{\hat{\Gamma}_G} \cdot 100\%. \quad (7.113)$$

The estimates are obtained by averaging over $Q = 261$ independent training sets, $\mathcal{T}^{(s)}$, $s = 1, 2, \dots, Q$. Accordingly (see p. 237):

$$\hat{\Gamma}_G = \langle G(\hat{\mathbf{w}}^{(s)}) \rangle = \frac{1}{Q} \sum_{s=1}^Q G(\hat{\mathbf{w}}^{(s)}) \quad (7.114)$$

where $G(\hat{\mathbf{w}}^{(s)}) = E\{e^2(\hat{\mathbf{w}}^{(s)})\}$ is the generalization error, and $\hat{\mathbf{w}}^{(s)}$ are the weights of the unpruned model estimated from $\mathcal{T}^{(s)}$. Due to the knowledge of the system, the model, and the distributions of the input and error signals $G(\cdot)$ can be determined analytically according to sv:gepol and the results of App. E. Similarly:

$$\hat{\Gamma}_{G,\mathcal{H}} = \langle G(\hat{\mathbf{w}}_{\mathcal{H}}^{(s)}) \rangle = \frac{1}{Q} \sum_{s=1}^Q G(\hat{\mathbf{w}}_{\mathcal{H}}^{(s)}) \quad (7.115)$$

where $\hat{\mathbf{w}}_{\mathcal{H}}^{(s)}$ are the *retrained* weights of the pruned model.

In addition, we evaluate the *average number of model errors*, $\langle \Delta m \rangle$, where Δm is defined as the sum of the number of erroneously deleted weights and the number of truly zero weights which have not been removed.

7.2.2 Results

Table 7.1 show the result obtained by using the pruning procedures mentioned above. The

System	WDV-algorithm	Pruning Alg.	$\Delta\hat{\Gamma}_G$ (%)	$\langle \Delta m \rangle$
#1	WDV1	SPA	52.3	2.63
		GEPA	38.4	3.53
	WDV2	SPA	66.1	1.55
		GEPA	55.0	2.42
#2	WDV1	SPA	40.0	2.94
		GEPA	34.7	3.65
	WDV2	SPA	63.3	1.31
		GEPA	44.5	2.44

Table 7.1: Results obtained by using various pruning and WDV algorithms. The system number refers to sv:prusys1, (7.103), respectively. $\Delta\hat{\Gamma}_G$ is the percentage improvement in average generalization error, and $\langle \Delta m \rangle$ is the average number of model errors.

simulations indicate:

- The statistical pruning algorithm (SPA) is superior to the generalization error based pruning algorithm (GEPA) since both $\Delta\hat{\Gamma}_G$ is larger and $\langle \Delta m \rangle$ is smaller in all cases.

- WDV2 works better than WDV1 when using both SPA and GEPA. This is at the expense of increased computational complexity by running the stack algorithm contrary to performing a simple ranking of modified saliencies. The additional computational complexity was small in these simulations; however, this may not be true when dealing with more complex problems. On the other hand, recall cf. Sec. 6.7.2.3 that the computational complexity partly can be controlled by specifying the stack-size, and partly by introducing a modified search cost.

7.3 Summary

The scope of the chapter was partly validation of the *GEN*-estimator proposed in Sec. 6.5.8, and partly validation of statistically based pruning procedures.

The potential of the *GEN*-estimator lies in considering incomplete, NN-models, i.e., models which do not model the present task perfectly. Dependent on assumptions on the model, the correlation of the input signal etc., various *GEN*-estimators appear (see also Table 6.1). In outline, we distinguish between NN- and LX-models, and between white and colored input signals. Three simple nonlinear system and models, which cover all possible cases, were considered for numerical investigation. An advantageous property of the suggested systems and models is that the “true” generalization error can be determined analytically, thus constituting a common reference w.r.t. comparison of generalization error estimators. The *GEN*-estimator has been compared to the *FPE*-estimator, cross-validation estimators (*C*-estimators), and the leave-one-out cross-validation estimator (*L*-estimator). The comparison is based on three different measures:

- The normalized bias (*NB*), which measures the percentage, average²⁶ deviation of a particular generalization error estimator relative to the “true” average generalization error.
- The mean squared error (*MSE*) equal to the average squared deviation of a particular generalization error estimator relative to the “true” average generalization error.
- The probability of proximity (Π) which measures the probability that the *GEN*-estimator is closer to the “true” average generalization error than another particular estimator.

We concluded that the *GEN*-estimator is superior to the *FPE*-estimator since the *NB* is lower for most training set sizes, the *MSE* is comparable – or slightly higher – and Π is somewhat above 0.5. The comparison of the *GEN* and cross-validation estimators are characterized by the fact that *MSE* of the *C*-estimators are huge compared to that of *GEN*. This excludes cross-validation as a proper alternative to the *GEN*-estimator²⁷. Finally, comparison of the *L*- and *GEN*-estimators showed that the *GEN*-estimator should be preferred in the case of colored input signals. In fact, correlation in the input signal is in conflict with the underlying idea of leave-one-out cross-validation.

Various statistical based pruning procedures were validated. This comprises: the Statistical Pruning Algorithm (SPA) Sec. 6.7.3 and the Generalization Error based Pruning

²⁶The average is performed w.r.t. various training sets with fixed sizes.

²⁷However, notice that this statement is valid only for moderate training set sizes. If a huge training set is available cross-validation is in fact the natural choice since it forms an unbiased and consistent estimator. Moreover, the computational complexity is low.

Algorithm (GEPA) Sec. 6.7.4. Both algorithms require a choice of a WDV-algorithm. The outcome of a WDV-algorithm is set of WDV's where a particular WDV contains the indices of weights under consideration for deletion. We employed partly the WDV1 based on individual deletion, and partly the WDV2 based on a block Hessian structure. Simulations indicated that using the SPA with WDV2 resulted in best performance.

CHAPTER 8

SIMULATION OF ARTIFICIAL SYSTEMS

In this chapter the performance of various algorithms are studied by simulating artificial systems. This is about:

- The weight initialization algorithm given in Sec. 5.3.2.
- Algorithms for parameter estimation cf. Ch. 5.
- Numerical experiments on using multi-layer perceptron neural networks for various signal processing tasks.

8.1 Validating the Weight Initialization Algorithm

In this section we demonstrate the usefulness of the weight initialization algorithm for 2-layer perceptron neural network described in Sec. 5.3.2. The weight initialization algorithm is compared to employing random weight initialization.

The example chosen in this simulation is described in Sec. 3.2.2 and consists in modeling the one-dimensional mapping $y(k) = g(x(k))$ where $y(k)$, $x(k)$ are the output and input signals, respectively, and $g(\cdot)$ is a nonlinear mapping given by `mf:gxexam`. We generated a training set containing $N = 500$ samples.

A $[1, 8, 1]$ -network (tangent hyperbolic neurons) is trained by using partly the Normalized Stochastic Gradient (NSG) algorithm Sec. 5.4, and partly the Recursive Gauss-Newton algorithm with Bierman Factorization (RGNB-algorithm) Sec. 5.7. The parameter setup concerning the NSG-algorithm is given in Table 8.1. The weight initialization algo-

Parameter	Interpretation	Value
itr	Number of iterations	149
α	Normalized step-size	0.2
κ	Regularization parameter	0

Table 8.1: Parameter setup for the NSG-algorithm.

rithm was implemented with the parameters: $h_{\max} = 0.8$ and $\delta = 10\%$. When employing

random initialization the weights were independently drawn from a uniform distribution over the interval $[-0.1, 0.1]$. The parameter setting w.r.t. the RGNB-algorithm is given in Table 8.2. The weight initialization algorithm and the random initialization were imple-

Parameter	Interpretation	Value
itr	Number of iterations	29
$\lambda(\ell)$	Instan. forgetting factor	$1 - 0.05 \cdot 0.9987^\ell$
γ	Regularization adjustment	0.01

Table 8.2: Parameter setup for the RGNB-algorithm.

mented as above. The only exception is that the bias weight of the linear output neuron was set equal to the time-average of $y(k)$ when considering random initialization.

The performance of the network was evaluated after each pass of the training set by calculating the *relative error index* defined by:

$$E = \sqrt{\frac{\frac{1}{N} \sum_{k=1}^N e^2(k)}{\frac{1}{N-1} \sum_{k=1}^N [y(k) - \langle y(k) \rangle]^2}}. \quad (8.1)$$

The numerator is the mean square training error and the denominator is the variance of the output signal as $\langle \cdot \rangle$ denotes time-averaging. E thus defines a noise-to-signal ratio.

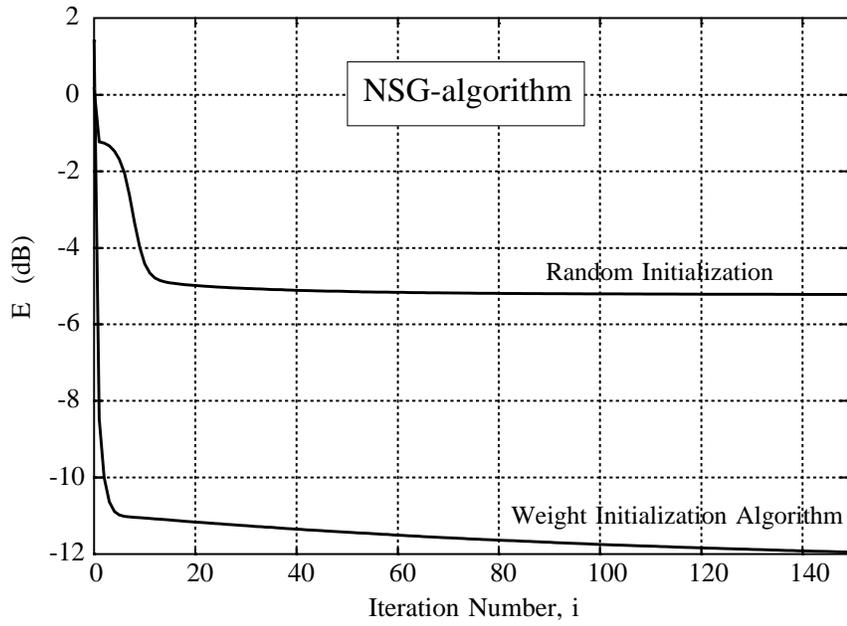
Fig. 8.1 shows plots of E as a function of the iteration number. It is observed that the benefits of using the weight initialization algorithm seems most noticeable when employing the NSG-algorithm. This is due to the fact that the NSG-algorithm is more sensitive to high eigenvalue spread of the Hessian matrix (of the cost function). We expect that the eigenvalue spread is reduced when using the weight initialization algorithm since the objective of the algorithm is to provide for small resemblance among the hidden neuron responses.

In summary, the simulation shows that the weight initialization algorithm provides a practicable alternative to pure random weight initialization.

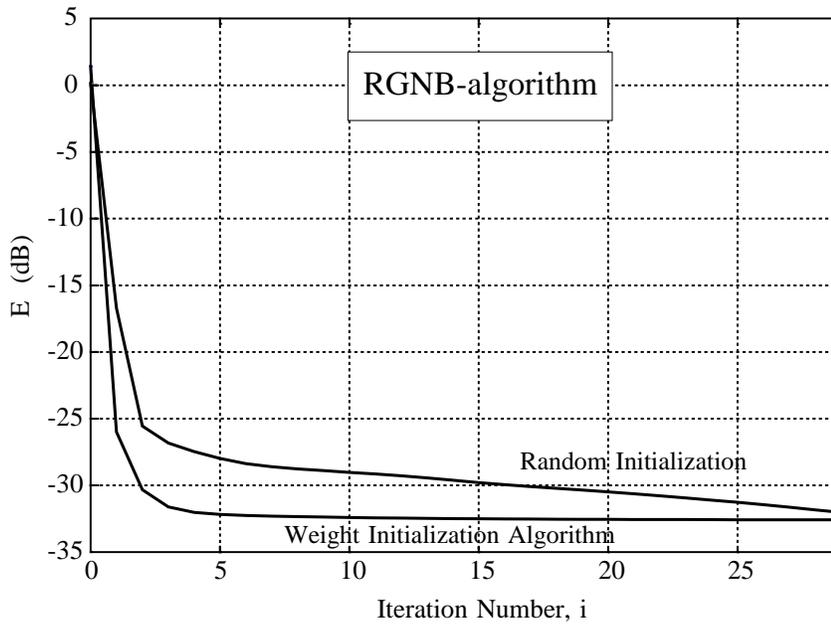
8.2 Comparison of Parameter Estimation Algorithms

This section provides an illustrative comparison of various parameter estimation algorithms presented in Ch. 5. As test example we choose the modeling of the Morison system described in Sec. 8.3.1 below. Furthermore, we used derivative preprocessing with parameters given in Table 8.5. Hence, we generated a training set $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, $k = 1, 2, \dots, N$, with $N = 500$ training data. $\mathbf{z}(k)$ is the $p = 3$ dimensional input vector signal and $y(k)$ is the output signal. The considered model is a 2-layer $[3, 10, 1]$ perceptron neural network (cf. Sec. 3.2.2) containing $m = 51$ weights. The performance of the network is evaluated by calculating the relative cross-validation error index, E cf. si:ecdef.

At first we present a comparison among the Stochastic Gradient (SG) algorithm, the Normalized Stochastic Gradient (NSG) algorithm, and the Recursive Gauss-Newton Algorithm with Bierman Factorization (RGNB). The comparison is based on evaluation of



(a)



(b)

Figure 8.1: Relative error index, E , plotted as a function of the number iterations, when using the weight initialization algorithm and random weight initialization, respectively. The considered algorithms are the normalized SG-algorithm and the RGNB-algorithm.

E as a function of computational complexity which is defined as the total number of multiplications and divisions¹. In Table 8.3 the computational complexity of the algorithms (with the configuration mentioned above) is listed. The complexities are found by using `pa:cpsg2lay`, (5.141), and (5.200). The initial weights of the network was independently

Algorithm	Computational Complexity
SG	$i \cdot 1.637 \cdot 10^5$
NSG	$i \cdot 2.095 \cdot 10^5$
RGNB	$i \cdot 2.508 \cdot 10^6$

Table 8.3: Computational complexity measured as the number of multiplications and divisions when using a $[3, 10, 1]$ -network and $N = 500$ training samples. i denotes the number of iterations, i.e., the number of training set replications.

drawn from a uniform distribution over the interval $[-0.01; 0.01]$, and the setting of the algorithm parameters is given in Table 8.4. Fig. 8.2 shows the relative cross-validation

Algorithm	Parameter	Interpretation	Value
SG	itr	Number of iterations	999
	μ	Step-size	0.05
	κ	Regularization parameter	0
NSG	itr	Number of iterations	999
	α	Normalized step-size	0.5
	κ	Regularization parameter	0
RGNB	itr	Number of iterations	34
	$\lambda(\ell)$	Instan. forgetting factor	$1 - 0.1 \cdot 0.9982^\ell$
	γ	Regularization adjustment	0.01

Table 8.4: Algorithm parameter setup.

error index versus the computational complexity.

Notes:

- Within each algorithm the algorithm parameters are adjusted in order to yield optimal algorithm performance. The adjustment is done by searching over possible candidate values; hence, an exhaustive search will possibly lead to a better result than that reported here. On the other hand, one would indeed dismiss an algorithm which requires that the parameters are carefully fine-tuned in order to avoid poor performance. The SG-algorithm requires a careful adjustment of the step-size, and moreover, recall that the magnitude of the step-size is impossible to predetermine. The adjustment of the normalized step-size (NSG-algorithm), α , seems to be an easier task. This is due to the fact that convergence interval: $0 \leq \alpha < 2$, cf. Sec. 5.4, is less dependent of the actual problem. Finally, the parameters of the RGNB-algorithm all have a very clear interpretation, and are normally easily

¹Recall cf. Ch. 5 that one division per definition is equivalent to 10 multiplications.

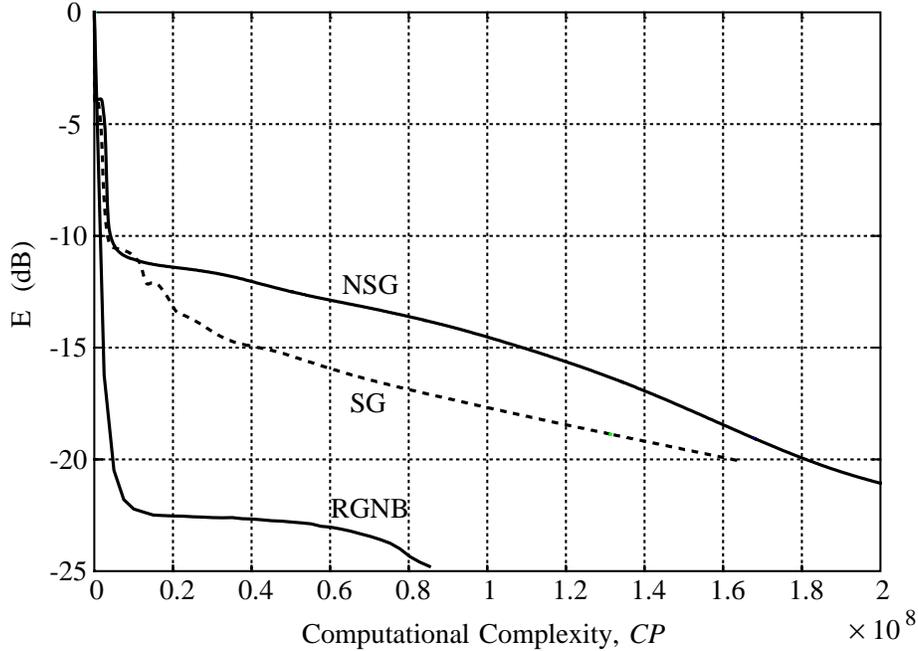


Figure 8.2: Comparison of the NSG, the SG, and the RGNB algorithms. The relative error index, E , is plotted versus the computational complexity, CP , calculated according to Table 8.3.

adjusted. Experiments have shown that practicable settings are²:

$$10^{-3} \leq \delta \leq 10^{-2} \quad (8.2)$$

$$\lambda(\ell) = 1 - \eta \cdot a^\ell, \quad 0.05 \leq \eta \leq 0.1, \quad 0.99 \leq a \leq 0.999. \quad (8.3)$$

- Clearly Fig. 8.2 confirms the result (reported elsewhere in the literature) that second order algorithms converge much faster than first order algorithms³. In order to reach $E = -20\text{dB}$ the RGNB-algorithm requires $CP \approx 5 \cdot 10^6$, whereas the first order algorithms require $CP \approx 1.7 \cdot 10^8$. Using the Intel 486 microprocessor as the hardware platform a floating point multiplication requires around 15 clock cycles. If we consider an overhead of 100% (move operations, etc.) then employing a 33MHz

²Recall that $0.05 \leq \eta \leq 0.1$ corresponds to initial values: $0.9 \leq \lambda(0) \leq 0.95$. Furthermore, $0.99 \leq a \leq 0.999$ is equivalent to raising $\lambda(\ell)$ from $\lambda(0)$ to 1 with a time constant which approximately lies in the interval $[100; 1000]$. The quoted δ -interval ensures that the algorithm operates well; however, if data are sparse so that regularization is urgent then probably larger values are required.

³In some comparisons reported in the literature the axis of abscissas represents the number of iterations rather than the computational complexity. However, this yields an unfair comparison since the computational complexity (per iteration) scales as $o(m)$ for first order algorithms and as $o(m^2)$ for second order algorithms, where m is the number of weights.

processor the training with first order algorithms will take about 3 min. in contrast to only 5 sec. using the second order algorithm⁴.

- The SG-algorithm seems in this example to perform better than the NSG-algorithm; however, a further fine-tuning of the step-sizes may lead to the opposite conclusion. In addition, the adjusting of the normalized step-size, α , is in general less problematic in comparison with the adjustment of the step-size, μ .

In addition, the simulations indicated that even without intensive parameter fine-tuning, the RGNB-algorithm generally has a faster convergence.

Moreover, we tested the capabilities of the off-line modified Gauss-Newton (MGN) algorithm cf. Sec. 5.5 on the training set mentioned above. By using random weight initialization simulations showed that the algorithm frequently is trapped in poor local minima. On the other hand, employing the weight initialization algorithm cf. Sec. 5.3.2 slightly ameliorates the performance. Fig. 8.3 shows E as a function of the number of iterations using the parameters: $\mu_{\max} = 1$, $\kappa = 0$. Obviously, the algorithm is quickly trapped in

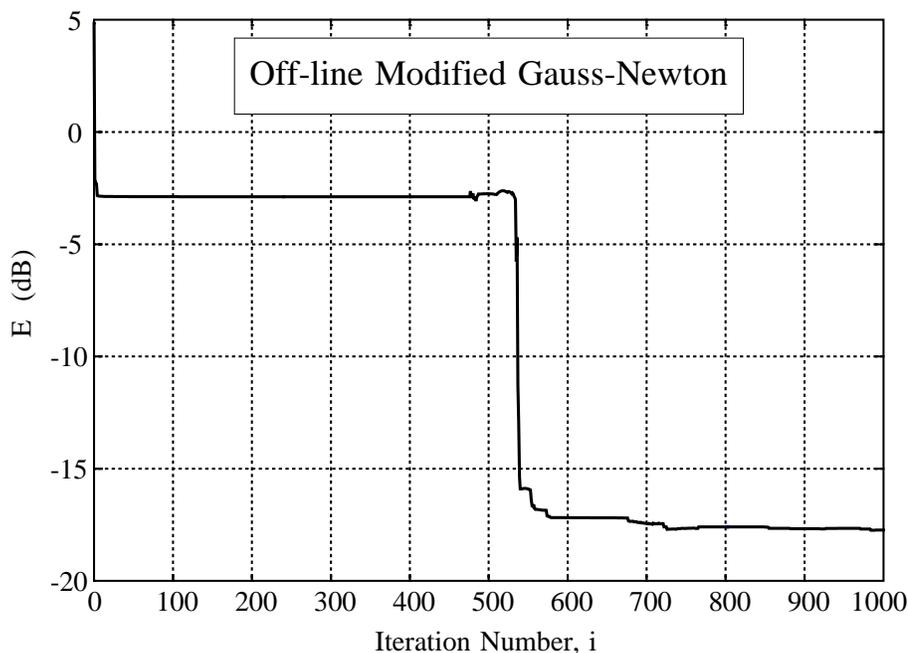


Figure 8.3: The relative error index, E , plotted versus the number of iterations when using the MGN-algorithm.

a local minimum with poor performance; however, at $i \approx 540$ the algorithm escapes the

⁴This calculation is of course connected with some uncertainty since – for instance – optimal coding is assumed indirectly.

local minimum and converges afterwards very rapidly to a (local) minimum with much better performance. The rapid convergence is a consequence of the second order nature of the algorithm. However, rapid convergence is of minor importance if the algorithm is likely to get stuck in local minima. In fact, the final performance of the MGN-algorithm is obtained in very few iterations of the RGNB-algorithm. Consequently, we conclude that a direct implementation of the MGN-algorithm is not recommendable. The algorithm may be improved by:

1. Adding stochastic noise to the weight update direction thus enabling the escape from local minima.
2. Using a very large $\mu_{\max} \gg 1$ ⁵ helps in escaping from local minima by tunneling through local maxima (see e.g., Fig. 5.7). This, however, significantly increases the computational complexity.

8.3 Testing Neural Networks for Signal Processing Tasks

In this section we test the capabilities of 2-layer feed-forward perceptron neural networks cf. Sec. 3.2.2 by simulating various synthetic signal processing tasks which comprise: System identification, inverse modeling, and time-series prediction according to the definitions in Ch. 1. The purpose is to substantiate the usefulness of the proposed algorithms from a purely methodical point of view, even though the ultimate objective – of course – is the solution of real world tasks. The synthetic systems are selected in order to meet the following claims:

- The structure of the system should not be tailored to the structure of the neural network. This reflects the fact that we consider “black-box” modeling. It is important to emphasize that if the detailed structure of the particular system is known in advance the knowledge should be incorporated into the choice of filter architecture. Consequently, the best choice of architecture, for modeling the considered systems, is obviously not a neural network; however, the neural network may be a reasonable candidate without any a priori assumptions.
- The chosen systems should preferably possess a physical interpretation so that realistic nonlinearities are dealt with.

8.3.1 System Identification

In the system identification case the task is to predict the output, $y_s(k)$, of a system from the input, $x_s(k)$. The considered system is Morison’s equation [Bendat 90, Sec. 7.4] which – in continuous time – is given by :

$$y_s(t) = c_1 \cdot \frac{dx_s(t)}{dt} + c_2 \cdot x_s(t) \operatorname{sgn}(x_s(t)) \quad (8.4)$$

where t is the continuous time and c_1, c_2 are some constants. The first term is an inertial force while the second defines a nonlinear friction. In this simulation we used: $c_1 = 0.2$, $c_2 = 0.8$. The physical interpretation of the systems is illustrated in Fig. 8.4⁶ $y_s(t)$ is the

⁵Recall that μ_{\max} is the initial step-size. The MGN-algorithm thus – in each iteration – reduces the step-size by factors of 2 until decrease in the cost function is noticed.

⁶The illustration is adapted from [Bendat 90, Fig. 7.7].

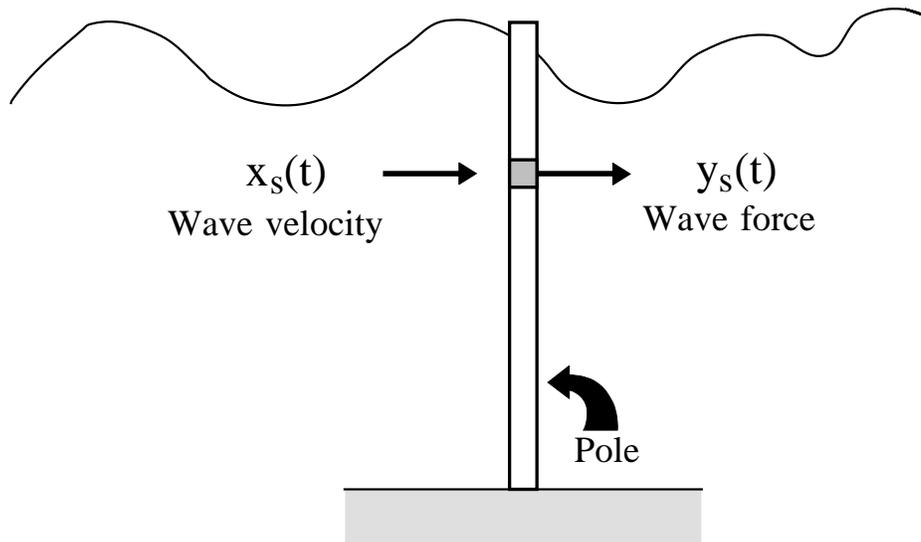


Figure 8.4: Illustration of the wave force problem.

force on a pole (in a certain depth) due to the wave velocity $x_s(t)$ which is assumed to be stationary and Gaussian. Note that since the system is non-recursive it may be identified by using a non-recursive model.

A discrete time approximation of the continuous time system is obtained by replacing the continuous derivative by a discrete time derivative approximation. A linear phase derivative filter may be designed using the Parks-McClelland equiripple FIR filter design procedure [Oppenheim & Schaffer 89, Sec. 7.6–7] which is discussed in Ch. 4. Fig. 8.5 shows the magnitude, $|H_d(f)|$, of the derivative filter when employing a maximum relative deviation⁷, $\alpha = 0.5\%$, within the normalized frequency⁸ range, $f \in [0; 1/4]$, and a filter order, $L = 9$. A discrete approximation of the system is with sampling period, $T = 1$, given by:

$$y_s(k) = c_1 \cdot (h_d(k) * x_s(k)) + c_2 \cdot x_s \left(k - \frac{L-1}{2} \right) \operatorname{sgn} \left(x_s \left(k - \frac{L-1}{2} \right) \right) \quad (8.5)$$

where $h_d(k)$ is the impulse response of the derivative filter and $*$ denotes convolution. The delay associated with the derivative filter is $(L-1)/2$, cf. Ch. 4; consequently, the term proportional to c_2 is delayed $(L-1)/2$ time steps in order to compensate the derivative filter delay. The discrete system is shown in Fig. 8.6. The input signal $x_s(k)$ is band-pass filtered white Gaussian noise with zero mean and variance $\sigma_{x_s}^2 = 0.5$. The cut-off

⁷The relative deviation is w.r.t. to the ideal frequency response of the derivative filter, i.e., $H(f) = j2f$. See further the definition given in `na:reldev`.

⁸That this, the frequency normalized w.r.t. the sampling frequency.

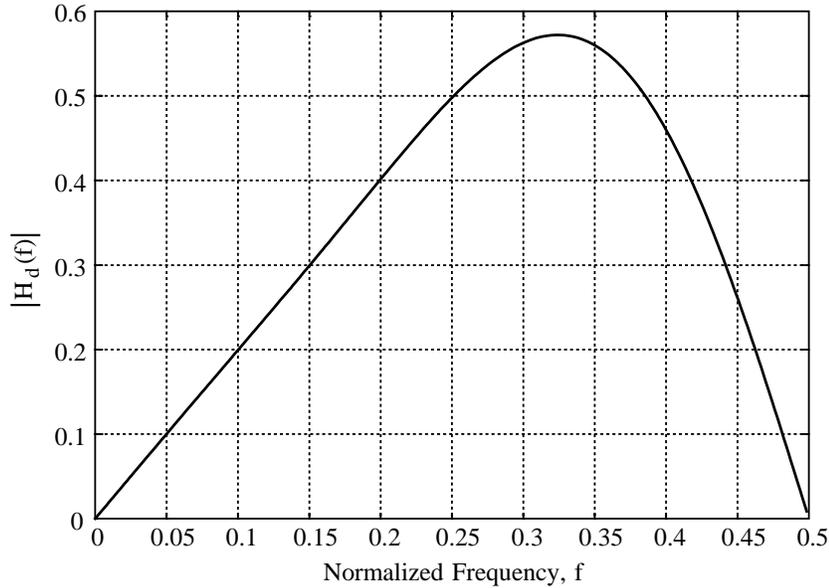


Figure 8.5: The magnitude of the discrete derivative FIR filter with maximum relative deviation, $\alpha = 0.5\%$, within the normalized frequency range, $f \in [0; 1/4]$ and filter order, $L = 9$.

frequencies are chosen as 0.05 and 0.15 which ensures that the derivative filter closely approximates the continuous derivative. Fig. 8.7 shows the input and output signals and the respective power spectra⁹.

8.3.2 Inverse Modeling

Consider a system with input, $x_s(k)$, and output, $y_s(k)$. The inverse modeling task is then to reconstruct the system input from the system output. That is, the input of the model is $x(k) = y_s(k)$, while the output of the model is $y(k) = x_s(k - D)$ where $D \geq 0$ is a suitable delay compensating the delay in the system, cf. Ch. 1, Ch. 4. Since we merely deal with non-recursive models, it is natural to consider purely recursive systems only (see also Ch. 1).

8.3.2.1 Difference Equation Approximation of Differential Equation Systems

Consider a class of systems given by the nonlinear differential equation:

$$a_n(y_s(t)) \cdot \frac{d^n y_s(t)}{dt^n} + a_{n-1}(y_s(t)) \cdot \frac{d^{n-1} y_s(t)}{dt^{n-1}} + \dots + a_1(y_s(t)) \cdot \frac{dy_s(t)}{dt} + a_0(y_s(t)) = b x_s(t) \quad (8.6)$$

⁹The power spectra are obtained by using the method of Welch [Oppenheim & Schaffer 89, Sec. 11.6]. Here we used a 1024 point Hann-weighted FFT and 50% overlap among neighbor windows.

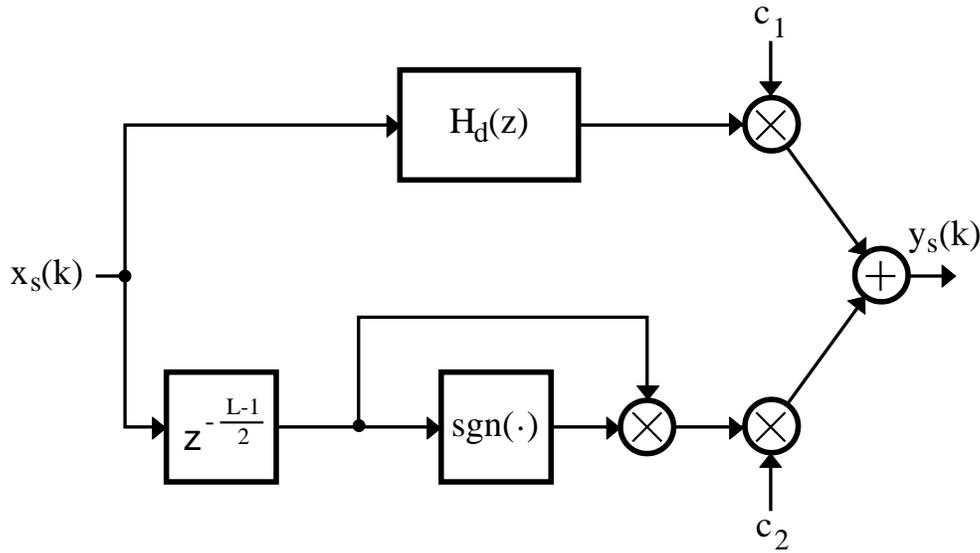


Figure 8.6: Discrete time implementation of Morison's equation.

where $x_s(t)$, $y_s(t)$ are the system input and output, respectively, $a_i(\cdot)$ are prescribed functions, and b is a constant. A difference equation approximation of the differential equation is obtained by sampling and approximating the continuous derivatives by discrete derivatives. Let T be the sampling period, i.e., $t = kT$ and per definition, $\forall \tau: x(t + \tau) \triangleq x(k + \tau/T)$. Thus the differential equation transforms into:

$$a_n(y_s(k)) \cdot D^n [y_s(k)] + a_{n-1}(y_s(k)) \cdot D^{n-1} [y_s(k)] + \dots + a_1(y_s(k)) \cdot D [y_s(k)] + a_0(y_s(k)) = b x_s(k) \quad (8.7)$$

where

$$\frac{d^i y_s(t)}{dt^i} \triangleq D^i [y_s(k)], \quad \forall i, \quad (8.8)$$

and $D^i[\cdot]$ denotes the discrete i 'th derivative operator.

Consider the Taylor series expansion:

$$y_s(t + \tau) - y_s(t) = \frac{dy_s(t)}{dt} \tau + \frac{d^2 y_s(t)}{dt^2} \frac{\tau^2}{2} + \dots + \frac{d^n y_s(t)}{dt^n} \frac{\tau^n}{n!} + o(\tau^n). \quad (8.9)$$

Next choose a set of τ -values:

$$\tau \in \{T, -k_1 T, -k_2 T, \dots, -k_{n-1} T\} \quad (8.10)$$

where $0 < k_1 < k_2 < \dots < k_{n-1}$ are prescribed integers. Writing out the series, for the above

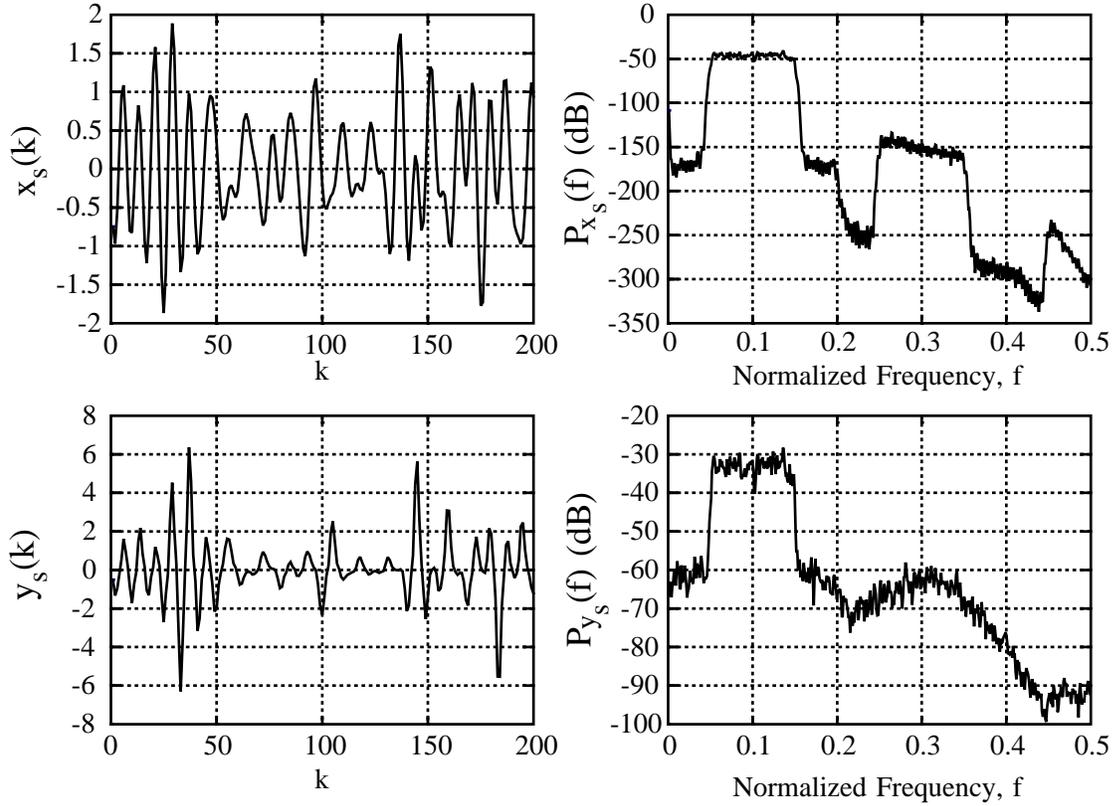


Figure 8.7: Simulation of Morison's equation. $x_s(k)$, $y_s(k)$, are the input and output signals, respectively, and $P_{x_s}(f)$, $P_{y_s}(f)$ are the associated power spectra. The parameters associated with Morison's equation were: $c_1 = 0.2$, $c_2 = 0.8$.

τ -values, yields the following system of linear equations:

$$\begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n!} \\ -k_1 & \frac{k_1^2}{2} & \cdots & \frac{(-k_1)^n}{n!} \\ \vdots & \vdots & \ddots & \vdots \\ -k_{n-1} & \frac{k_{n-1}^2}{2} & \cdots & \frac{(-k_{n-1})^n}{n!} \end{bmatrix} \cdot \begin{bmatrix} TD[y_s(k)] \\ T^2 D^2[y_s(k)] \\ \vdots \\ T^n D^n[y_s(k)] \end{bmatrix} = \begin{bmatrix} x(k+1) - x(k) \\ x(k-k_1) - x(k) \\ \vdots \\ x(k-k_{n-1}) - x(k) \end{bmatrix}. \quad (8.11)$$

Using Cramer's theorem from linear algebra it is possible to solve simultaneously for the relevant derivative approximations. The solution has the general structure:

$$\begin{aligned} D^i[y_s(k)] &= T^{-i} \boldsymbol{\gamma}_i^\top [y_s(k+1), y_s(k), y_s(k-k_1), \dots, y_s(k-k_{n-1})]^\top \\ &= \mathbf{h}_{di}^\top \mathbf{y}_{s,L}(k+1) \end{aligned} \quad (8.12)$$

where

- $\boldsymbol{\gamma}_i = [\gamma_{i,1}, \gamma_{i,2}, \dots, \gamma_{i,n+1}]^\top$ is a $(n+1)$ -dimensional vector.
- \mathbf{h}_{d^i} is the $L = k_{n-1} + 2$ dimensional tap coefficient vector of an equivalent FIR filter given by:

$$\mathbf{h}_{d^i} = T^{-i} \cdot [\gamma_{i,1}, \gamma_{i,2}, \underbrace{0, \dots, 0}_{k_1-1}, \gamma_{i,3}, \underbrace{0, \dots, 0}_{k_2-k_1-1}, \dots, \gamma_{i,n+1}]^\top. \quad (8.13)$$

- $\mathbf{y}_{s,L}(k+1)$ is the tapped delay line

$$\mathbf{y}_{s,L}(k+1) = [y_s(k+1), y_s(k), \dots, y_s(k-k_{n-1})]^\top. \quad (8.14)$$

Substituting `si:dieqn` into `si:difreqn` we finally obtain the following recursive implementation:

$$y_s(k+1) = \varphi(y_s(k), y_s(k-k_1), \dots, y_s(k-k_{n-1}), x(k)) \quad (8.15)$$

where $\varphi(\cdot)$ is a nonlinear function which depends on $\boldsymbol{\gamma}_i$, $i = 1, 2, \dots, n$, the functions $a_i(\cdot)$, and the coefficient b . The exact expression for $\varphi(\cdot)$ is found by simple algebraic manipulations.

In order to make the above recursion stable¹⁰ simulations have shown that it is normally advantageous to construct the FIR filters so that they are of minimum-phase type, i.e., all zeros of the transfer functions should be located inside the unit circle of the complex z -plane. For this purpose we designed an algorithm which search over possible sets $\{k_1, k_2, \dots, k_{n-1}\}$, and succeeded in derivative approximations which ensured a stable implementation. Furthermore, the obtained $D^i[\cdot]$ closely approximate the continuous derivatives in the desired frequency range, as elaborated below.

8.3.2.2 The Pendulum

The physics of the pendulum is shown in Fig. 8.8 and the dynamics is described by the following second order nonlinear differential equation:

$$\frac{d^2 y_s(t)}{dt^2} + c \frac{dy_s(t)}{dt} + \left(\frac{2\pi}{P}\right)^2 \sin(y_s(t)) = b x_s(t) \quad (8.16)$$

where c is a damping parameter¹¹, P is the periodic time of the undamped linear oscillator¹², and b is a constant defining the strength of the driving force, $x_s(t)$. The inverse modeling task is then to reconstruct the driving force, $x_s(t)$, from measurements of the angle deflection, $y_s(t)$.

The first and second order derivatives are approximated by using the technique described in Sec. 8.3.2.1. In this case we employed¹³:

$$D[y_s(k)] = \mathbf{h}_d^\top \mathbf{y}_{s,4}(k+1) \quad (8.17)$$

$$D^2[y_s(k)] = \mathbf{h}_{d^2}^\top \mathbf{y}_{s,10}(k+1) \quad (8.18)$$

¹⁰Note since the recursion is *nonlinear* it is not possible to give general guidelines which ensure stability.

¹¹Here we deal with linear viscous damping; however, other types of damping may as well appear, e.g., velocity-proportional damping $y'_s(t)|y'_s(t)|$ due to air drag.

¹²That is, $\sin(y_s(t))$ is replaced by $y_s(t)$.

¹³In spite of the fact that the filter, \mathbf{h}_d , is not minimum-phase no instability occurred in the present case.

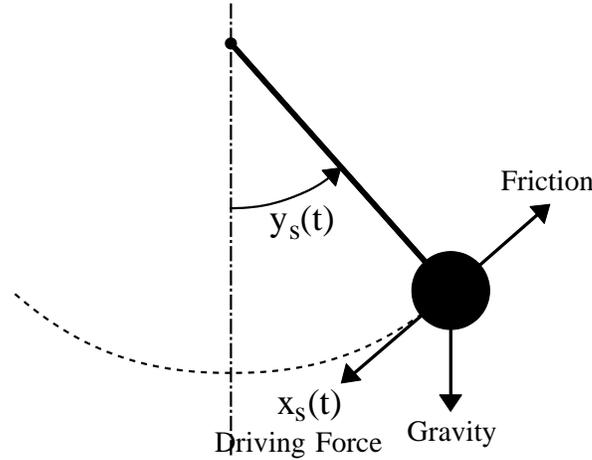


Figure 8.8: The physics of the pendulum. $x_s(t)$ is the driving force and $y_s(t)$ is the angle deflection.

where

$$\mathbf{y}_{s,L}(k+1) = [y_s(k+1), y_s(k), y_s(k-1), \dots, y_s(k-L+2)]^\top, \quad (8.19)$$

and

$$\mathbf{h}_d = T^{-1} \left[\frac{1}{3}, \frac{1}{2}, -1, \frac{1}{6} \right]^\top \quad (8.20)$$

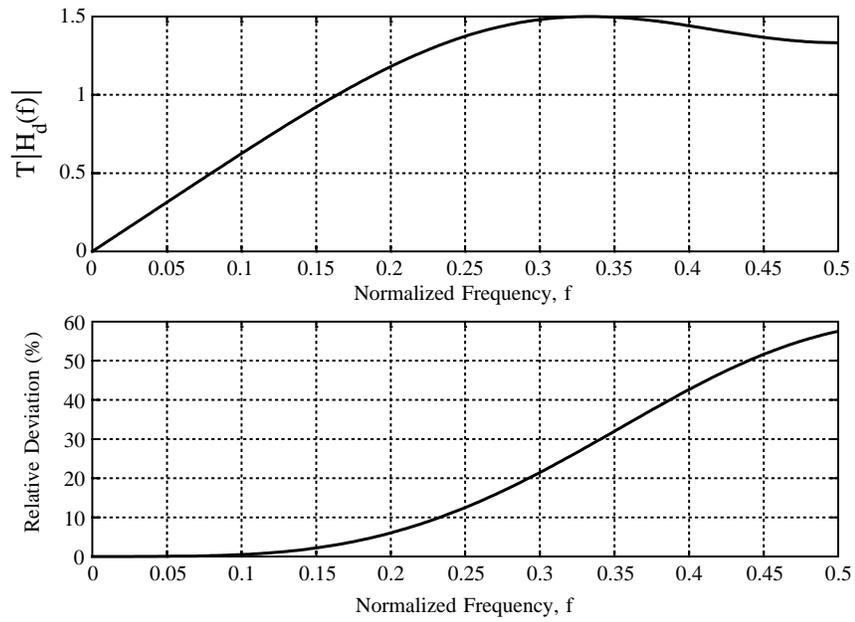
$$\mathbf{h}_{d^2} = T^{-2} [0.7667, -0.9054, -0.9683, 1.4889, 0, -0.7528, 0.4222, 0, -0.0683, 0.0169]^\top. \quad (8.21)$$

In Fig. 8.9 the magnitude of the transfer functions and the relative deviations¹⁴ are depicted. `si:pendul` is sampled with $T = 0.05$ by using the parameters $c = 0.2$, $P = 1$, and $b = 7$. The system input $x_s(k)$ is a zero mean Gaussian low-pass filtered white noise sequence with normalized cut-off frequency 0.25 and variance, $\sigma_{x_s}^2 = 1$. Fig. 8.10 shows the input and output signals and the respective power spectra¹⁵.

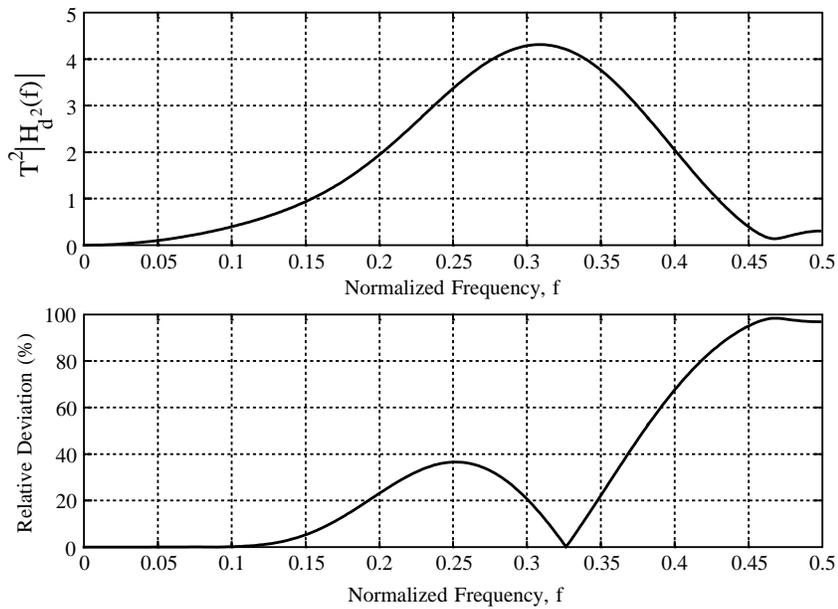
¹⁴The relative deviation is cf. `na:reldev` in this case given by:

$$\frac{|T^i |H_{d^i}(f)| - (2\pi f)^i|}{(2\pi f)^i}.$$

¹⁵We employed Welch's method using 1024 point Hann-weighted FFT's and 50% overlap among neighbor windows.



(a)



(b)

Figure 8.9: The magnitude of the transfer functions defining the first and second order derivative approximations. The curves representing the relative deviations show that the derivative filters are operable in the range $f \in [0; 0.1]$.

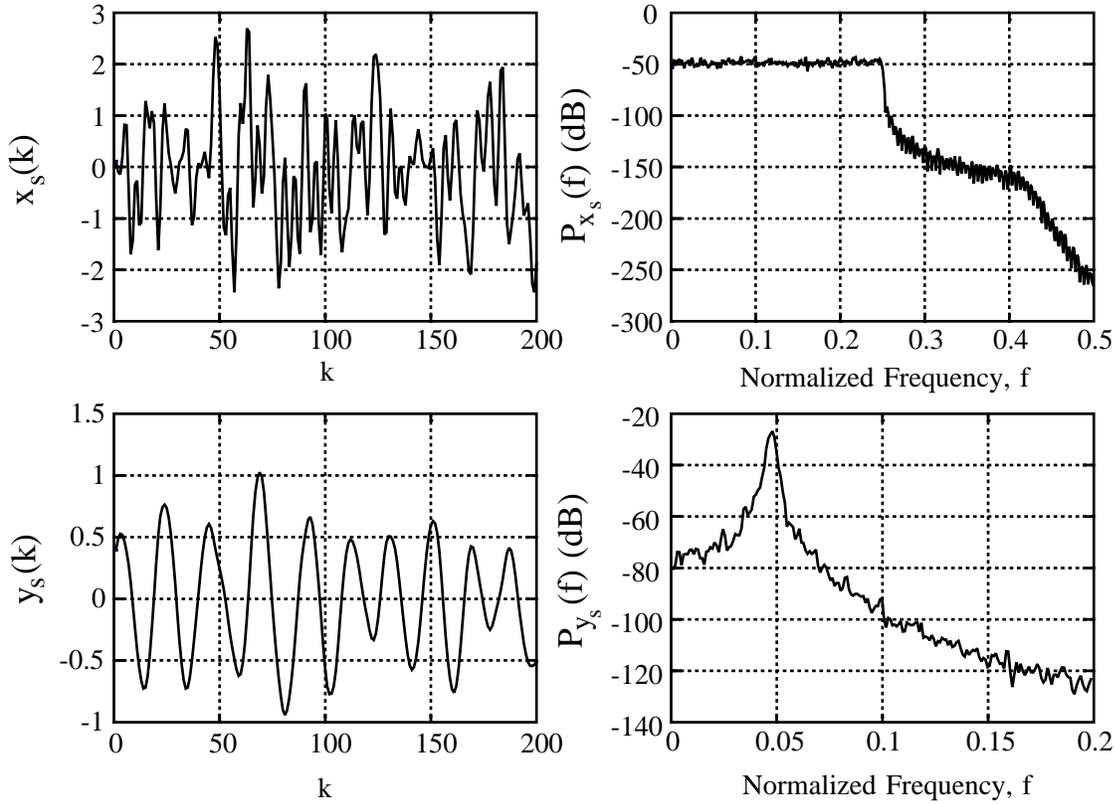


Figure 8.10: Simulation of the pendulum. $x_s(k)$, $y_s(k)$, are the input and output signals, respectively, and $P_{x_s}(f)$, $P_{y_s}(f)$ are the associated power spectra. Note that $y_s(k)$ is effectively band-limited to the operable range of the derivative filters, viz. $f \in [0; 0.1]$.

8.3.2.3 The Nonlinear Channel

A simple nonlinear channel is depicted in Fig. 8.11. The limiting $\tan^{-1}(\cdot)$ functions model saturating effects in the transmitter and the receiver, and the linear low-pass filter, $H(z)$, models spectral distortion. The filter is given by:

$$H(z) = \frac{0.1}{1 - 2.46z^{-1} + 2.49z^{-2} - 1.21z^{-3} + 0.24z^{-4}}, \quad (8.22)$$

and the magnitude is shown in Fig. 8.12. The system input, $x_s(k)$, is – as in the system identification case – a band-pass filtered zero mean Gaussian white noise with variance $\sigma_{x_s}^2 = 0.5$. The cut-off frequencies were 0.05 and 0.15. The channel parameters were in this simulation chosen as: $c_1 = c_2 = 1.2$. Fig. 8.13 shows the input and output signals and their respective power spectra¹⁶.

¹⁶We employed Welch's method using 1024 point Hann-weighted FFT's and 50% overlap among neighbor windows.

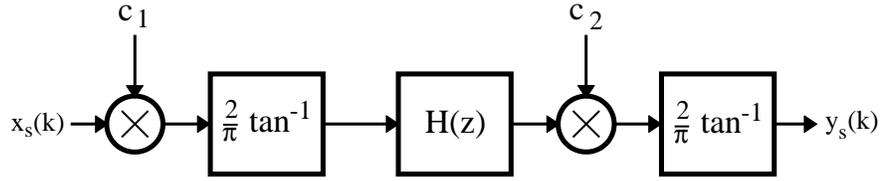


Figure 8.11: The nonlinear channel. c_1 , c_2 are channel parameters.

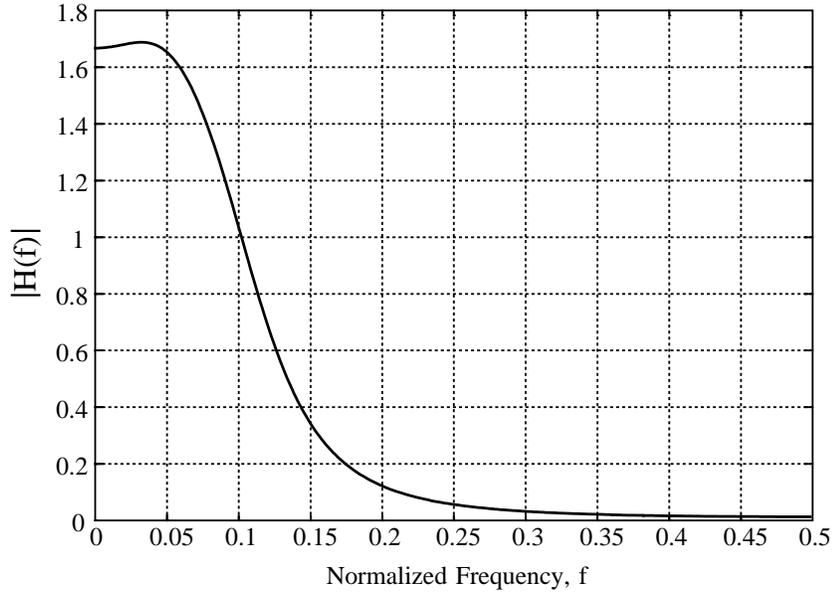


Figure 8.12: The magnitude of the low-pass filter used when modeling the nonlinear channel.

8.3.3 Time-Series Prediction

In the time-series prediction case we deal with the nonlinear Mackey-Glass differential equation which describes certain physiological control systems [Mackey & Glass 77]. The

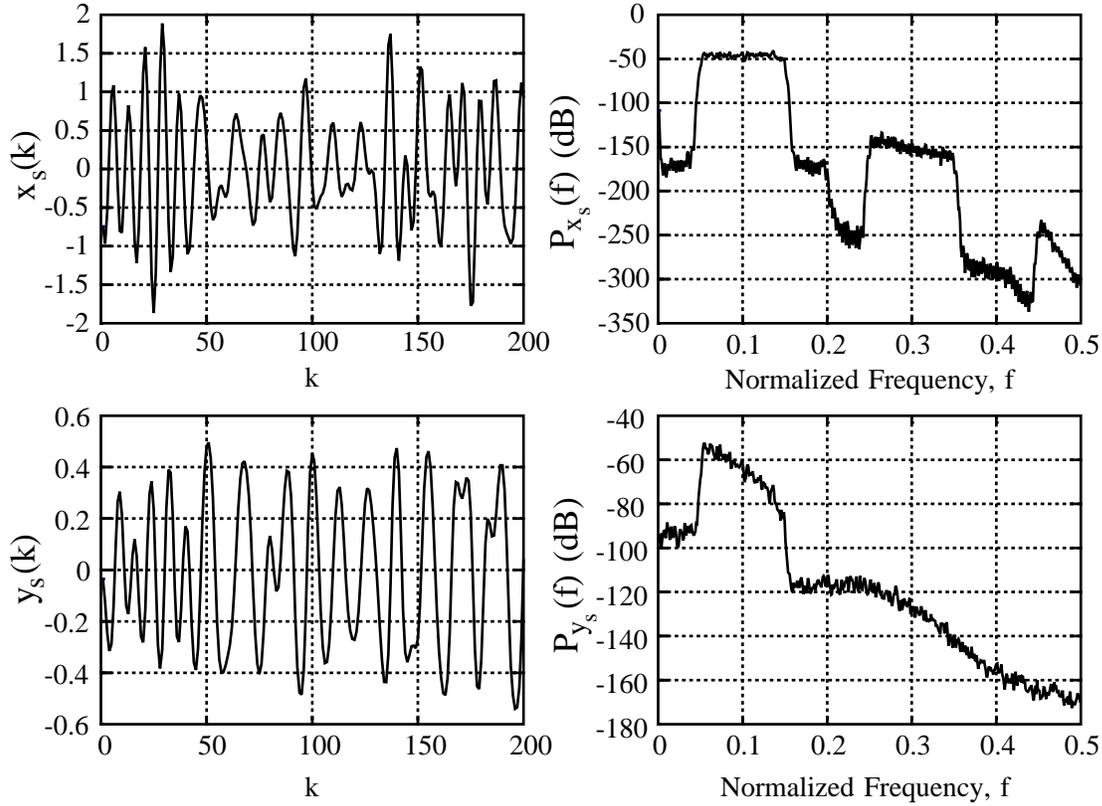


Figure 8.13: Simulation of the nonlinear channel. $x_s(k)$, $y_s(k)$, are the input and output signals, respectively, and $P_{x_s}(f)$, $P_{y_s}(f)$ are the associated power spectra.

differential equation is given by:

$$\frac{dy_s(t)}{dt} = by_s(t) + \frac{ay_s(t - \tau)}{1 + y_s^{10}(t - \tau)} \quad (8.23)$$

where a , b , τ are constants. The technique mentioned in Sec. 8.3.2.1 is used to approximate the continuous derivative. In this simulation we employed:

$$D[y_s(k)] = \mathbf{h}_d^\top \mathbf{y}_{s,11}(k + 1) \quad (8.24)$$

where

$$\mathbf{h}_d = T^{-1} [0.4375, -0.0460, 0, -0.5, 0, 0, 0.1750, 0, -0.0804, 0, 0.0139]^\top. \quad (8.25)$$

In Fig. 8.14 the magnitude of the transfer function and the relative deviation are depicted. Employing $a = 0.2$, $b = -0.1$, and $\tau = 17$, the solution of `si:mkeqn` becomes chaotic, i.e., large sensitivity on initial conditions. That means, even if two solutions are adjacent at

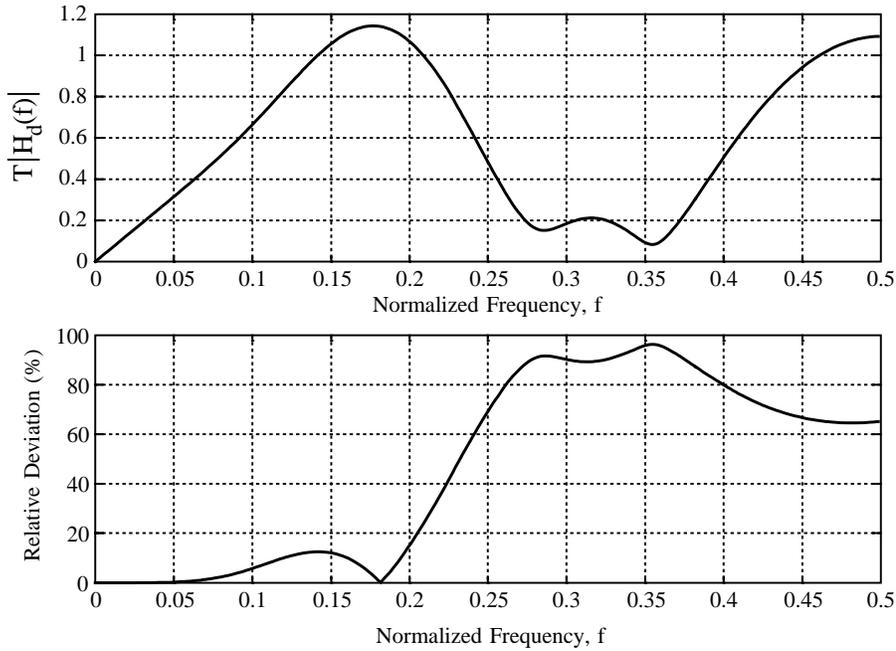


Figure 8.14: The magnitude of the transfer function defining a first order derivative approximation. The curves representing the relative deviations show that the derivate filter is operable in the range $f \in [0; 0.05]$.

a certain time instant they will diverge from each other at an exponential rate¹⁷ as time goes by. The chaotic motion may be interpreted as a complex eigen-oscillation of the systems; hence, the system possesses infinite memory. A consequence is that the errors done when discretizing the differential equation may accumulate so that the discrete systems behave differently; however, choosing the sampling period sufficiently small we may be able to reconstruct the geometric structure of the chaotic attractor. In particular, Takens [Farmer & Sidorowich 88], [Takens 81] showed that the state space may be embedded by using a tapped delay line: $[y_s(t), y_s(t - \delta), \dots, y_s(t - (d - 1)\delta)]$, where d is the so-called embedding dimension and δ is a suitable delay time. d is related to the fractal dimension of the chaotic attractor according to $d = 2r + 1$ where r is the attractor dimension¹⁸.

By employing $T = 0.1$ with the derivative approximation given by `si:hd1` we generated a time-series of which the structure seemed robust towards a further reduction of T . The obtained time-series, say $y_s(n)$, is tremendously over-sampled as the power spectrum is approximately band-limited to the normalized frequency range $[0; 0.01]$. In agreement with

¹⁷This divergence is characterized by the so-called Liapunov exponents, see e.g., [Farmer & Sidorowich 88], [Thomson & Stewart 86, Sec. 17.3].

¹⁸ $r \approx 2.1$, cf. e.g., [Lapedes & Farber 87] for the parameter setting: $a = 0.2$, $b = -0.1$, and $\tau = 17$.

other simulation studies, e.g., [Farmer & Sidorowich 88], [Lapedes & Farber 87], [Moody & Darken 89] the time-series is interpolated (resampled) by a factor of 10, i.e., $y_s(k) = y_s(10n)$ where the time index k corresponds to the sampling period $T = 1$. In Fig. 8.15 the time-series, $y_s(k)$, and the corresponding power spectrum, $P_{y_s}(f)$ ¹⁹ are shown.

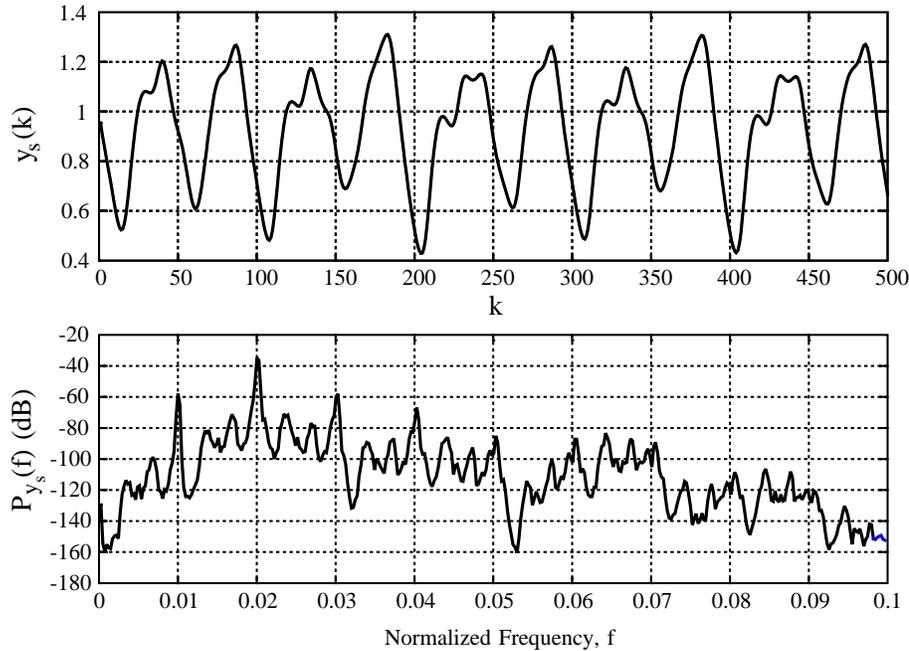


Figure 8.15: Simulation of the Mackey-Glass equation. $y_s(k)$ is the generated chaotic, quasi-periodic time-series, and $P_{y_s}(f)$ is the corresponding power spectrum. Note that the power spectrum is dominated by a tone at $f = 0.02$ (periodic time 50).

8.3.4 Modeling the Simulated Systems

The systems described above were used for system identification, inverse modeling and time-series prediction which are shown in Fig. 8.16 – 8.18. The architecture of the employed model is – in accordance with the generic architecture Ch. 4, Fig. 4.1 – depicted in Fig. 8.19. The preprocessing unit transforms the input signal, $x(k)$, into a vector signal $\mathbf{z}(k) = [z_1(k), z_2(k), \dots, z_p(k)]^\top$ by filtering with p different linear filters specified by the impulse responses $h_i(k)$, $k = 1, 2, \dots, L$, $i = 1, 2, \dots, p$. The associated transfer functions are denoted by, $H_i(z)$. The processing of the preprocessing unit is expressed by:

$$\mathbf{z}(k) = \mathbf{C}\mathbf{x}(k) \quad (8.26)$$

¹⁹We employed Welch’s method using 4096 point Hann-weighted FFT’s and 50% overlap among neighbor windows.

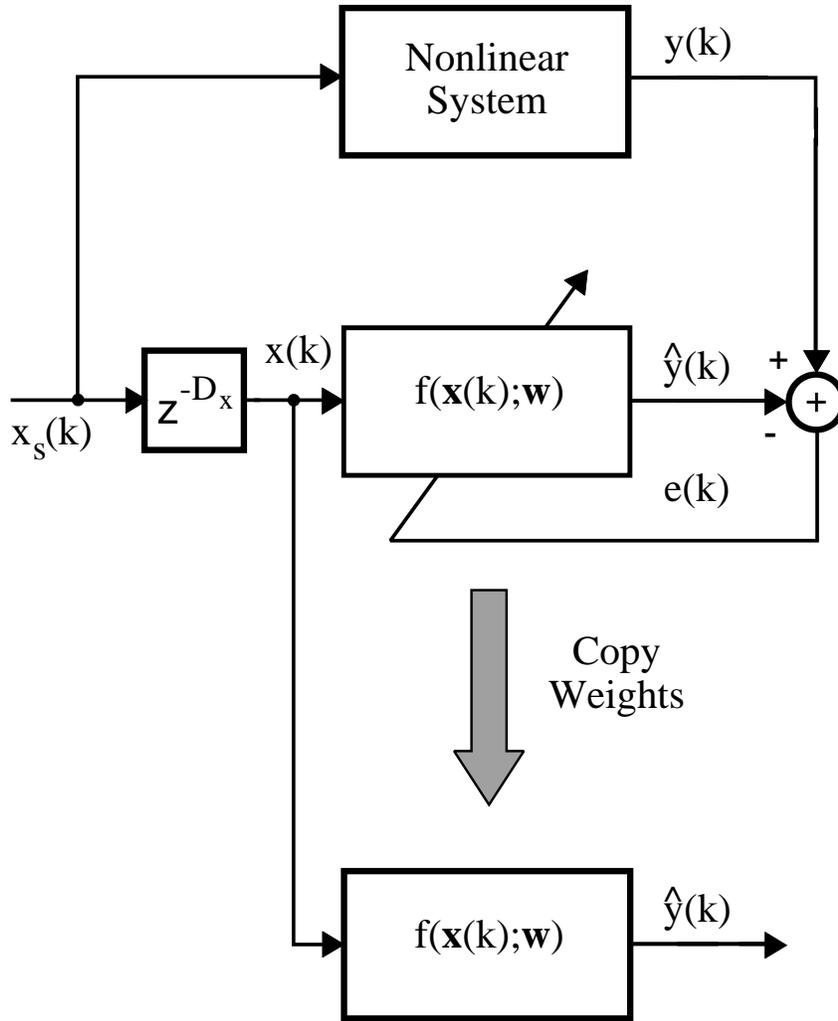


Figure 8.16: System Identification.

where $\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-1+L)]^\top$ is the L -dimensional input vector signal and \mathbf{C} is the $p \times L$ dimensional preprocessing matrix:

$$\mathbf{C} = \begin{bmatrix} h_1(1) & h_1(2) & \cdots & h_1(L) \\ h_2(1) & h_2(2) & \cdots & h_2(L) \\ \vdots & \vdots & \ddots & \vdots \\ h_p(1) & h_p(2) & \cdots & h_p(L) \end{bmatrix}. \quad (8.27)$$

In this simulation we consider two preprocessing methods (cf. Ch. 4): The principal component analysis (PCA) and the derivative preprocessor (DPP). \mathbf{C} is determined by running the *Preprocessing Algorithm* (PPA) cf. p. 90. Furthermore, this algorithm provides estimates of the number of components, p , the memory length, L , and the phase parameter,

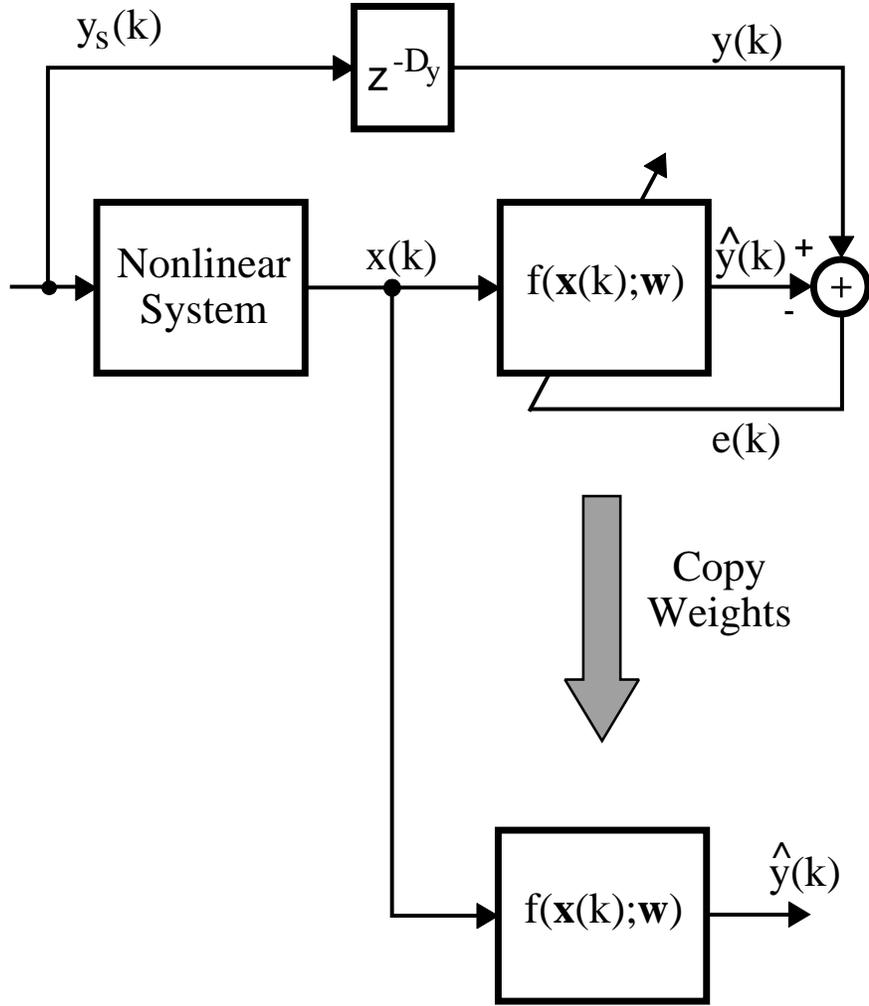


Figure 8.17: Inverse Modeling.

D . The delays D_x , D_y are subsequently determined by:

$$D_x = \begin{cases} -D & D \leq 0 \\ 0 & \text{otherwise} \end{cases}, \quad D_d = \begin{cases} D & D \geq 0 \\ 0 & \text{otherwise} \end{cases}. \quad (8.28)$$

L and D is found by searching over sets, \mathcal{L} , \mathcal{P} , of possible candidate values. The sets are selected according to preliminary estimates, L_{np} , D_{np} , originating from a nonparametric linear impulse response estimate of the relation among the input and output signals. In the present simulation we used: $\mathcal{L} = \{13, 17, 21, 25, 29, 33, 37\}$. The phase parameter set was $\mathcal{P} = \{0, 1, \dots, -10\}$ w.r.t. the system identification case, and $\mathcal{P} = \{0, 1, \dots, 20\}$ w.r.t. inverse modeling. In the time-series prediction case the phase parameter is determined by the time we want to predict into the future. We employed $D_x = -D = 100$ which is beyond the characteristic period time 50 cf. Fig. 8.15. In connection with PCA p is

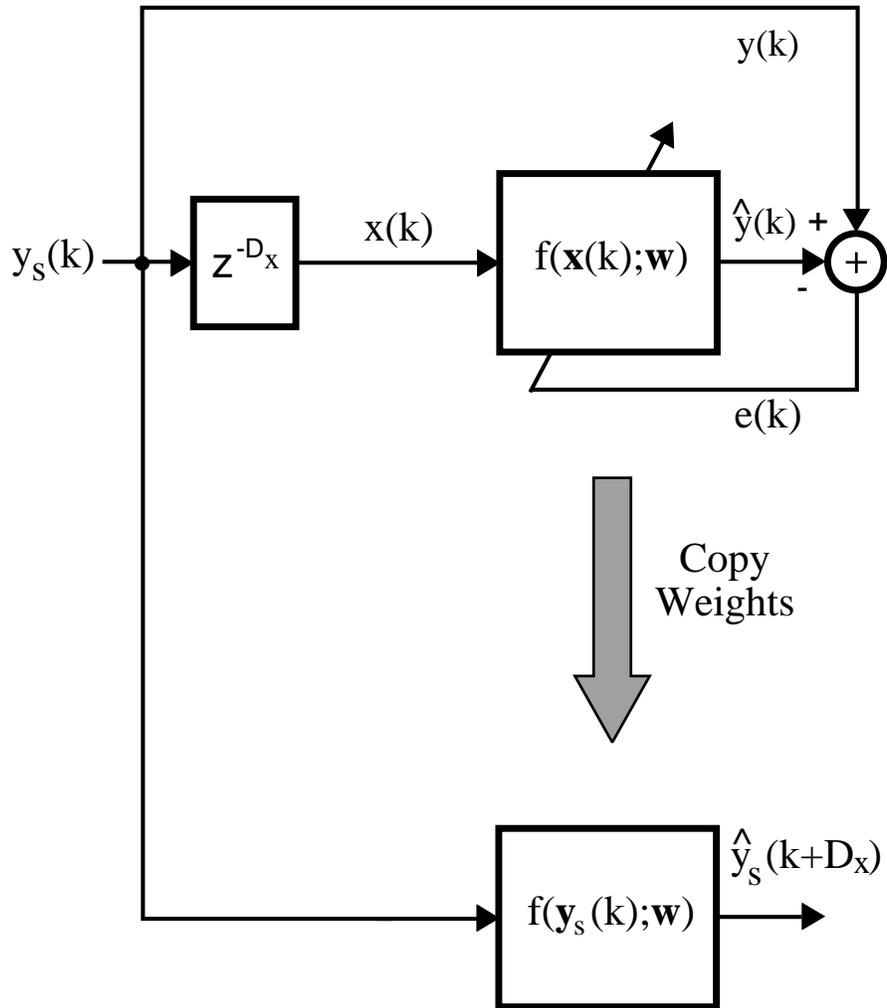


Figure 8.18: Time-Series Prediction.

determined so that 97.5% of the variance in $x(k)$ is explained by $z(k)$. When dealing with the DPP we used $p = 3$.

For each system we generated a training set with $N = 8850$ samples, and a independent cross-validation set (cf. Sec. 6.5.3) containing $N_c = 8850$ samples. However, when dealing with the pendulum we used: $N = N_c = 6970$. The generalization error is estimated by

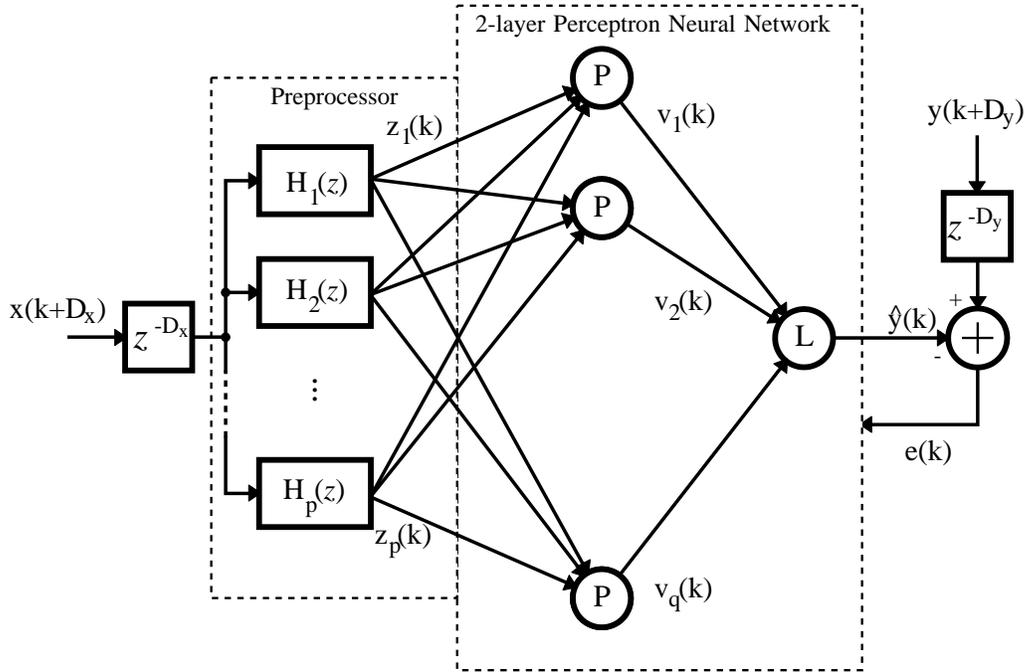


Figure 8.19: The architecture of the employed model, $f(\mathbf{x}(k); \mathbf{w})$, in accordance with the generic architecture Fig. 4.1. The preprocessor is specified by the filters $H_i(z)$ while the nonlinear filter is a 2-layer $[p, q, 1]$ perceptron neural network.

using the *relative cross-validation error index*²⁰:

$$E_c = \sqrt{\frac{\frac{1}{N_c - L + 1} \sum_{k=L}^{N_c} e^2(k)}{\frac{1}{N_c - L} \sum_{k=L}^{N_c} [y(k) - \langle y(k) \rangle]^2}} \quad (8.29)$$

where $e(k) = y(k) - \hat{y}(k)$ is the error signal on the cross-validation set, and $\langle y(k) \rangle$ is the time-average of $y(k)$. The nominator equals the usual cross-validation estimate cf. Sec. 6.5.3 – and E_c may be interpreted as a noise-to-signal ratio. Note that $E_c = 1$ is obtained by using the time-average $\langle y(k) \rangle$ as an estimate of $y(k)$, for all k , which corresponds to the most simple conceivable model. On the other hand, $E_c = 0$, appears when the filter perfectly models the task under consideration²¹.

Recall that the PPA estimates the memory length and the phase parameter by minimizing the E_c over the sets \mathcal{L} and \mathcal{P} , where E_c is calculated from estimating the weights

²⁰Note that the summation starts at $k = L$ in order to avoid the effect of transients stemming from the preprocessing filters, $H_i(z)$.

²¹Even though the systems considered in this simulation are noiseless $E_c = 0$ is not attainable since the structure of the models (intentionally) do not reflect the structure of the systems.

of an LL-model, i.e.,

$$\hat{y}(k) = [1, \mathbf{z}(k)^\top] \cdot \mathbf{w} \quad (8.30)$$

where \mathbf{w} is an $m = p + 1$ dimensional weight vector. Table 8.5 summarize the results of running the PPA.

System	Preprocessor	L	D	p	m	E_c
Morison	PCA	13	-5	4	5	0.3839
Morison	DPP	13	-2	3	4	0.3819
Pendulum	PCA	13	3	5	6	0.6386
Pendulum	DPP	17	8	3	4	0.3854
Nonlinear Channel	PCA	13	0	5	6	0.2004
Nonlinear Channel	DPP	21	7	3	4	0.2216
Mackey-Glass	PCA	13	-100	4	5	0.5426
Mackey-Glass	DPP	13	-100	3	4	0.6174

Table 8.5: Results of the Preprocessing Algorithm. L is the memory length, D is the phase parameter, p is the dimension of the vector $\mathbf{z}(k)$, $m = p + 1$ is the total number of weights in the employed LL-model (cf. `si:llmod`), and E_c is the relative cross-validation error index.

The employed filter architecture (see Fig. 8.19) is a 2-layer $[p, q, 1]$ perceptron neural network described in Sec. 3.2.2. The weights are estimated using the Recursive Gauss-Newton Algorithm with Bierman Factorization (RGNB) cf. Sec. 5.7. In Table 8.6 characteristic algorithm parameters are listed²². The results obtained with the neural network

Parameter	Definition	Interpretation	Value
h_{\max}	Fig. 5.2	Dynamic range	0.8
δ	pa:bjdef	Transition width param.	10%
itr	p. 110	Number of iterations	2 9 (Morison)
$\lambda(\ell)$	pa:lamsch	Instan. forgetting factor	$1 - 0.05 \cdot 0.9931^\ell$ $1 - 0.05 \cdot 0.9916^\ell$ (Pendulum)
γ	pa:kappaeqn	Regularization adjustment	0.01

Table 8.6: Parameter setup for the RGNB-algorithm.

architecture are summarized in Table 8.7. In this context we make the following observations:

- By comparison with the results listed in Table 8.5 we notice that the neural network architecture outperform the LL-filter in all cases. This is due to the following facts:
 - The nonlinear of nature of the modeled systems.

²²Note that some of the parameters refer to the Weight Initialization Algorithm Sec. 5.3.2.

System	Preprocessor	L	D_x	D_y	p	q	m	E_c
Morison	PCA	13	5	0	4	15	91	0.0615
Morison	PCA	13	5	0	4	25	211	0.0876
Morison	DPP	13	2	0	3	20	101	0.0556
Morison	DPP	13	2	0	3	40	201	0.0337
Pendulum	PCA	13	0	3	5	15	106	0.5717
Pendulum	PCA	13	0	3	5	35	246	0.5649
Pendulum	DPP	17	0	8	3	20	101	0.1417
Pendulum	DPP	17	0	8	3	40	201	0.1007
Nonlinear Channel	PCA	13	0	0	5	15	106	0.0299
Nonlinear Channel	PCA	13	0	0	5	35	246	0.0247
Nonlinear Channel	DPP	21	0	7	3	40	201	0.1051
Mackey-Glass	PCA	13	100	0	4	15	91	0.4263
Mackey-Glass	PCA	13	100	0	4	35	211	0.2326
Mackey-Glass	DPP	13	100	0	3	20	101	0.4289
Mackey-Glass	DPP	13	100	0	3	40	201	0.3918

Table 8.7: Results obtained by using the architecture shown in Fig. 8.19. L is the memory length, D_x , D_y are the delays of the input and output, respectively, p , q is the number of input and hidden neurons, respectively, $m = q(p + 2) + 1$ is the total number of weights, and finally, E_c is the relative cross-validation error index.

- The total number of weights, m , constitutes a reasonable fraction of the number of training data, N . This enables the network to generalize properly, i.e., keeping the weight fluctuation penalty term low (according to Sec. 6.3.4).

The improvement factor – defined as E_c of the LL-filter in proportion to E_c of the neural network filter – lies in the range 1.13 – 11.33. The actual value depends on various factors, for instance, the degree of nonlinearity in the concerned system.

- By increasing the number of hidden neurons the relative cross-validation error index generally decreases²³. Since the structure of the considered systems are not captured by the neural network models incomplete modeling is in question. Now, in general, the universal approximation theorem cf. Sec. 3.2.2 states that perfect modeling is merely achieved in the limit of an infinite number of hidden neurons. However, since a (fully connected) architecture with, say q_0 , hidden neurons is embedded in architectures with $q > q_0$ then the approximation capability²⁴ of the larger networks is at least as good – possibly better. This matter is retrieved in E_c since the number of training data is kept relatively large.
- Evaluating the usefulness of the proposed preprocessing methods we notice that the DPP works the best when dealing with Morison’s equation and the pendulum. According to si:mordis the identification of Morison’s equation is described by the

²³Note that in the Morison PCA case E_c is actually larger when using $m = 211$ than when using $m = 91$. This is probably due to the fact that the weights are trapped in a local minimum of the cost function.

²⁴That is, the mean square model error (cf. Sec. 6.3.4) decreases as q increases.

model:

$$y(k) = \varphi(x(k), D[x(k)]) \quad (8.31)$$

where $\varphi(\cdot)$ is a nonlinear mapping. Hence, a DPP with two components (i.e., $p = 2$) is an evident preprocessor²⁵. Likewise, the inverse model of the pendulum is expressed by:

$$y(k) = \varphi(x(k), D[x(k)], D^2[x(k)]). \quad (8.32)$$

This means, a DPP with 3 components is tailored to this problem.

The E_c associated with modeling Morison's equation using PCA is approximately twice as large than that obtained by using DPP. In the pendulum the difference in performance is even more noticeable (a factor of approx. 5.5) which may be explained by considering the properties of PCA. The power spectrum of the input signal, $x(k)$ is a narrow and peaks at $f \approx 0.05$ while the output signal, $y(k)$, is a band-limited ($f \in [0; 0.25]$) white noise signal²⁶. Consequently, the filter should display high gain at frequencies outside the narrow-band region. However, since the signals, $z_i(k)$, produced by the PCA attempt to explain the variance in $\mathbf{x}(k)$ the power spectra $P_i(f)$ of $z_i(k)$ will resemble the power spectrum of the input signal – especially as regards the essential components. This is depicted in Fig. 8.20(a,b). Furthermore, the

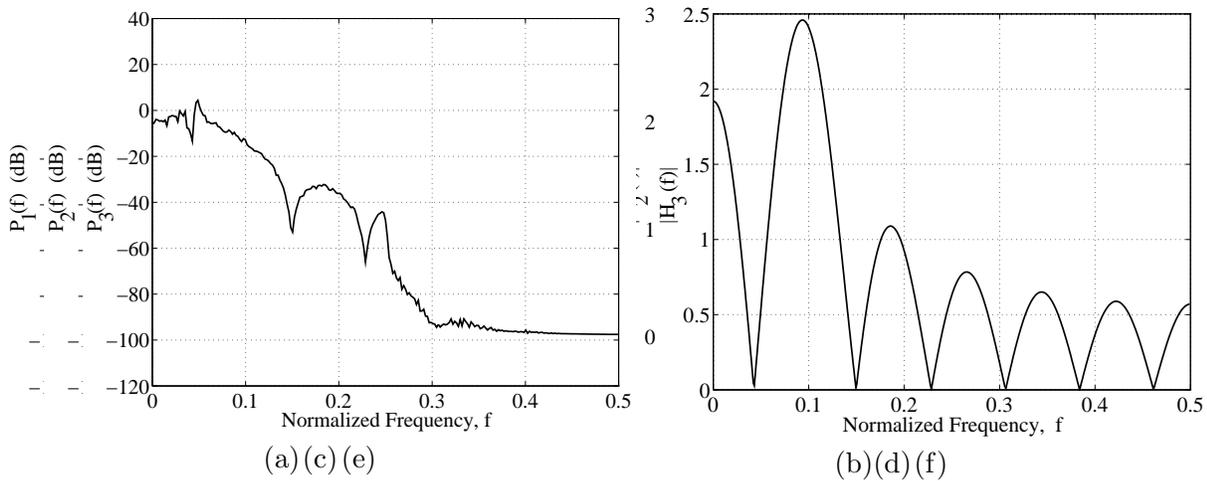


Figure 8.20: (a)

figure shows the magnitude frequency response of the preprocessing filters, $H_i(f)$, and the power spectrum of the output. The high gain requirement outside the narrow-band region is difficult to achieve since the nonlinear filter succeeding the preprocessor is *memoryless*.

It may consequently be concluded that the PCA preprocessing should merely be used in cases where a considerable overlap between the input and output spectra is present. However, naturally it is impossible to conclude that PCA always will be the best choice in this case. The modeling of Morison's equation is thus a counter-example of this claim.

²⁵Note that $p = 3$ was employed in the simulation.

²⁶See Fig. 8.10 with $P_x(f) = P_{y_s}(f)$ and $P_y(f) = P_{x_s}(f)$ according to Fig. 8.17.

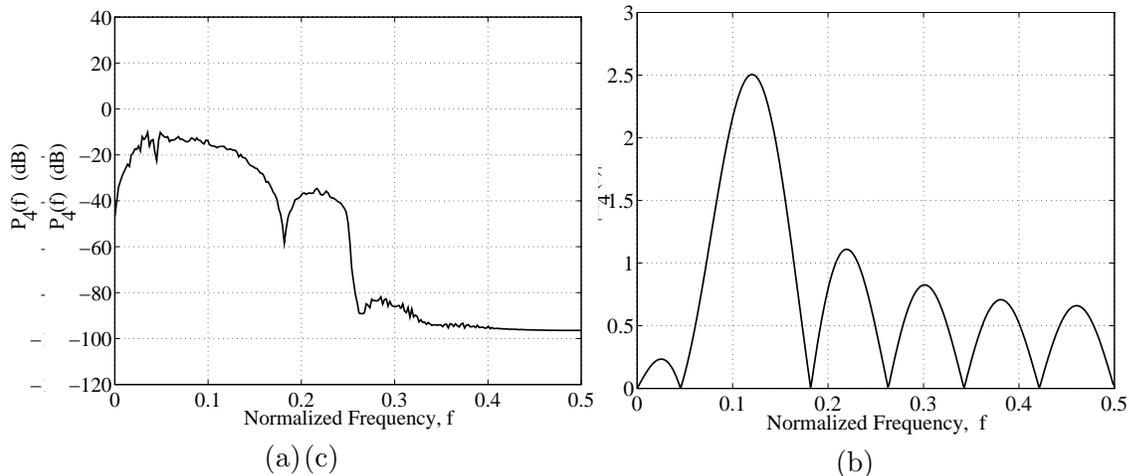


Figure 8.20: (b) Inverse modeling of the pendulum using PCA preprocessing. $P_i(f)$, $P_y(f)$ are the power spectra of the signals, $z_i(k)$, and the output $y(k)$, respectively. In addition the magnitude frequency response of the preprocessing filters, $H_i(f)$, are depicted²⁸. The power spectra is obtained by the Welch method [Oppenheim & Schaffer 89, Sec. 11.6] with 50% overlapping Hanning-weighted spectra of 512 bins.

The PCA preprocessing is superior to the DPP when equalizing the nonlinear channel and predicting the Mackey-Glass time-series. This is perhaps due to the fact that derivatives do not naturally enter the model²⁹. In addition, the input-output power spectra have a large degree of overlap – in fact, they become identical in the Mackey-Glass case.

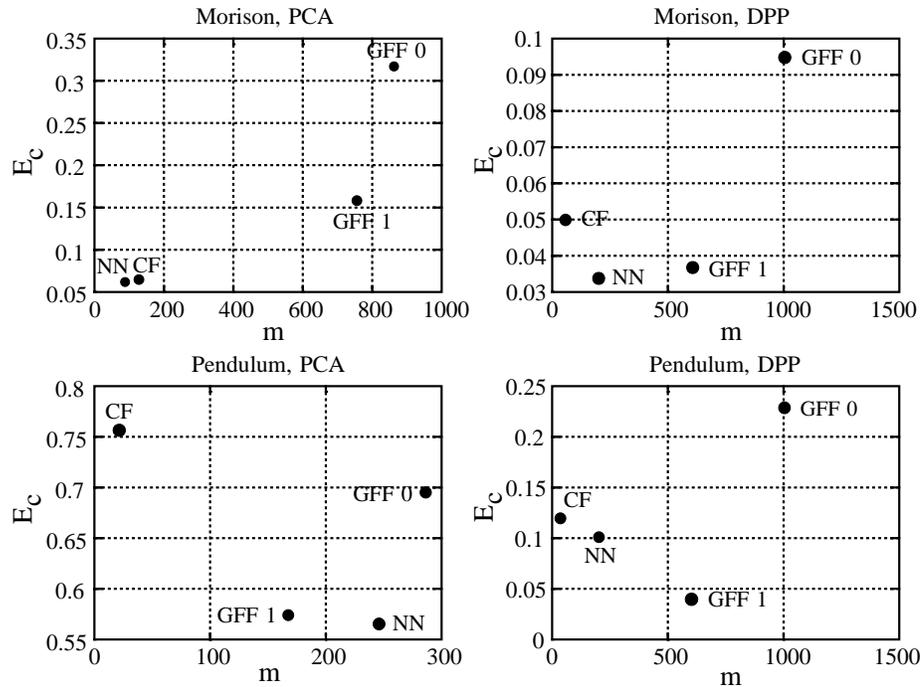
Moreover, it should be noted that the results concerning the LL-model Table 8.5 may guide the choice of a suitable preprocessor since – in all cases – the preprocessor which gave the smallest E_c when using the LL-model also turned out to be the best when using the neural network model. This observation is connected with the fact that all memory is located within the preprocessor.

In order to further substantiate the utility of the neural network filter architecture a comparison with XN-filter architectures simulated in [Hoffmann 92a, Sec. 8.1] is provided. The architectures comprise:

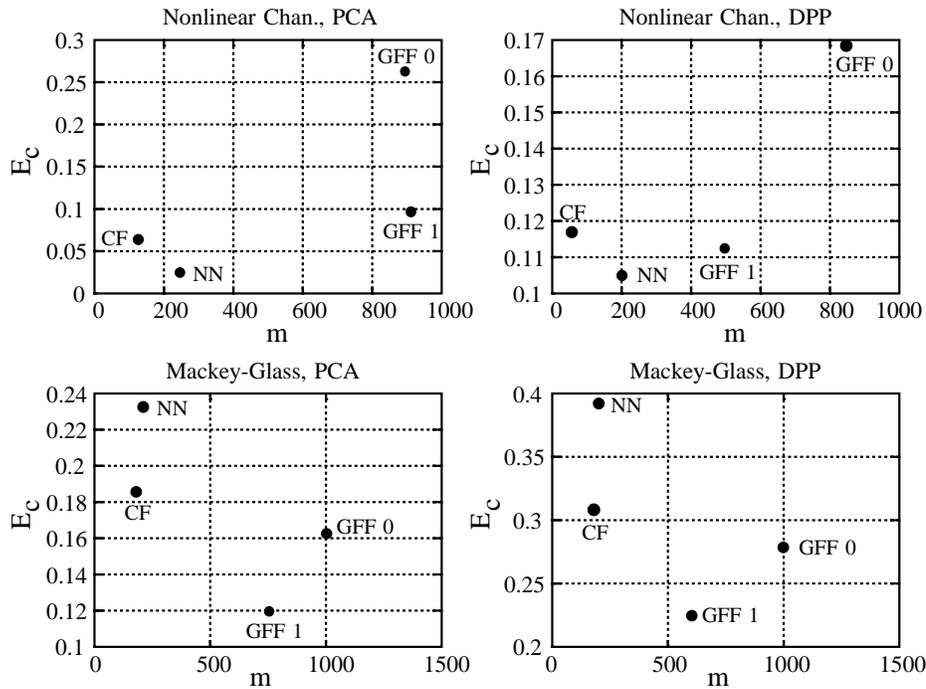
1. The Chebyshev filter (see e.g., Sec. 3.2.1.3).
2. The gate function filter (GFF) (see Sec. 3.3.3) with polynomial basis functions with zero and first order, i.e., the models act locally as a bias term and a linear filter, respectively.

The (best) results attained in [Hoffmann 92a, Sec. 8.1] when employing the Chebyshev and gate function filters are summarized in Table 8.8. In addition, Fig. 8.21 provides a graphical comparison w.r.t. the mentioned architectures. The relative cross-validation error index, E_c , is plotted as a function of the total number of weights in the model, m . m conveys

²⁹Even though the Mackey-Glass time-series is described by an first order differential equation, a future sample is not obviously a function of the derivative, $y'_s(t)$.



(a)



(b)

Figure 8.21: Comparison of different filter architectures. The relative cross-validation error index, E_c , is plotted as a function of the total number of weights, m . NN denotes the best 2-layer neural network architecture, cf. Table 8.7, CF denotes the Chebyshev filter, and GFF 0, GFF 1 denote the zero and first order gate function filters, respectively.

– to some extent – the complexity of the model with respect to computational burden and the required number of training examples in order to ensure proper generalization, cf. Ch. 6. In many cases the neural network outperforms the other architectures with m comparable or even less³⁰. However, when predicting the Mackey-Glass time-series the neural network performs worse. No convincing arguments explaining this fact have been discovered. On the other hand, various considerations are still viable. It might be the case that the cost function is afflicted with a lot of local minima and “flat” regions which increase the possibility that the weight estimation algorithm get stuck in poor minima. Another explanation is based on the observation that the gate function filters – which are local approximation filters – performs better than the global approximating filters, viz. the Chebyshev and the neural network filters; consequently, the nature of the input-output surface³¹ is probably better captured by a local approximation filter. In the case of using PCA preprocessing Fig. 8.22 shows a projection of the 4-dimensional input-output surface onto the subspace spanned by $[z_1(k), z_3(k), y(k)]$. The surface possesses

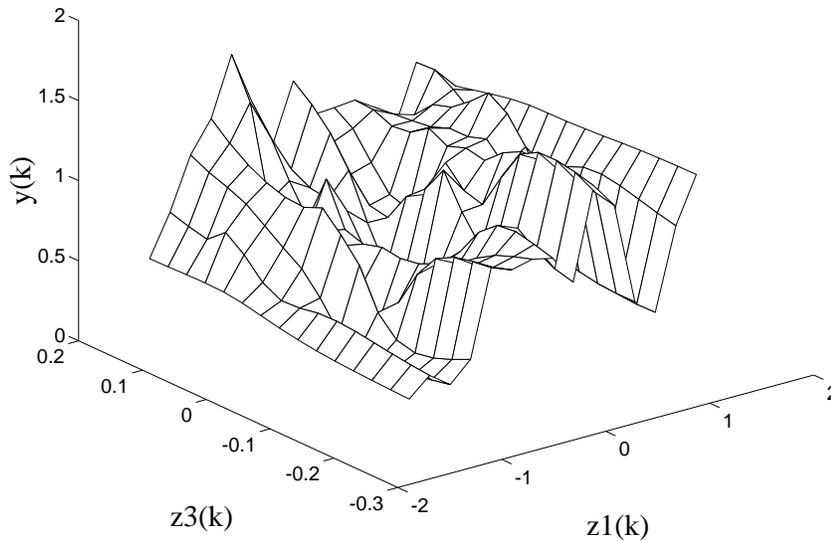


Figure 8.22: Projection of the 4-dimensional Mackey-Glass input-output surface, $\varphi(z_1(k), z_2(k), \dots, z_4(k), y(k)) = 0$ onto the subspace spanned by $[z_1(k), z_3(k), y(k)]$ when using PCA preprocessing.

³⁰In connection with the GFF’s methods for optimizing the architecture were employed; however, the architecture of the neural network was not optimized. Hence, optimizing the architecture may possibly improve the generalization ability, subsidiarily, confine the complexity.

³¹The input-output surface is the surface spanned by the dynamics of the Mackey-Glass system within the space: $[z_1(k), z_2(k), \dots, z_p(k), y(k)]$ where $p = 4$ in the present case. That is, the surface is given by: $\varphi(z_1(k), z_2(k), \dots, z_4(k), y(k)) = 0$ where $\varphi(\cdot)$ is the nonlinear function specifying the dynamics of the Mackey-Glass system.

relatively high dynamics. Thus in certain regions of the $[z_1, z_3]$ -space the surface is nearly horizontal; contemporary, it is pretty steep in other regions. Consequently, it may be easier to approximate the surface by several hyperplanes – as done within the first order gate function filter – contrary to performing global approximation, e.g., by means of a neural network.

Prediction of the Mackey-Glass time-series has – especially in the neural network community – become a benchmark problem. In Table 8.9 various achievements are summarized.

Notes:

- None of the mentioned works used PCA preprocessing. Instead, a resampling technique based on the theorem of Takens [Takens 81] was employed. That is,

$$\mathbf{z}(k) = [x(k), x(k - \delta), \dots, x(k - \delta(p - 1))]^\top \quad (8.33)$$

where $\delta = 6$ is the resampling period. Usually $p = 4$ was employed, i.e., $\mathbf{z}(k) = [x(k), x(k - 6), x(k - 12), x(k - 18)]^\top$, and consequently the memory length is $L = 19$. However, no particular difference between these two methods is observed. Employing a LL-filter with $L = 19$ the resampling technique gives ($N = N_c = 8850$) $E_c = 0.5429$, while $E_c = 0.5505$ is achieved with PCA ($L = 19, p = 4$).

- The output neurons of the referred multi-layer perceptron neural networks (MFPNN) are nonlinear.
- In [Farmer & Sidorowich 88], [Stokbro et al. 90] and [Wulff & Hertz 92] *iterative prediction* was used. Suppose that the prediction time is D_x . Instead of using the *direct prediction* training setup, i.e.,

$$\begin{aligned} \mathbf{x}(k) &= [x(k - D_x), x(k - D_x - 1), \dots, x(k - D_x - L + 1)]^\top \\ y(k) &= x(k) \end{aligned}$$

the filter is trained to perform a *one-step prediction*. Let $f(\cdot)$, $\hat{\mathbf{w}}$ denote the nonlinear filtering function and the estimated weights, respectively, then the one-step predictor becomes:

$$\hat{x}(k + 1) = f\left([x(k), x(k - 1), x(k - L + 1)]^\top; \hat{\mathbf{w}}\right). \quad (8.34)$$

A D_x -step predictor is then obtained by the iterative scheme³²:

for $n = 1, 2, \dots, D_x$

$$\hat{x}(k + n) = f\left([\hat{x}(k + n - 1), \dots, \hat{x}(k + 1), x(k), \dots, x(k + n - L)]^\top; \hat{\mathbf{w}}\right)$$

end

In [Farmer & Sidorowich 88] it is demonstrated that iterative prediction generally is preferred to direct prediction. In consequence, the results obtained in Table 8.7 are possibly better due to iterative prediction.

³²If the resampling technique (mentioned in the first item) was used then a 6 step predictor is considered. Consequently, it is only possible to perform the prediction times: $D_x = 6n, n = 1, 2, \dots$.

- Some of the results concerns using $D_x = 85$ rather than $D_x = 100$. Due to the inherent divergence associated with the chaotic attractor which is characterized by the so-called Liapunov exponents (see e.g., [Farmer & Sidorowich 88]) the mean square prediction error is smaller when considering $D_x = 85$. In fact, it is possible to show that the mean square prediction error scales faster or equal to $\exp(\lambda_{\max} \cdot n)$ where λ_{\max} is the largest (positive) Liapunov exponent ($\approx 5.4 \cdot 10^{-3}$ w.r.t. the Mackey-Glass attractor) and n is the prediction time.
- The reported findings differ in the way of calculating the relative cross-validation error index, even though several are rather scanty w.r.t. this subject. Recall that in this Thesis the relative cross-validation error is defined on an *independent* cross-validation set³³ as we consider modeling stationary systems. In other findings the cross-validation set succeeds the training set immediately, and consequently, possibly non-stationarities are taken into account. Hence, we expect that E_c is underestimated. [Stokbro et al. 90] used this approach for sure, and it is believed that also [Lapedes & Farber 87] and [Moody & Darken 89] did. As a matter of fact, in [Moody & Darken 89] the number of adjustable weights are incredibly large compared to the number of training data, e.g., 541 weights of a 3-layer neural network were estimated using merely 500 training data! Accordingly, an extremely poor generalization ability (defined as usual) is expected. On the other hand, [Wulff & Hertz 92] used an independent cross-validation set approach and the results concerning MFPNN's are comparable of those reported in Table 8.7.

8.4 Summary

In this chapter the abilities of multi-layer feed-forward perceptron neural networks for the implementation of various signal processing tasks were studied numerically.

At first we focused on the basic algorithms for training neural networks. It was demonstrated that that the Weight Initialization Algorithm presented in Sec. 5.3.2 provides a practicable alternative to pure random weight initialization.

Next, a comparison of various weight estimation algorithms were provided. The fact that recursive second order algorithms converge faster than first order algorithms was confirmed. In particular, we compared the Recursive Gauss-Newton algorithm with Bierman Factorization (RGNB) to the Stochastic Gradient (SG), and the Normalized Stochastic Gradient (NSG) algorithms. By fast convergence we here consider that the performance (relative training error index) increases rapidly as a function of the computational complexity. Furthermore, the convergence properties of the off-line Modified Gauss-Newton (MGN) algorithm were treated. It was concluded that the MGN-algorithm very likely got trapped in poor local minima. In summary, it is suggested that the RGNB-algorithm is applicable for training neural networks. However, it should be emphasized that the algorithms have been evaluated on networks containing a moderate number of weights; consequently, the results are possibly not valid for networks with a huge number of weights³⁴.

Finally, the capabilities of the generic nonlinear filter architecture (GNFA) (cf. Ch. 4) were tested on various standard signal processing tasks including: System identification,

³³That means, the cross-validation set is far off the training set as regards time.

³⁴Within signal processing complexity considerations often is an essential part of the design, especially in connection with on-line processing. Consequently, we claim that the network sizes dealt with in this Thesis are realistic.

inverse modeling, and time-series prediction. The nonlinearity within the GNFA was implemented by using a 2-layer perceptron neural network (MFPNN).

The considered preprocessing methods included the Principal Component Analysis (PCA) and the Derivative Preprocessor (DPP). It was demonstrated that both methods are workable. If the resemblance among the spectra of the input and output of the model is poor then PCA may not be viable. If the system at hand cannot be approximated by using a few derivatives then DPP is not an obvious choice, and moreover, due to the high-pass response of the derivative filters amplification of noise may reduce the overall quality of the model. In all cases, the performance obtained by using an LL-filter guided the selection of a suitable preprocessing method.

The results obtained by using the generic architecture based on MFPNN's were compared to using Chebyshev filters and gate function filters which were simulated in [Hoffmann 92a]. The MFPNN gave comparable – and in several cases – better performance. However, the MFPNN did worse when predicting the Mackey-Glass time-series. Two explanations were listed: The cost function is possibly afflicted with many local minima and “flat” regions. Secondly, we noticed that the gate function filters – which are local approximation filters – perform better than the global approximating filters, i.e., the Chebyshev filter and the MFPNN. Consequently, the nature of the input-output surface is probably better captured by a local approximation filter. In addition, the MFPNN was compared to related neural network approaches aiming at predicting the Mackey-Glass series. The comparison is to some extent precluded by different implementation strategies, e.g., as regards the calculation of cross-validation performance. The performance was comparable to findings of [Wulff & Hertz 92] which used a cross-validation approach similar to that employed in this Thesis.

System	Preprocessor	Model	L	D_x	D_y	p	l	m	E_c
Morison	PCA	Cheb. Filt.	13	5	0	4	5	126	0.0641
Morison	PCA	GFF	13	5	0	4	0	864	0.3180
Morison	PCA	GFF	13	5	0	4	1	755	0.1580
Morison	DPP	Cheb. Filt.	13	2	0	3	5	56	0.0498
Morison	DPP	GFF	13	2	0	3	0	1007	0.0949
Morison	DPP	GFF	13	2	0	3	1	608	0.0369
Pendulum	PCA	Cheb. Filt.	13	0	9	5	2	21	0.7562
Pendulum	PCA	GFF	13	0	9	5	0	287	0.6949
Pendulum	PCA	GFF	13	0	9	5	1	168	0.5744
Pendulum	DPP	Cheb. Filt.	17	0	8	3	4	35	0.1193
Pendulum	DPP	GFF	17	0	8	3	0	1006	0.2291
Pendulum	DPP	GFF	17	0	8	3	1	604	0.0399
Nonlinear Chan.	PCA	Cheb. Filt.	13	0	0	5	5	126	0.0641
Nonlinear Chan.	PCA	GFF	13	0	0	5	0	895	0.2633
Nonlinear Chan.	PCA	GFF	13	0	0	5	1	912	0.0965
Nonlinear Chan.	DPP	Cheb. Filt.	17	0	5	3	5	56	0.1169
Nonlinear Chan.	DPP	GFF	17	0	5	3	0	847	0.1685
Nonlinear Chan.	DPP	GFF	17	0	5	3	1	496	0.1125
Mackey-Glass	PCA	Cheb. Filt.	13	100	0	4	6	180	0.1857
Mackey-Glass	PCA	GFF	13	100	0	4	0	1005	0.1626
Mackey-Glass	PCA	GFF	13	100	0	4	1	755	0.1197
Mackey-Glass	DPP	Cheb. Filt.	13	100	0	3	9	180	0.3076
Mackey-Glass	DPP	GFF	13	100	0	3	0	1002	0.2783
Mackey-Glass	DPP	GFF	13	100	0	3	1	604	0.2247

Table 8.8: Results obtained in [Hoffmann 92a, Sec. 8.1] using the Chebyshev filter and the gate function filter (GFF). L is the memory length, D_x , D_y are the delays of the input and output, respectively, p is the number of preprocessing filters, and l is the polynomial order. In connection with the GFF's the basis functions are polynomials with order 0 and 1. Finally, E_c is the relative cross-validation error index.

Reference	Model	m	D_x	N	N_c	E_c
[Farmer & Sidorowich 88]	Local linear filter	-	100	10000	500	0.005
[Lapedes & Farber 87]	[4, 10, 10, 1]-MFPNN	171	100	500	500	0.05
[Moody & Darken 89]	[4, 20, 20, 1]-MFPNN	541	85	500	500	0.05
[Moody & Darken 89]	Loc. recep. field	8850	85	8850	500	0.03
[Stokbro et al. 90]	Loc. recep. field	125	100	500	1000	0.035
[Wulff & Hertz 92]	[15, 10, 1]-MFPNN	171	100	500	250	0.32
[Wulff & Hertz 92]	[4, 10, 10, 1]-MFPNN	171	100	500	250	0.25
[Wulff & Hertz 92]	Recurrent neural net	≈ 100	100	500	250	0.09

Table 8.9: Comparison of models for predicting the Mackey-Glass time-series. m is the number of parameters, D_x is the delay on the input corresponding to the prediction time, N , N_c are the number of training and cross-validation samples, respectively, and finally, E_c is the relative cross-validation error index (the numbers are approximate). The local linear filter and the localized receptive field network are discussed in Sec. 3.4.1 and Sec. 3.3.1, respectively. The localized receptive filter in [Moody & Darken 89] was of zero order, while a first-order filter was used in [Stokbro et al. 90].

CHAPTER 9

CONCLUSION

In recent years much attention was directed to adaptive models as devices for design of flexible signal processing systems. This is due to the fact that they are able to learn a task from acquired data which indeed is valuable when the underlying physical mechanisms are difficult – or impossible – to clarify. Furthermore, the models possess the ability to generalize to cases which were not explicitly specified by the learning examples, i.e., they capture the structure of the unknown system. The effort has up to now mainly been on linear adaptive models which have been successfully applied to various signal processing tasks including: System identification, adaptive control systems, equalization of communication channels, noise reduction, speech coding, prediction of time series. However, often it appears that linear models are approximations of systems which fundamentally are *nonlinear* by nature, and consequently a large potential consists in extending the models to be nonlinear. Within signal processing the literature provides several examples which substantiate the usefulness of nonlinear models, e.g., [Bendat 90], [Chen et al. 90c], [Falconer 78], [Farmer & Sidorowich 88], [Narendra & Parthasarathy 90], [Nguyen & Widrow 89], [Lapedes & Farber 87], which cover the subjects mentioned above.

In this Thesis we focused on neural network architectures for implementation of the non-recursive, nonlinear adaptive model with additive error. The model forms a nonlinear relation between the input and output signals which both are assumed to be one-dimensional. The purpose is to adapt “black-box” models for various general signal processing tasks, such as: System identification, inverse modeling and time-series prediction. The objective of the Thesis was the study of various phases involved in the design of a suitable model architecture from a relatively limited number of training data. This includes:

- Selection of a proper basic architecture.
- Estimation of the model from acquired data.
- Model validation and architecture optimization.

The choice of a proper basic architecture was in Ch. 4 formulated in terms of the *generic nonlinear filter architecture*. The architecture is composed of a *preprocessing unit* and a *canonical filter*, and may be viewed as a heterogeneous three-layer neural network.

The main objective of the preprocessing unit is to project a lag space of the input signal onto a low dimensional signal vector which acts as input to the succeeding canonical filter. This is done in order to control the complexity of the canonical filter so that it is possible

to perform proper estimation from a limited data set; paraphrased, to avoid the “curse of dimensionality”. The choice of preprocessor is indeed a difficult task since – in fact – it cannot be designed independent of the succeeding canonical filter. Moreover, the preprocessor design should incorporate any a priori knowledge that may be available. We suggested three simple preprocessors:

- The principal component analysis (PCA) preprocessor which from numerical simulations seems practicable if the spectra of the input and output have a large overlap. This is indeed the case when considering time-series prediction.
- The derivative preprocessor (DPP) which is validated by simulations. A major drawback is high sensitivity to noise on the input.
- The Laguerre function preprocessor which is motivated by the ability to handle filters with infinite memory length.

The canonical filter discussed in Ch. 3 emerges from the formulation of a novel architecture taxonomy. The taxonomy is based on a few significant architectural properties which include:

- Global versus local approximation.
- Parametric versus nonparametric architectures.

Dealing with parametric architectures the mapping of the input vector signal onto the output signal is characterized by a weight vector which is estimated from a training data set, i.e., the estimated weights define an encoding of the training data. On the other hand, in a nonparametric architecture no encoding is done, i.e., the determination of an output sample requires explicit knowledge of all training data (sometimes a suitable subset). The Thesis mainly focused on parametric architectures.

Within local approximation filters the input space is divided into a number of subsets. In each subset a local filter is formulated, and the final output results by combining the outputs of the local filters. A global approximation approach is characterized by the fact that no dividing of the input space is performed. A further refinement of local filters consists in characterizing the shape and the connection of the subsets. In particular, the partition function filters - which possess disjoint subsets - are interesting since the weights associated with the distinct filters can be estimated independently; however, it still may be problematic to determine the boundaries which separate the subspaces. The use of local approximation involves a trade off between the number of local filters and the complexity of the filter in order to avoid the “curse of dimensionality”. In particular, that means, if data are sparse only a few local filters are possible – eventually we better use global approximation. The trade off may be settled by using architecture optimization techniques (mentioned below).

In summary, the intention was to provide a unified view of possible architectures, and the taxonomy might be used to guide the choice of a suitable architecture when facing a particular modeling task. Furthermore, the taxonomy may suggest the development of novel structures. Finally, since the taxonomy is based on a few simple architectural properties only, future research definitely may convey to a further refinement.

Ch. 3 also provided a survey of numerous architectures suggested in the literature – in terms of the canonical filter representation. Particularly, we attached importance to various interpretations of the signal processing within the multi-layer feed-forward perceptron neural network (MFPNN) which belongs to the class of global approximation filters.

In Ch. 8 the capabilities of the generic nonlinear filter architecture were tested on various synthetic signal processing tasks, including: System identification, inverse modeling (e.g., channel equalization), and prediction of chaotic time-series. A 2-layer feed-forward perceptron neural network implementation of the canonical filter was employed, and as preprocessors we used PCA and DPP. The findings were compared to results obtained by using a linear filter, and by using other selected types of nonlinear filters (reported in [Hoffmann 92a]) which were: The Chebyshev filter, zero- and first-order gate function filters. The investigated architectures indeed outperformed the linear filter. In addition – in many cases – they outperformed the other mentioned nonlinear filter architectures. However, no general conclusions can be drawn since the results are highly dependent on various matters such as, the number of training data, the task under consideration, etc.. Moreover, a fair comparison should include statements concerning the required computational complexity.

The next item of the design phase is model estimation, i.e., estimation of the model weights. Ch. 5 dealt with the application of standard first and second order optimization schemes for layered filter architectures, such as the MFPNN. We demonstrated how the well-known back-propagation algorithm can be generalized to deal with general layered filter architectures in order to implement both first order gradient descent algorithms and second order Gauss-Newton algorithms. As the cost function we used the sum of the least squares cost and a weight decay regularization term. In Ch. 8 the capabilities of various algorithms were tested in connection with the MFPNN architecture. The simulations showed that the recursive second order algorithms indeed outperform the stochastic gradient algorithms – in terms of the required computational complexity – to reach a certain level of performance. Moreover, simulations with off-line algorithms indicated that these often are trapped in local minima; consequently, we recommend the use of recursive algorithms which have a built-in mechanism for escaping local minima. Furthermore, in connection with 2-layer perceptron neural networks we developed a weight initialization algorithm. The object was to reduce the resemblance among the response of the hidden neurons and with that, the eigenvalue spread of the Hessian matrix (of the cost function). This helps in speeding up the algorithm convergence – especially when dealing with first order algorithms – as demonstrated by simulations.

The last design phase is model validation and architecture optimization which is dealt with in Ch. 6. At first, a suitable measure of model quality has to be defined. We employed the generalization error which is defined as the expected squared error on a test sample which is independent of the data used for training, but originates from the same probability distribution. That is, we consider stochastic input-output signals. Since the generalization error depends on the actual training data it may be rather noisy; consequently we focused on the average generalization error – where the average is w.r.t. training data. In order to clarify various matters influencing the average generalization, we suggested to perform a *model error decomposition*, q.e., the average generalization error is the sum of:

1. The inherent noise variance, which specifies the amount of variance of the output which cannot be explained by knowledge of the input.
2. The mean square model error (*MSME*), which defines the minimal increase in average generalization when applying a particular model for a particular task. If the model is *complete* *MSME* equals zero; whereas it is non-zero when the model is *incomplete*, i.e., not able to model the task perfectly.

3. The weight fluctuation penalty (*WFP*), which stems from the fact that the model is estimated from a finite data set.

Under mild conditions, the *MSME* decreases – or remains unchanged – when the complexity of the filter increases, i.e., more weights are added. On the other hand, the *WFP* typically increases with increasing filter complexity. Consequently, there will normally be a trade off between these two terms (also known as the bias/variance dilemma). However, when dealing with incomplete models there is no theoretical guarantee that the *WFP* actually increases. The decomposition furthermore led to a discussion of the fundamental limitations in the search for an optimal architecture. The result is in summary: It is impossible to predict how an expansion of the architecture will influence the average generalization error without assumptions on the system which generated the data. Even if a priori knowledge is available, it is not necessarily obvious how to incorporate this knowledge. However, it may still be possible to predict the change when reducing the architecture, e.g., by eliminating weights.

In addition, the benefits of using regularization, with reference to reducing the average generalization error, were discussed in terms of the model error decomposition. Concerning the linear complete model, it was demonstrated analytically, that the average generalization error may be reduced by incorporating a weight decay regularization. This result is a modification of the findings in [Krogh & Hertz 91]. However, the optimal setting of the regularization parameter is highly dependent on the system being modeled, so only suboptimal procedures may be available.

On the basis of the above stated results we treated two different types of algorithms for architecture optimization – or synthesis:

- The first class is based on comparing the *estimated* average generalization error associated with a set of possible architectures, and finally selecting the one with minimal average generalization error.
- The other class is based on a statistical hypothesis testing framework.

The subject of estimating the average generalization has already been addressed in the literature. This comprises e.g., various cross-validation estimators, Akaike’s Final Prediction Error (*FPE*) estimator, and Moody’s Generalized Prediction Error (*GPE*) estimator which all were reviewed. The cross-validation estimators are appealing since they do not presume anything about the model; however, some of the data are reserved for cross-validation, and consequently, when data are sparse, the amount of data left over for training is inappropriately low. The remaining estimators differ mainly on two model assumptions: Complete versus incomplete models, and linear versus nonlinear models (w.r.t. the weights). The lack of a priori knowledge – which led to consideration of “black-box” models – typically results in that complete models cannot be guaranteed. Consequently – in the present context – incomplete, nonlinear models are the case. The only estimator which explicitly take these matters into account is – to the knowledge of the author – the *GPE*-estimator. The average over the training set – in order to constitute the average generalization error – is explicitly done by averaging both w.r.t. to input and output samples; however, within the *GPE*-estimator the averaging w.r.t. to the input is neglected. Emphasis has consequently been put into the development of a novel generalization error estimator called *GEN*: The Generalization Error Estimator for Incomplete, Nonlinear Models which may be viewed as an extension of both the *FPE* and the *GPE* estimators. The estimator is encumbered

with some advantages and drawbacks which – in a concise form – are:

Advantages:

- All data in the training set are used for estimating the weights. This is especially important when data are sparse.
- The model may be linear as well as nonlinear.
- Both incomplete and complete models are treated.
- The input may be correlated as well as white. In a signal processing context we often deal with correlated input.
- The inherent noise may be correlated and dependent on the input. This extends the usual assumption that the inherent noise is an i.i.d. sequence independent of the input.
- Noiseless systems are also considered.
- The estimated weights are not required to be the global minimum of the cost function.
- The possibility of including a regularization term in the cost function is treated.

Drawbacks:

- A fundamental prerequisite for the derivation is a large training set. In fact, this assumption enters all considered generalization error estimators.
- The model is assumed to be properly approximated by a linear expansion in the weights in the vicinity of the weight estimate.
- The weight estimate is assumed to be locally unique, i.e., the cost function has a non-zero curvature.
- Only local effects due to fluctuations in the weight estimate are considered.
- The effects of imperfect training are not accounted for.

In Ch. 7 we tested the quality of the *GEN* on selected numerical examples by comparison with the cross-validation estimators and the *FPE*-estimator. In most cases quantitative arguments led to the conclusion that the *GEN*-estimator is a profitable alternative. The (relative) bias of *GEN* is typically lower than that of *FPE*; but larger than those of the cross-validation estimators. On the other hand, the average squared error of *GEN* is comparable to that of *FPE*. However, the average squared error afflicted with the cross-validation estimators are typically tremendously larger than that of *GEN*. Still there is need for further investigations in order to clarify the functionality of the *GEN* in connection with architecture optimization.

The other class of algorithms for architecture synthesis are based on a statistical hypothesis testing framework. In general, we test the hypothesis of model reduction, i.e., testing restrictions on the weight space. However, the hypotheses have to be formulated a priori, so the only general appealing hypothesis is to test if some of the weights can

be removed. A so-called Wald test statistic was employed for this purpose. The crucial quantity entering this test statistic is the covariance matrix of the estimated weight vector. Various estimates of the covariance estimator were derived, and furthermore, it was demonstrated that the distinguishing between complete and incomplete models plays a decisive part.

Both the mentioned types of synthesis algorithms do not address the selection of suitable candidate architectures. For that purpose we suggested three different Weight Deletion Vector (WDV) algorithms which result in a set of candidate weights which subsequently may be tested for removal by using the statistical testing framework – or by using algorithms based on comparing the generalization error. The first algorithm is inspired by the well-known Optimal Brain Damage (OBD) algorithm, and the other may be viewed as generalizations hereof. Contrary to OBD the extended WDV-algorithms take correlation among the weight estimates into account. In Ch. 7 we tested these algorithms in conjunction with the *FPE*-estimator and the statistical testing framework. The simulations indicated that a preference to the statistical testing framework relative to using *FPE*. In addition, the WDV-algorithm which takes weight correlation into account outperformed the one based on OBD.

In summary, the Thesis provides a survey of the phases involved in the design of neural network filters, including: Selection of a proper basic architecture, model estimation, architecture optimization. Various algorithms were proposed and validated by means of analytical results and simulations.

APPENDIX A

GENERALIZATION ERROR ESTIMATES FOR XN-MODELS

In this appendix the derivation of generalization error estimates for XN-models are presented. The aim is mainly to handle NN-models, i.e., models which are nonlinear in both the weights and the mapping. Estimation of the generalization error, which measures the quality of a model, is an important tool for selecting among competing models.

The derivation is done in two cases: When estimating the model parameters by minimizing the usual least squares (LS) cost function and when minimizing the LS cost function with a regularization term. We denote the estimator: *GEN*, i.e., **Generalization error estimator for incomplete, nonlinear models**. See also App. I.

A.1 The Basis of Estimating the Generalization Error

This section presents the framework in which the suggested generalization error estimators are derived.

A.1.1 Systems and Models

We consider discrete nonlinear systems of the form:

$$y(k) = g(\mathbf{x}(k)) + \varepsilon(k) \tag{A.1}$$

where

- k is the discrete time index,
- $y(k)$ the output signal,
- $\mathbf{x}(k)$ denotes the multivariate (vector) input signal,
- $g(\cdot)$ constitutes a nonlinear mapping,
- $\varepsilon(k)$ is an inherent noise.

ASSUMPTION A.1 *The input $\mathbf{x}(k)$ and the inherent noise $\varepsilon(k)$ are assumed to be strictly stationary sequences. Furthermore, $E\{\varepsilon(k)\} = 0$ and $E\{\varepsilon^2(k)\} = \sigma_\varepsilon^2$.*

However, in many cases it is possible to restrict Ass. A.1 to comply with:

ASSUMPTION A.2 The input $\mathbf{x}(k)$ is assumed to be a strictly stationary sequence and $\varepsilon(k)$ is a white¹, strictly stationary sequence with zero mean and variance σ_ε^2 . Furthermore, $\mathbf{x}(k)$ is assumed independent of $\varepsilon(k)$, $\forall k$.

ASSUMPTION A.3 $\mathbf{x}(k)$ is assumed to be an M -dependent stationary sequence², i.e., $\mathbf{x}(k)$ and $\mathbf{x}(k + \tau)$ are independent $\forall |\tau| > M$. M is denoted the dependence lag.

ASSUMPTION A.4 $g(\cdot)$ is assumed time invariant.

Let \mathcal{F} be a set of nonlinear functionals which are parameterized by an m -dimensional vector $\mathbf{w} = [w_1, w_2, \dots, w_m]^\top$. A feed-forward neural networks with hidden units is an example of \mathcal{F} . Now, let $f(\cdot) \in \mathcal{F}$. The XN-model³ of the nonlinear system in Eq. (A.1) becomes:

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w})\hat{y}(k) + e(k; \mathbf{w}). \quad (\text{A.2})$$

ASSUMPTION A.5 $f(\cdot)$ is assumed time invariant.

When considering an LX-model⁴ the filter output is explicitly expressed as:

$$f(\mathbf{x}(k); \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\varphi}(\mathbf{x}(k)) = \mathbf{w}^\top \mathbf{z}(k) \quad (\text{A.3})$$

where $\boldsymbol{\varphi}(\cdot)$ is an arbitrary time invariant linear or nonlinear mapping: $\mathbb{R}^L \mapsto \mathbb{R}^m$.

The error signal $e(k; \mathbf{w})$ is in general the sum of two components. Substituting Eq. (A.1) into Eq. (A.2) results in:

$$\begin{aligned} e(k; \mathbf{w}) &= y(k) - f(\mathbf{x}(k); \mathbf{w}) \\ &= \underbrace{g(\mathbf{x}(k)) - f(\mathbf{x}(k); \mathbf{w})}_{\text{Adjustment Error}} + \underbrace{\varepsilon(k)}_{\text{Inherent Noise}} \end{aligned} \quad (\text{A.4})$$

Within a specific model it is only possible to evaluate the error signal, that is, we are never capable of estimating the *adjustment error* unless it is known a priori that the system is noise free, i.e., $\varepsilon(k) \equiv 0$. The existence of a adjustment error is significant when estimating the generalization error as explained in Sec. A.2. This leads to the following definition:

DEFINITION A.1 If there exists parameters, \mathbf{w}° , such that $\forall \mathbf{x}(k)$ $g(\mathbf{x}(k)) \equiv f(\mathbf{x}(k); \mathbf{w}^\circ)$ we signify the model as complete; otherwise, as incomplete. \mathbf{w}° is denoted the true parameters⁵.

¹By “white” is simply meant: $E\{\varepsilon(k)\varepsilon(k + \tau)\} \propto \delta(k)$.

²A weaker assumption which may substitute As. A.3 aims at assuming $\mathbf{x}(k)$ to be a strongly mixing sequence [Rosenblatt 85, p. 62]. That is, $\mathbf{x}(k)$, $\mathbf{x}(k + \tau)$ is asymptotically independent as $|\tau| \rightarrow \infty$. Formally,

$$\sup_{\mathbf{x}(k), \mathbf{x}(k+\tau)} |\text{Prob}\{\mathbf{x}(k)\mathbf{x}(k + \tau)\} - \text{Prob}\{\mathbf{x}(k)\}\text{Prob}\{\mathbf{x}(k + \tau)\}| \rightarrow \mathbf{0}, \text{ as } |\tau| \rightarrow \infty$$

where $\text{Prob}\{\cdot\}$ denotes probability.

³Recall that the XN-model is linear or nonlinear in the weights and nonlinear in the mapping of \mathbf{x} .

⁴The LX-model is linear in the weights and linear or nonlinear w.r.t. the mapping of \mathbf{x} .

⁵Note that the true parameters (weights) are not necessarily unique. However, this is not crucial to the arguments in the following.

Usually the lack of knowledge concerning the structure of $g(\cdot)$ precludes the possibility of suggesting a complete model with finite order m . Consequently, we claim that incomplete models are the common case.

A.1.2 Estimation of Model Parameters

Given a set of connected input-output pairs, called the training set:

$$\mathcal{T} = \{\mathbf{x}(k); y(k)\}, \quad k = 1, 2, \dots, N \quad (\text{A.5})$$

the model is estimated by minimizing some cost function, say

$$C_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N c(k; \mathbf{w}). \quad (\text{A.6})$$

$c(k; \mathbf{w})$ is denoted the instant cost.

In this context the employed cost function is a sum of an ordinary least squares (LS) cost, $S_N(\mathbf{w})$, and a regularization term, $R_N(\mathbf{w})$:

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa R_N(\mathbf{w}) \quad (\text{A.7})$$

where

•

$$S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N (y(k) - f(\mathbf{x}(k); \mathbf{w}))^2. \quad (\text{A.8})$$

• $\kappa \geq 0$ is a regularization parameter which determines the trade off between the LS cost, S_N , and the regularization term R_N . Note that $\kappa = 0$ corresponds to no regularization at all.

•

$$R_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N r(k; \mathbf{w}) \quad (\text{A.9})$$

is the regularization term. Note that R_N in general depends on the training set, \mathcal{T} (however, often only the input).

The ordinary weight decay procedure is obtained by letting

$$R_N(\mathbf{w}) = r(k; \mathbf{w}) = \sum_{i=1}^m w_i^2, \quad (\text{A.10})$$

and κ is denoted the weight decay parameter.

Let Ω be a closed, bounded set (i.e., a compact set) in weight space over which the cost, C_N , is minimized.

DEFINITION A.2 Define \mathcal{W} as the set of weight vectors, $\hat{\mathbf{w}}$, which locally minimize *gend:trae*, i.e., let $\dim(\boldsymbol{\theta}) = \dim(\mathbf{w})$ then

$$\mathcal{W} = \{\mathbf{w} \in \Omega \mid \exists \delta > 0, \forall \|\boldsymbol{\theta}\| < \delta, C_N(\mathbf{w} + \boldsymbol{\theta}) \geq C_N(\mathbf{w})\} \quad (\text{A.11})$$

where $\|\cdot\|$ denotes a vector norm.

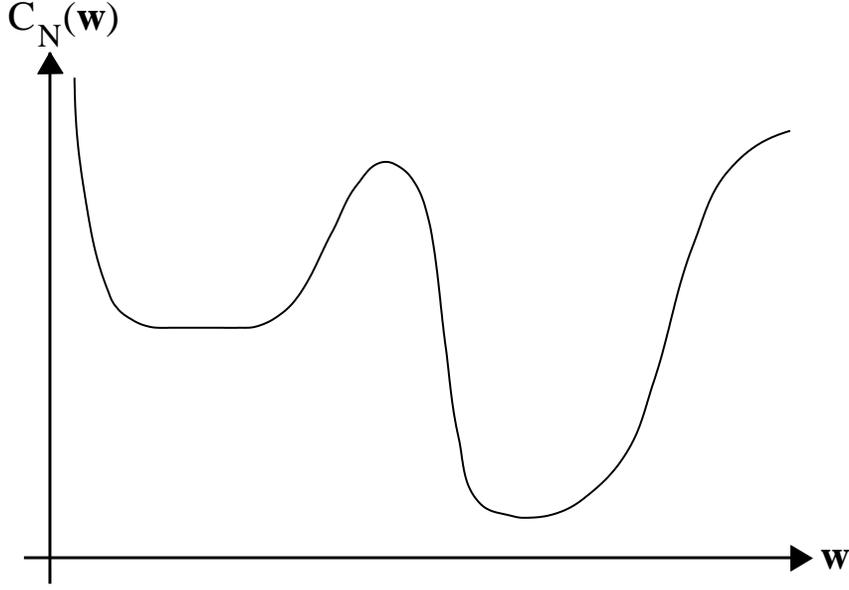


Figure A.1: General shape of the cost function. Notice the existence of local minima and that a minimum can be non-unique.

In Fig. A.1 the general shape of the cost function is shown. Notice three facts:

1. \mathcal{W} contains in general local minima.
2. The minima may be non-unique (i.e., the cost function is “flat”).
3. The curvature (the second order derivative) of the cost-function near a minimum may vanish, e.g., $C_N(w) \propto (w - \hat{w})^4$.

For the purpose of estimating the generalization error we restrict the cost function to comply with the following assumptions:

ASSUMPTION A.6 Assume that there exists a covering of Ω in compact subsets⁶:

$$\Omega = \bigcup_i \Omega(i) \tag{A.12}$$

such that $\hat{\mathbf{w}}(i) \in \Omega(i)$ uniquely minimizes $C_N(\mathbf{w})$ within the the partition $\Omega(i)$

ASSUMPTION A.7 Assume that $\forall i$ ⁷:

$$\frac{\partial C_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w}} = \mathbf{0}, \quad \mathbf{a}^\top \frac{\partial^2 C_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}. \tag{A.13}$$

⁶Note that this assumption is irrelevant when dealing with LX-models.

⁷Here and in the following, by definition,

$$\left. \frac{\partial C_N(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\hat{\mathbf{w}}(i)} = \frac{\partial C_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w}}.$$

Given As. A.6 and A.7, then \mathcal{W} according to Def. A.2 is the countable (possibly infinite) set:

$$\mathcal{W} = \{\hat{\mathbf{w}}(i)\}. \quad (\text{A.14})$$

A particular LS-estimator within the set \mathcal{W} is denoted $\hat{\mathbf{w}}$. Fig. A.2 shows an example of a cost function in accordance with the above assumption.

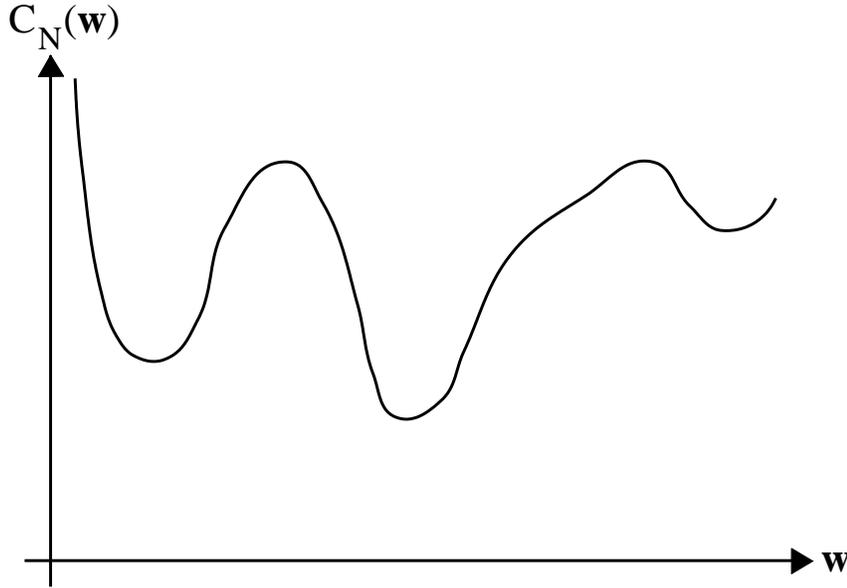


Figure A.2: Shape of a cost function which meet Ass. A.6 , A.7. Note that all minima are unique and that the curvature exists near a minimum.

The optimal weight vector set \mathcal{W}^* is the set of weight vectors which minimize the *expected cost function*

$$C(\mathbf{w}) = E_{\mathbf{x},\varepsilon}\{c(\mathbf{w})\} \quad (\text{A.15})$$

where $c(\mathbf{w})$ by gend:ck (A.7) and (A.1) is

$$\begin{aligned} c(\mathbf{w}) &= e^2(k) + r(\mathbf{w}) \\ &= (y - f(\mathbf{x}; \mathbf{w}))^2 + r(\mathbf{w}) \\ &= (g(\mathbf{x}) - f(\mathbf{x}; \mathbf{w}) + \varepsilon)^2 + r(\mathbf{w}) \end{aligned} \quad (\text{A.16})$$

and $E_{\mathbf{x},\varepsilon}\{\cdot\}$ denotes expectation w.r.t. the joint p.d.f. of $[\mathbf{x}, \varepsilon]$.

The weights are optimal in the sense that they reflect the “best” attainable weights in the actual model and with respect to the employed cost function.

Similar to As. A.6 and A.7 we make the following assumptions:

ASSUMPTION A.8 *Assume that there exists a covering of Ω in compact subsets⁸ (in general*

⁸Note that this assumption is irrelevant when dealing with LX-models.

different from that in `gend:lssplit`):

$$\Omega = \bigcup_i \Omega^*(i) \quad (\text{A.17})$$

such that $\mathbf{w}^*(i) \in \Omega^*(i)$ uniquely minimizes $C(\mathbf{w})$ within the the partition $\Omega^*(i)$

ASSUMPTION A.9 Assume that $\forall i$:

$$\frac{\partial C(\mathbf{w}^*(i))}{\partial \mathbf{w}} = \mathbf{0}, \quad \mathbf{a}^\top \frac{\partial^2 C(\mathbf{w}^*(i))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}. \quad (\text{A.18})$$

DEFINITION A.3 Given As. A.8 and A.9, \mathcal{W}^* is defined as the countable (possibly infinite) set:

$$\mathcal{W}^* = \{\mathbf{w}^*(i)\}. \quad (\text{A.19})$$

A particular optimal weight vector within the set \mathcal{W}^* is denoted \mathbf{w}^* .

A.1.2.1 Consistency

Here we only deal with strong consistency and make the following definition:

DEFINITION A.4 The estimator $\hat{\mathbf{w}}$ is a strongly consistent estimator of \mathbf{w}^* if

$$\text{Prob}\{\hat{\mathbf{w}} \rightarrow \mathbf{w}^*\} = 1, \quad \text{as } N \rightarrow \infty. \quad (\text{A.20})$$

In [Seber & Wild 89, Ch. 12] various assumptions leading to strong consistency are given. However, we will apply the result of [White 81, Theorem 2.1] because it is applicable for both incomplete models and for models estimated by minimizing a cost function with a regularization term. We state the necessary additional assumptions:

ASSUMPTION A.10

1. The instant cost given by `gend:ct` is assumed to be a continuous function of \mathbf{w} on a compact set and a measurable function of \mathbf{x} and ε , $\forall \mathbf{w}$.
2. Suppose the existence of a function $\zeta(\mathbf{x}, \varepsilon)$ which complies with

$$E_{\mathbf{x}, \varepsilon}\{\zeta(\mathbf{x}, \varepsilon)\} < \infty \quad (\text{A.21})$$

so that $c(\mathbf{w}) \leq \zeta(\mathbf{x}, \varepsilon)$, $\forall \mathbf{x}, \varepsilon, \mathbf{w}$.

3. Every data sample in the training set is assumed to be a random sample of the joint distribution of $[\mathbf{x}(k), \varepsilon(k)]$.

THEOREM A.1 Suppose that As. A.1, A.8, A.10 hold and recall that $\mathbf{w}^*(i)$ uniquely minimizes $C(\mathbf{w})$ within the compact subset $\Omega^*(i)$. If $\hat{\mathbf{w}}$ minimizes $C_N(\mathbf{w})$ within $\Omega^*(i)$ then $\hat{\mathbf{w}}$ is a strongly consistent estimator of $\mathbf{w}^*(i)$ as $N \rightarrow \infty$.

PROOF See Theorem 2.1 of [White 81]. ■

A.1.3 Generalization Ability

The quality of an estimated model is judged as good if the model has the ability to generalize. By that is meant that the model is able to predict the output signal properly using input data which are not present in the training set. In this context we make the following definition:

DEFINITION A.5 *The generalization error⁹, G , is defined as the expected, squared error on a test sample, $\{\mathbf{x}_t; y_t\}$, which is independent of the training set but originates from the same distribution.*

$$\begin{aligned} G(\mathbf{w}) &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ [y_t - f(\mathbf{x}_t; \mathbf{w})]^2 \right\} \\ &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ e_t^2(\mathbf{w}) \right\}. \end{aligned} \quad (\text{A.22})$$

$E_{\mathbf{x}_t, \varepsilon_t} \{\cdot\}$ denotes expectation with respect to the joint p.d.f. of $[\mathbf{x}_t, \varepsilon_t]$. Note that G depends both on the model, $f(\cdot)$, and the weights, \mathbf{w} . Clearly the model has a better generalization ability the smaller the generalization error is.

The LS cost function, $S_N(\hat{\mathbf{w}})$ given by `gend:sn`, is usually not a reliable measure of the quality of a model because it depends on the actual training set. However, if $e^2(k; \mathbf{w})$ is a mean-ergodic sequence, i.e., cf. [Papoulis 84a, Ch. 9-5]

$$E_{\mathbf{x}(k), \varepsilon(k)} \{e^4(k; \mathbf{w})\} < \infty, \quad (\text{A.23})$$

and

$$E_{\mathbf{x}(k), \varepsilon(k)} \{e^2(k; \mathbf{w})e^2(k + \tau; \mathbf{w})\} \rightarrow 0, \quad \text{as } \tau \rightarrow \infty. \quad (\text{A.24})$$

then

$$\lim_{N \rightarrow \infty} \{S_N(\mathbf{w})\} = G(\mathbf{w}). \quad (\text{A.25})$$

A.2 Derivation of Generalization Error Estimates

When a model has been estimated w.r.t. to some cost function `gend:trae` (i.e., the weights $\hat{\mathbf{w}}$ has been estimated), the quality of the model is determined as the generalization error, $G(\hat{\mathbf{w}})$, which according to `gend:gentrue`, (A.4) is given by

$$\begin{aligned} G(\hat{\mathbf{w}}) &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ e_t^2(\hat{\mathbf{w}}) \right\} \\ &= E_{\mathbf{x}_t, \varepsilon_t} \left\{ (\varepsilon_t + g(\mathbf{x}_t) - f(\mathbf{x}_t; \hat{\mathbf{w}}))^2 \right\}. \end{aligned} \quad (\text{A.26})$$

In order to evaluate the expectation in this expression we have to know the system $g(\mathbf{x}_t)$ and the joint p.d.f. of $[\mathbf{x}_t, \varepsilon_t]$. However, these claims are not met in general. The only knowledge of the actual system is obtained implicitly from the acquired training data. For that reason we derive generalization error estimates which are based on training data solely.

The generalization error estimator is denoted *GEN* (**G**eneralization **e**rror estimator for incomplete, **n**onlinear models, see also App. I) and defined by:

⁹The term “generalization error” may seem misleading since it in fact is the expectation of the *squared* error signal. However, the nomenclature is in keeping with literature on this subject and furthermore, the attached substantive “generalization” should prevent any misunderstanding.

DEFINITION A.6 *Define the average generalization error:*

$$\Gamma = E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\} \quad (\text{A.27})$$

where \mathcal{T} is the training set.¹⁰ Provided that As. A.1 or As. A.2 and As. A.3 through As. A.9 hold then *GEN* is defined as an consistent ($N \rightarrow \infty$) estimator of the average generalization error based on a second order Taylor series expansion of the cost function cf. As. A.11 and As. A.13 below.

Notice three facts:

- Expectation w.r.t. to the training set, i.e., the samples $\{\mathbf{x}(k); y(k)\}$, and expectation w.r.t. to the samples $\{\mathbf{x}(k); \varepsilon(k)\}$ are identical according to `gend:nonsys`.
- $G(\hat{\mathbf{w}})$ depends on the training set via $\hat{\mathbf{w}}$.
- *GEN* does not estimate the generalization error of the actual training set; on the contrary, the average generalization error w.r.t. training sets of size N .

The derivation of *GEN* depends on the chosen cost function. In this thesis we consider the following cost functions:

1. The usual LS cost function, i.e., cf. `gend:trae`, (A.8)

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N (y(k) - f(\mathbf{x}(k); \mathbf{w}))^2. \quad (\text{A.28})$$

2. The LS cost function with a regularization term which is independent of the training data, i.e.,

$$\begin{aligned} C_N(\mathbf{w}) &= S_N(\mathbf{w}) + \kappa R_N(\mathbf{w}) \\ &= S_N(\mathbf{w}) + \kappa r(\mathbf{w}) \end{aligned} \quad (\text{A.29})$$

where $\kappa \geq 0$ is the regularization parameter. Define: $r(\mathbf{w}) = R_N(\mathbf{w})$ where $r(\cdot)$ is assumed to be an arbitrary two times differentiable function which depends on the weight vector only. Note that if $\kappa = 0$ the cost passes into the usual LS cost.

The derivation is for the sake of retaining the leitmotif split up into two parts dealing with the above cost functions.

A.2.1 LS Cost Function

When applying the LS cost, i.e.,

$$C_N(\mathbf{w}) = S_N(\mathbf{w}), \quad (\text{A.30})$$

¹⁰Writing out the expectation

$$\Gamma = \int \cdots \int G(\hat{\mathbf{w}}(\{\mathbf{x}(k); y(k)\})) \cdot p(\mathbf{x}(1); y(1), \mathbf{x}(2); y(2), \cdots, \mathbf{x}(N); y(N)) \\ d\mathbf{x}(1)dy(1)d\mathbf{x}(2)dy(2) \cdots d\mathbf{x}(N)dy(N)$$

gend:excost and (A.22) gives that the expected cost is equal to the generalization error as $r_t \equiv 0$, i.e.,

$$C(\mathbf{w}) = G(\mathbf{w}). \quad (\text{A.31})$$

We make the following definitions:

DEFINITION A.7

1. *The instantaneous gradient vector of the output:*

$$\boldsymbol{\psi}(k; \mathbf{w}) = \frac{\partial f(\mathbf{x}(k); \mathbf{w})}{\partial \mathbf{w}} \quad (\text{A.32})$$

which is assumed to exist $\forall \mathbf{w} \in \Omega$.

2. *The instantaneous second order derivative matrix of the output:*

$$\boldsymbol{\Psi}(k; \mathbf{w}) = \frac{\partial \boldsymbol{\psi}(k; \mathbf{w})}{\partial \mathbf{w}^\top} \quad (\text{A.33})$$

which is assumed to exist $\forall \mathbf{w} \in \Omega$. Note that $\boldsymbol{\Psi}$ is a symmetric matrix.

3. *The gradient vector of the cost function:*

$$\begin{aligned} \frac{\partial S_N(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{N} \sum_{k=1}^N \frac{\partial e^2(k; \mathbf{w})}{\partial \mathbf{w}} \\ &= \frac{2}{N} \sum_{k=1}^N \frac{\partial e(k; \mathbf{w})}{\partial \mathbf{w}} e(k; \mathbf{w}) \\ &= -\frac{2}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}) e(k; \mathbf{w}). \end{aligned} \quad (\text{A.34})$$

4. *Similarly the gradient vector of the expected cost function:*

$$\frac{\partial G(\mathbf{w})}{\partial \mathbf{w}} = -2E_{\mathbf{x}_t, \varepsilon_t} \{ \boldsymbol{\psi}_t(\mathbf{w}) e_t(\mathbf{w}) \}. \quad (\text{A.35})$$

5. *The Hessian matrix of the cost function:*

$$\begin{aligned} \mathbf{H}_N(\mathbf{w}) &= \frac{1}{2} \frac{\partial^2 S_N(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \\ &= \frac{\partial}{\partial \mathbf{w}^\top} \left(\frac{1}{2} \frac{\partial S_N(\mathbf{w})}{\partial \mathbf{w}} \right) \\ &= -\frac{1}{N} \sum_{k=1}^N \frac{\partial}{\partial \mathbf{w}^\top} (\boldsymbol{\psi}(k; \mathbf{w}) e(k; \mathbf{w})) \\ &= \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}) \boldsymbol{\psi}^\top(k; \mathbf{w}) - \boldsymbol{\Psi}(k; \mathbf{w}) e(k; \mathbf{w}) \end{aligned} \quad (\text{A.36})$$

which is nonsingular at $\hat{\mathbf{w}}$ by As. A.7. Note that the Hessian is symmetric.

6. Similarly the Hessian matrix of the expected cost function (generalization error):

$$\mathbf{H}(\mathbf{w}) = \frac{1}{2} \frac{\partial^2 G(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} = E_{\mathbf{x}_t, \varepsilon_t} \left\{ \boldsymbol{\psi}_t(\mathbf{w}) \boldsymbol{\psi}_t^\top(\mathbf{w}) - \boldsymbol{\Psi}_t(\mathbf{w}) e_t(\mathbf{w}) \right\} \quad (\text{A.37})$$

which is nonsingular at \mathbf{w}^* by As. A.9.

The recipe for deriving the *GEN*-estimate is a second order Taylor series expansion of the cost function which formally is expressed by the following assumption:

ASSUMPTION A.11 *Let the minimization of S_N on the training set \mathcal{T} result in the estimate: $\hat{\mathbf{w}}^{11}$. Assume the existence of an optimal weight vector \mathbf{w}^* such that the remainders of the second order Taylor series expansion of G around \mathbf{w}^* are negligible. That is: Let $\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$ then*

$$G(\hat{\mathbf{w}}) \approx G(\mathbf{w}^*) + \Delta \mathbf{w}^\top \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w}, \quad (\text{A.38})$$

as $\partial G(\mathbf{w}^*) / \partial \mathbf{w} = \mathbf{0}$ according to As. A.9. Further, let Ξ denote the hypersphere with centre in \mathbf{w}^* in which the second order expansion is valid (w.r.t. to prescribed bounds on the higher order derivatives).

Further assume that the remainders of expanding S_N around $\hat{\mathbf{w}}$ to the second order is negligible, i.e.,

$$S_N(\mathbf{w}^*) \approx S_N(\hat{\mathbf{w}}) + \Delta \mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w}, \quad (\text{A.39})$$

as $\partial S_N(\hat{\mathbf{w}}) / \partial \mathbf{w} = \mathbf{0}$ according to As. A.7.

Note that this assumption is trivially met when dealing with LX-models. In Fig. A.3 an example of cost functions which fulfil As. A.11 is shown. To ensure the validity of the *GEN*-estimate we make the following additional assumption:

ASSUMPTION A.12 *We assume large training sets, $N \rightarrow \infty$ and $N \gg 2M + 1$ where M is the dependence lag defined by As. A.3. Further we assume that the dimension of the weight vector, m , is finite.*

Below we list six theorems stating estimates of the generalization error. The theorems differ by various assumptions concerning the model, input and noise. These assumptions are summarized in Table A.1. Furthermore, we include an additional theorem concerning the *GEN*-estimate when dealing with complete, LX-models. In Sec. A.2.1.1 the proofs are given.

THEOREM A.2 *Consider a nonlinear system and a model given by `gend:nonsys` and `gend:model`, respectively. Suppose that the model is an NN-model which is either **incomplete** (see Def. A.1) or alternatively **complete** with the restriction that \mathbf{w}^* (defined in As. A.11) is not the global optimum of $G(\mathbf{w})$. Further, suppose that As. A.1, A.3 – A.9, and A.11 – A.12 hold. The *GEN*-estimate is then given by:*

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N - \tau}{N} \left(\mathbf{R}(\tau) + \mathbf{R}^\top(\tau) \right) \right) \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (\text{A.40})$$

¹¹Note that the weight estimate is highly dependent on the chosen weight estimation algorithm due to local optimization, initial conditions, etc. (see further Ch. 5). An alternative algorithm used on the same training set may therefore result in a different weight estimate.

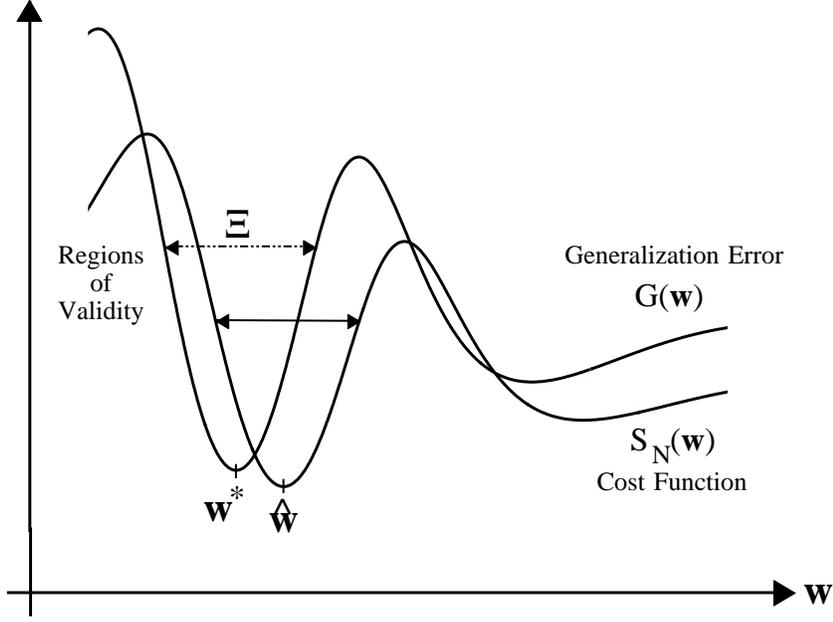


Figure A.3: Example of cost function $S_N(\mathbf{w})$ and generalization error (expected cost) $G(\mathbf{w})$ which fulfil As. A.11, q.e., that a proper expansion of both $S_N(\mathbf{w})$ and $G(\mathbf{w})$ around their respective minima to the second order is possible. The regions of validity (hyperspheres) for the second order expansion are shown. In particular, the region of validity around \mathbf{w}^* is denoted by Ξ .

where $\text{tr}[\cdot]$ denotes the trace and the correlation matrices $\mathbf{R}(\tau)$, $0 \leq \tau \leq M$ are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}). \quad (\text{A.41})$$

THEOREM A.3 Suppose that the nonlinear system is given by *gend:nonsys* and the model given by *gend:model* is an **incomplete** (see Def. A.1) LX-model cf. *gend:linmod*. Further, suppose that As. A.1, A.3 – A.5, A.7, A.9, and A.12 hold then the GEN-estimate is:

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1} \right] \quad (\text{A.42})$$

where the correlation matrices $\mathbf{R}(\tau)$, $0 \leq \tau \leq M$ are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \mathbf{z}(k) e(k; \hat{\mathbf{w}}) \mathbf{z}^\top(k + \tau) e(k + \tau; \hat{\mathbf{w}}), \quad (\text{A.43})$$

and

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k). \quad (\text{A.44})$$

where $\mathbf{z}(k)$ is defined by *gend:linmod*.

Model \ Input		Independent	Dependent
		Incomplete	NN-model
LX-model	Theorem A.5		Theorem A.3
Complete	NN-model	Theorem A.6 (if global optimum is found)	
	LX-model	Theorem A.6 Theorem A.7 (if input is Gaussian)	

Table A.1: Table which shows the major conditions for employing the various theorems estimating the generalization error. Recall that the LX-model is linear in the parameters while the NN-model is nonlinear in the parameters.

THEOREM A.4 Consider a nonlinear system and model given by *gend:nonsys* and *gend:model*, respectively. Suppose that the model is either **incomplete** (see Def. A.1) or alternatively **complete** with the restriction that \mathbf{w}^* (defined in As. A.11) is not the global optimum of $G(\mathbf{w})$. Further, suppose that As. A.2 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. A.4 – A.9, and A.11 – A.12 are supposed to hold. The GEN-estimate is then given by:

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\mathbf{R}(0) \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (\text{A.45})$$

where $\mathbf{R}(0)$ cf. *gend:R* is:

$$\mathbf{R}(0) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) e^2(k; \hat{\mathbf{w}}). \quad (\text{A.46})$$

THEOREM A.5 Suppose that the nonlinear system is given by *gend:nonsys* and the model given by *gend:model* is an **incomplete** (see Def. A.1) LX-model cf. *gend:linmod*. Further, suppose that As. A.2 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. A.4, A.5, A.7, A.9, and A.12 are supposed to hold. The GEN-estimate then yields:

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) e^2(k; \hat{\mathbf{w}}) \right) \left(\sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \right)^{-1} \right] \quad (\text{A.47})$$

where $\mathbf{z}(k)$ is defined by *gend:linmod*.

THEOREM A.6 Suppose that the nonlinear system is given by `gend:nonsys` and the model given by `gend:model` is **complete** (see Def. A.1) and either an NN- or an LX-model. Further, suppose that As. A.2 – A.9, A.11, and A.12 hold and that \mathbf{w}^* defined in As. A.11 is the **global minimum**¹² of $G(\mathbf{w})$. Finally, let $E\{\varepsilon^2\} = \sigma_\varepsilon^2 \neq 0$. The GEN-estimate then coincides with the FPE-criterion [Akaike 69]:

$$GEN = FPE = \frac{N + m}{N - m} S_N(\hat{\mathbf{w}}), \quad N > m. \quad (\text{A.48})$$

THEOREM A.7 Suppose that the system given by `gend:nonsys` and the model `gend:model` is a **complete** (see Def. A.1) LX-model cf. `gend:linmod`. Further, suppose that As. A.2, A.4, A.5, A.7, and A.9 hold, the input vector is marginally Gaussian distributed with zero mean and positive definite covariance matrix \mathbf{H} , i.e., $\mathbf{z}(k) \in \mathcal{N}(\mathbf{0}, \mathbf{H})$, $\forall k$, and that $\mathbf{z}(k_1)$ is independent of $\mathbf{z}(k_2)$ as $k_1 \neq k_2$. Finally, let $E\{\varepsilon^2\} = \sigma_\varepsilon^2 \neq 0$. The GEN-estimate coincides with [Hansen 93] and is given by:

$$GEN = \frac{N(N - 1)}{(N - m)(N - m - 1)} S_N(\hat{\mathbf{w}}), \quad N > m + 1. \quad (\text{A.49})$$

Additional Theorem Concerning Complete LX-Models

THEOREM A.8 Suppose that the system given by `gend:nonsys` and the model `gend:model` is a **complete** (see Def. A.1) LX-model cf. `gend:linmod`. Furthermore, suppose that As. A.2 – A.5, A.7, A.9 hold. The average generalization error, Γ , complies with the following ranking:

$$\Gamma > \frac{N + m}{N - m} E_{\mathcal{T}}\{S_N(\hat{\mathbf{w}})\} > \left(1 + 2\frac{m}{N}\right) E_{\mathcal{T}}\{S_N(\hat{\mathbf{w}})\}. \quad (\text{A.50})$$

From this theorem it is clear that on the average the FPE-estimate given in Th. A.6 (i.e., $E_{\mathcal{T}}\{FPE\}$) is an underestimate of the average generalization error, Γ . On the other hand, on the average the FPE-estimate is larger than the estimate, $(1 + 2m/N)S_N(\hat{\mathbf{w}})$, i.e., closer to Γ . This estimator is known as the *predicted squared error* (PSE) estimator [Barron 84].

A.2.1.1 Proof of Theorems

In this section we give the proofs of Th. A.2 – A.8. Taking the expectation w.r.t. the training set, \mathcal{T} , of the Taylor series expansions in `gend:Gtaylor`, (A.39) we get:

$$E_{\mathcal{T}}\{S_N(\mathbf{w}^*)\} \approx E_{\mathcal{T}}\{S_N(\hat{\mathbf{w}})\} + E_{\mathcal{T}}\{\Delta\mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}})\Delta\mathbf{w}\} \quad (\text{A.51})$$

$$E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\} \approx E_{\mathcal{T}}\{G(\mathbf{w}^*)\} + E_{\mathcal{T}}\{\Delta\mathbf{w}^\top \mathbf{H}(\mathbf{w}^*)\Delta\mathbf{w}\}. \quad (\text{A.52})$$

Recall that $\Gamma = E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\}$ and $\Delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$.

Comments:

- When dealing with an LX-model, which is assumed in Th. A.3, A.5, and A.7, the second order Taylor series expansions are *exact* (cf. Ch. 5).¹³

¹²Note the possibility of more coexisting global minima in which the expected cost (i.e., $G(\mathbf{w}^*)$) reaches the same level. Further note that the requirement is trivially fulfilled when dealing with LX-models.

¹³In what follows all approximate symbols, \approx , in series expansions are exact *only* when considering LX-models.

- Recall that the weight estimate, $\hat{\mathbf{w}}$ obtained by training on the present training set defines an associated \mathbf{w}^* through the required validity of the Taylor series expansion concerning $S_N(\cdot)$ cf. `gend:Staylor`. Furthermore, recollect that Ξ is the hypersphere with centre \mathbf{w}^* and radius $\Delta\mathbf{w}$, i.e., the region in weight-space in which the expansion of $G(\cdot)$ is valid cf. `gend:Gtaylor`. Now consider all possible training sets of size N and the corresponding sets of minimizers, \mathcal{W} . In general only a *finite number of training sets* will result in sets, \mathcal{W} , which contain an estimate $\hat{\mathbf{w}}$ which is in Ξ . Consequently, the above averaging – formally considered – is done w.r.t. training sets which result in sets, \mathcal{W} , containing a $\hat{\mathbf{w}} \in \Xi$. However, $S_N \rightarrow G$ in the limit $N \rightarrow \infty$ (cf. page 327). Consequently, in this limit, every training set contains a $\hat{\mathbf{w}}$ which is in Ξ .

As \mathbf{w}^* does not depend on the training set `gend:sn` and (A.22) gives

$$E_{\mathcal{T}} \{S_N(\mathbf{w}^*)\} = \frac{1}{N} \sum_{k=1}^N E_{\mathcal{T}} \{e^2(k; \mathbf{w}^*)\} = G(\mathbf{w}^*) = E_{\mathcal{T}} \{G(\mathbf{w}^*)\}. \quad (\text{A.53})$$

Applying this equality allow us to substitute `gend:es` into `gend:eg`). Accordingly,

$$\Gamma \approx E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + E_{\mathcal{T}} \left\{ \Delta\mathbf{w}^{\top} \mathbf{H}_N(\hat{\mathbf{w}}) \Delta\mathbf{w} \right\} + E_{\mathcal{T}} \left\{ \Delta\mathbf{w}^{\top} \mathbf{H}(\mathbf{w}^*) \Delta\mathbf{w} \right\}. \quad (\text{A.54})$$

Next we estimate the three addends in `gend:lam`. As only one training set is available the first addend is estimated by

$$E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} \approx S_N(\hat{\mathbf{w}}). \quad (\text{A.55})$$

On the other hand, it is possible to artificially create a number of training sets with size $N_0 < N$ by successively drawing N_0 examples of the N possible. Hence, averaging the cost function obtained by training on these sets may result in a better estimate of $E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\}$. The averaging should; however, be done carefully in order to ensure that the estimates $\hat{\mathbf{w}}$ (gained on the training sets of size N_0) all lies within the hypersphere Ξ . This is eventually very time consuming, subsidiary impossible, unless it is assumed that the global minimum of all training sets are found and that these minima are “close” (i.e., within a second order expansion).

Next, the two last addends in `gend:lam` are evaluated. For that purpose an approximate expression for $\Delta\mathbf{w}$ is derived. Since $\hat{\mathbf{w}}$ minimizes $S_N(\mathbf{w})$ we have:

$$\frac{\partial S_N(\hat{\mathbf{w}})}{\partial \mathbf{w}} = \mathbf{0}. \quad (\text{A.56})$$

Applying the mean value theorem we get:

$$\frac{\partial S_N(\mathbf{w}^*)}{\partial \mathbf{w}} + \frac{\partial^2 S_N(\tilde{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^{\top}} \Delta\mathbf{w} = \mathbf{0} \quad (\text{A.57})$$

where

$$\tilde{\mathbf{w}} = \theta \mathbf{w}^* + (1 - \theta) \hat{\mathbf{w}}, \quad 0 < \theta < 1. \quad (\text{A.58})$$

The terms of order greater than two in the series expansion `gend:Staylor` is by assumption insignificant when the perturbations \mathbf{w} lie in the hypersphere with centre $\hat{\mathbf{w}}$ and radius $\Delta\mathbf{w}$. That is,

$$\frac{\partial^2 S_N(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^{\top}} \approx \frac{\partial^2 S_N(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^{\top}}. \quad (\text{A.59})$$

for all \mathbf{w} lying in the hypersphere. In particular, $\tilde{\mathbf{w}}$ lies in this hypersphere.
 gend:dwfirst then becomes:

$$\begin{aligned}
 \Delta\mathbf{w} &\approx - \left[\frac{\partial^2 S_N(\mathbf{w}^*)}{\partial\mathbf{w}\partial\mathbf{w}^\top} \right]^{-1} \frac{\partial S_N(\mathbf{w}^*)}{\partial\mathbf{w}} \\
 &= - \left[\frac{1}{2} \frac{\partial^2 S_N(\mathbf{w}^*)}{\partial\mathbf{w}\partial\mathbf{w}^\top} \right]^{-1} \frac{1}{2} \frac{\partial S_N(\mathbf{w}^*)}{\partial\mathbf{w}} \\
 &= \mathbf{H}_N^{-1}(\mathbf{w}^*) \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*)
 \end{aligned} \tag{A.60}$$

where the last equality is due to Def. A.7.

The second addend in gend:lam is now evaluated.

Using the following rule of matrix algebra:

$$\text{tr}[\mathbf{A}\mathbf{B}] = \text{tr}[\mathbf{B}\mathbf{A}] \tag{A.61}$$

where \mathbf{A} is an arbitrary $p \times q$ matrix, \mathbf{B} is an arbitrary $q \times p$ matrix, and $\text{tr}[\cdot]$ denotes the trace. Accordingly:

$$E_{\mathcal{T}} \left\{ \Delta\mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta\mathbf{w} \right\} = E_{\mathcal{T}} \left\{ \text{tr} \left[\mathbf{H}_N(\hat{\mathbf{w}}) \Delta\mathbf{w} \Delta\mathbf{w}^\top \right] \right\}. \tag{A.62}$$

Applying gend:dw and the fact that the Hessian is symmetric:

$$\begin{aligned}
 E_{\mathcal{T}} \left\{ \Delta\mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta\mathbf{w} \right\} &\approx \\
 E_{\mathcal{T}} \left\{ \text{tr} \left[\mathbf{H}_N(\hat{\mathbf{w}}) \mathbf{H}_N^{-1}(\mathbf{w}^*) \cdot \right. \right. \\
 \left. \left. \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \mathbf{H}_N^{-1}(\mathbf{w}^*) \right] \right\}, &\tag{A.63}
 \end{aligned}$$

and employing gend:sdevapprox) with $\mathbf{w} = \hat{\mathbf{w}}$ yields:

$$\begin{aligned}
 E_{\mathcal{T}} \left\{ \Delta\mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta\mathbf{w} \right\} &\approx \\
 E_{\mathcal{T}} \left\{ \text{tr} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \mathbf{H}_N^{-1}(\mathbf{w}^*) \right] \right\}. &\tag{A.64}
 \end{aligned}$$

Similarly, the third addend of gend:lam becomes:

$$\begin{aligned}
 E_{\mathcal{T}} \left\{ \Delta\mathbf{w}^\top \mathbf{H}(\mathbf{w}^*) \Delta\mathbf{w} \right\} &\approx \\
 E_{\mathcal{T}} \left\{ \text{tr} \left[\mathbf{H}(\mathbf{w}^*) \mathbf{H}_N^{-1}(\mathbf{w}^*) \cdot \right. \right. \\
 \left. \left. \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \mathbf{H}_N^{-1}(\mathbf{w}^*) \right] \right\}. &\tag{A.65}
 \end{aligned}$$

Complete, LX-Models The most plain case to treat is complete, LX-models. In addition, if the input is Gaussian distributed a very simple result is obtained, cf. Th. A.7 and [Hansen 93]. In this paragraph it is provided that the assumptions of Th. A.7 hold.

Since the model is complete and we assume that the optimal weights \mathbf{w}^* defined in As. A.11 is the global minimum of $G(\mathbf{w})$ ¹⁴

$$e(k; \mathbf{w}^*) = \varepsilon(k). \quad (\text{A.66})$$

To show this $G(\mathbf{w})$ is evaluated. Applying the error decomposition `gend:error`, the definition in `gend:gentrue`, and Def. A.1¹⁵ yield:

$$\begin{aligned} G(\mathbf{w}) &= E \left\{ [\varepsilon_t + f(\mathbf{x}_t; \mathbf{w}^\circ) - f(\mathbf{x}_t; \mathbf{w})]^2 \right\} \\ &= E \left\{ \varepsilon_t^2 \right\} + E \left\{ [f(\mathbf{x}_t; \mathbf{w}^\circ) - f(\mathbf{x}_t; \mathbf{w})]^2 \right\} \\ &\quad + 2E \left\{ (f(\mathbf{x}_t; \mathbf{w}^\circ) - f(\mathbf{x}_t; \mathbf{w})) \varepsilon_t \right\} \\ &= E \left\{ \varepsilon_t^2 \right\} + E \left\{ [f(\mathbf{x}_t; \mathbf{w}^\circ) - f(\mathbf{x}_t; \mathbf{w})]^2 \right\} \end{aligned} \quad (\text{A.67})$$

where \mathbf{w}° is the true weights and the independence of \mathbf{x}_t and ε_t is used to obtain the last equality. Now, minimization of `gend:gcom` gives:

$$\mathbf{w}^\circ = \mathbf{w}^* = \arg \min_{\mathbf{w}} G(\mathbf{w}), \quad (\text{A.68})$$

and

$$G(\mathbf{w}^*) = E\{\varepsilon_t^2\} = \sigma_\varepsilon^2. \quad (\text{A.69})$$

Hence, the adjustment error $f(\mathbf{x}(k); \mathbf{w}^\circ) - f(\mathbf{x}(k); \mathbf{w}) \equiv 0$ and in consequence the error $e(k; \mathbf{w}^*)$ equals the inherent error $\varepsilon(k)$.

According to Def. A.7 and `gend:linmod`

$$\boldsymbol{\psi}(k; \mathbf{w}) = \frac{\partial f(\mathbf{x}(k); \mathbf{w})}{\partial \mathbf{w}} = \mathbf{z}(k) \quad (\text{A.70})$$

$$\mathbf{H}_N(\mathbf{w}) = \mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \quad (\text{A.71})$$

$$\mathbf{H}(\mathbf{w}) = \mathbf{H} = E_{\mathbf{z}_t} \left\{ \mathbf{z}_t \mathbf{z}_t^\top \right\}. \quad (\text{A.72})$$

The second addend in `gend:es` is thus by the above and `gend:secterm`:

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} = \text{tr} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N E_{\mathcal{T}} \left\{ \mathbf{z}(k_1) \mathbf{z}^\top(k_2) \varepsilon(k_1) \varepsilon(k_2) \mathbf{H}_N^{-1} \right\} \right]. \quad (\text{A.73})$$

The expectation w.r.t. \mathcal{T} is carried out by noting that¹⁶

$$E_{\mathcal{T}} \{ \cdot \} = E_{\{\mathbf{z}(k), \varepsilon(k)\}} \{ \cdot \} = E_{\{\mathbf{z}(k)\}} \left\{ E_{\{\varepsilon(k)\}} \{ \cdot | \{\mathbf{z}(k)\} \} \right\}. \quad (\text{A.74})$$

¹⁴Within LX-models this assumption is trivial as only one global minimum exists.

¹⁵Subscript, $[\mathbf{x}_t, \varepsilon_t]$, of the expectation operator is omitted.

¹⁶Recall that $\mathbf{x}(k)$ is mapped into $\mathbf{z}(k)$ cf. `gend:linmod`.

As $\varepsilon(k)$ is a (second order) white sequence the expectation w.r.t. $\varepsilon(k)$ contributes only when $k_1 = k_2$. Setting: $E\{\varepsilon^2(k)\} = \sigma_\varepsilon^2$ gend:seclin transforms into

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} &= \frac{\sigma_\varepsilon^2}{N} \cdot \text{tr} \left[E_{\mathbf{z}(k)} \left\{ \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \mathbf{H}_N^{-1} \right\} \right] \\ &= \frac{m}{N} \sigma_\varepsilon^2 \end{aligned} \quad (\text{A.75})$$

as the term in the expectation square brackets becomes the $m \times m$ identity matrix.

Analogously the second addend of gend:eg is by gend:thterm:

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \right\} &= \text{tr} \left[E_{\mathcal{T}} \left\{ \mathbf{H} \mathbf{H}_N^{-1} \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \mathbf{z}(k_1) \mathbf{z}^\top(k_2) \varepsilon(k_1) \varepsilon(k_2) \mathbf{H}_N^{-1} \right\} \right] \\ &= \frac{\sigma_\varepsilon^2}{N} \cdot \text{tr} \left[\mathbf{H} \cdot E_{\mathbf{z}(k)} \left\{ \mathbf{H}_N^{-1} \right\} \right]. \end{aligned} \quad (\text{A.76})$$

In general the distribution of $\mathbf{z}(k)$ is required in order to calculate $E\{\mathbf{H}_N^{-1}\}$. Otherwise, we may resort to approximate results.

Consider the case where the input is Gaussian distributed independent sequence, i.e., $\mathbf{z}(k) \in \mathcal{N}(\mathbf{0}, \mathbf{H})$. The result of Sec. B.2 (see further [Anderson 84]) is applicable and gives:

$$E \left\{ \mathbf{H}_N^{-1} \right\} = \frac{N}{N - m - 1} \mathbf{H}^{-1}, \quad N > m + 1. \quad (\text{A.77})$$

Thus gend:thlin transforms into:

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \right\} = \frac{\sigma_\varepsilon^2}{N} \text{tr} \left[\frac{N}{N - m - 1} \cdot \mathbf{I} \right] = \frac{m}{N - m - 1} \sigma_\varepsilon^2. \quad (\text{A.78})$$

Substituting gend:gtruelin, (A.75) and (A.78) into gend:es and (A.52) they yield:

$$\sigma_\varepsilon^2 = E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + \frac{m}{N} \sigma_\varepsilon^2 \quad (\text{A.79})$$

$$\Gamma = \sigma_\varepsilon^2 + \frac{m}{N - m - 1} \sigma_\varepsilon^2. \quad (\text{A.80})$$

Eliminating σ_ε^2 (presupposed that $\sigma_\varepsilon^2 \neq 0$):

$$\Gamma = \frac{N(N - 1)}{(N - m)(N - m - 1)} E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\}, \quad N > m + 1. \quad (\text{A.81})$$

Using the approximation gend:snapprox the estimate of Γ becomes:

$$GEN = \frac{N(N - 1)}{(N - m)(N - m - 1)} S_N(\hat{\mathbf{w}}), \quad N > m - 1. \quad (\text{A.82})$$

This proves the result of Th. A.7.

Note that this estimator is unbiased as

$$E_{\mathcal{T}} \{GEN\} = \frac{N(N - 1)}{(N - m)(N - m - 1)} E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} = \Gamma. \quad (\text{A.83})$$

If no information concerning the input distribution is available the approximate result, Th. B.4, may be used. This theorem states for $(2M + 1)/N \rightarrow 0$:

$$E \left\{ \mathbf{H}_N^{-1} \right\} = \mathbf{H}^{-1} + \epsilon \left(\frac{2M + 1}{N} \right) \quad (\text{A.84})$$

where M is the dependence lag of the input, see As. A.3.

Substituting this approximation into `gend:thlin` we get:

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \right\} &= \frac{\sigma_{\epsilon}^2}{N} \cdot \text{tr} \left[\mathbf{H} \mathbf{H}^{-1} \right] + o \left(\frac{2M + 1}{N} \right) \\ &\approx \frac{m}{N} \sigma_{\epsilon}^2, \quad \frac{2M + 1}{N} \rightarrow 0. \end{aligned} \quad (\text{A.85})$$

The analogous to `gend:gene`) thus becomes:

$$\Gamma \approx \left(1 + \frac{m}{N} \right) \sigma_{\epsilon}^2. \quad (\text{A.86})$$

`gend:tra` still holds and σ_{ϵ}^2 is estimated by:

$$\sigma_{\epsilon}^2 = \frac{E_{\mathcal{T}} \{ S_N(\hat{\mathbf{w}}) \}}{1 - \frac{m}{N}}, \quad N > m. \quad (\text{A.87})$$

Inserting this in `gend:gencom` yields:

$$\Gamma = \frac{1 + \frac{m}{N}}{1 - \frac{m}{N}} E_{\mathcal{T}} \{ S_N(\hat{\mathbf{w}}) \}, \quad N > m, \quad \frac{2M + 1}{N} \rightarrow 0 \quad (\text{A.88})$$

and using `gend:snapprox` result in:

$$GEN = \frac{1 + \frac{m}{N}}{1 - \frac{m}{N}} S_N(\hat{\mathbf{w}}), \quad N > m, \quad \frac{2M + 1}{N} \rightarrow 0. \quad (\text{A.89})$$

GEN thus coincides with the *FPE*-criterion [Akaike 69]. Observe that the first factor induces terms which tend to zero faster than $(2M + 1)/N$; thus it may seem inconsistent not to neglect these terms. However, below it is shown that GEN is better approximated by keeping the terms. This is the content of Th. A.8.

The proof falls into two parts. First we show that Γ in `gend:gamcomlin` is “under estimated”. Secondly we show that neglectation of higher order terms imply a smaller estimate.

Focus on `gend:thlin` which cf. `gend:thlinapprox` is approximated by $m\sigma_{\epsilon}^2/N$ when using the approximation of Th. B.4. It is next showed that `gend:thlinapprox` is a lower bound on the term in `gend:thlin`, i.e.,

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \right\} = \frac{\sigma_{\epsilon}^2}{N} \cdot \text{tr} \left[\mathbf{H} \cdot E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} \right] > \frac{\sigma_{\epsilon}^2}{N} \cdot \text{tr} \left[\mathbf{H} \mathbf{H}^{-1} \right]. \quad (\text{A.90})$$

Obviously it suffices to show:

$$\text{tr} \left[\mathbf{H} \cdot E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} \right] > \text{tr} \left[\mathbf{H} \mathbf{H}^{-1} \right]. \quad (\text{A.91})$$

Since \mathbf{H} is positive definite it is possible to perform a Cholesky decomposition, $\mathbf{H} = \mathbf{Q} \mathbf{Q}^{\top}$, where \mathbf{Q} is defined as the $m \times m$ matrix: $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m]$. `gend:ineq1` gives

$$\text{tr} \left[\mathbf{Q} \mathbf{Q}^{\top} \cdot E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} \right] > \text{tr} \left[\mathbf{Q} \mathbf{Q}^{\top} \mathbf{H}^{-1} \right] \quad (\text{A.92})$$

⇕ (Using the rule: $\text{tr}[\mathbf{AB}] = \text{tr}[\mathbf{BA}]$)

$$\text{tr} \left[\mathbf{Q}^\top \cdot E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} \mathbf{Q} \right] > \text{tr} \left[\mathbf{Q}^\top \mathbf{H}^{-1} \mathbf{Q} \right] \quad (\text{A.93})$$

⇕

$$\sum_{i=1}^m \mathbf{q}_i^\top E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} \mathbf{q}_i > \sum_{i=1}^m \mathbf{q}_i^\top \mathbf{H}^{-1} \mathbf{q}_i. \quad (\text{A.94})$$

Applying Th. B.5 which states:

$$E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} > \mathbf{H}^{-1} \Leftrightarrow \mathbf{a}^\top \left(E_{\mathbf{x}(k)} \left\{ \mathbf{H}_N^{-1} \right\} - \mathbf{H}^{-1} \right) \mathbf{a} > 0 \quad \forall \mathbf{a} \neq \mathbf{0}, \quad (\text{A.95})$$

then the inequality `gend:ineq1` is proved. In summary:

$$\Gamma > \frac{N+m}{N-m} E_T \{ S_N(\hat{\mathbf{w}}) \}, \quad N > m. \quad (\text{A.96})$$

Secondly, applying the Taylor series expansion

$$\frac{1}{1 - \frac{m}{N}} = 1 + \frac{m}{N} - \left(\frac{m}{N} \right)^2 + \dots, \quad (\text{A.97})$$

the first factor of `gend:gencomlin` is expanded by:

$$\frac{N+m}{N-m} = \frac{1 + \frac{m}{N}}{1 - \frac{m}{N}} = 1 + 2\frac{m}{N} + o(N^{-1}). \quad (\text{A.98})$$

The remainder of this expansion is always positive, i.e.,

$$\frac{N+m}{N-m} > 1 + 2\frac{m}{N}. \quad (\text{A.99})$$

This is proved by:

$$\begin{aligned} \frac{N+m}{N-m} &> 1 + 2\frac{m}{N} \\ N+m &> N+2m-m-2\frac{m^2}{N} \\ 0 &> -2\frac{m^2}{N}. \end{aligned}$$

The final result is the ranking:

$$\Gamma > \frac{N+m}{N-m} E_T \{ S_N(\hat{\mathbf{w}}) \} > \left(1 + 2\frac{m}{N} \right) E_T \{ S_N(\hat{\mathbf{w}}) \}. \quad (\text{A.100})$$

This ends the proof of Th. A.8.

Complete NN-Models If the model is complete, As. A.2 holds, and the optimal weights \mathbf{w}^* defined in As. A.11 correspond to the global minimum of $G(\mathbf{w})$ (recall, that more equally good minima may exist) then, as shown in the preceding paragraph:

$$e(k; \mathbf{w}^*) = \varepsilon(k). \quad (\text{A.101})$$

The inverse Hessian, $\mathbf{H}_N^{-1}(\mathbf{w}^*)$, is approximated by using the result of Corollary B.1, i.e.,

$$\mathbf{H}_N^{-1}(\mathbf{w}^*) = \left(\mathbf{I} + \sum_{i=1}^{\infty} (-1)^i \left(\mathbf{H}^{-1}(\mathbf{w}^*) \Theta(\mathbf{w}^*) \right)^i \right) \mathbf{H}^{-1}(\mathbf{w}^*) \quad (\text{A.102})$$

where

$$\Theta(\mathbf{w}^*) = \mathbf{H}_N(\mathbf{w}^*) - \mathbf{H}(\mathbf{w}^*) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\theta}(k). \quad (\text{A.103})$$

The last equality appears per definition as \mathbf{H}_N is a sum over N examples. Substituting into `gend:hnxp` and reorganizing gives:

$$\begin{aligned} \mathbf{H}_N^{-1}(\mathbf{w}^*) = & \\ & \mathbf{H}^{-1}(\mathbf{w}^*) - \frac{1}{N} \sum_{k=1}^N \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k) \mathbf{H}^{-1}(\mathbf{w}^*) + \\ & \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_1) \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_2) \mathbf{H}^{-1}(\mathbf{w}^*) + \dots \end{aligned} \quad (\text{A.104})$$

With these facts in mind, and presuming that the assumptions of Th. A.6 hold, then the second addend in `gend:es` yields cf. `gend:secterm`:

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} \approx & \\ E_{\mathcal{T}} \left\{ \text{tr} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) \varepsilon(k_1) \boldsymbol{\psi}^{\top}(k_2; \mathbf{w}^*) \varepsilon(k_2) \mathbf{H}^{-1}(\mathbf{w}^*) \right] \right\} + & \\ E_{\mathcal{T}} \left\{ \text{tr} \left[\frac{1}{N^3} \sum_{k_1=1}^N \sum_{k_2=1}^N \sum_{k_3=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) \varepsilon(k_1) \boldsymbol{\psi}^{\top}(k_2; \mathbf{w}^*) \varepsilon(k_2) \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_3) \right. \right. & \\ \left. \left. \mathbf{H}^{-1}(\mathbf{w}^*) \right] \right\} + \dots \end{aligned} \quad (\text{A.105})$$

Since the input $\mathbf{x}(k)$ and the inherent noise $\varepsilon(k)$ are assumed to be strictly stationary sequences and in addition the means of the involved stochastic matrices are zero¹⁷ the results of App. C are applicable. That means, that from the second term (in `gend:seccom1`) and onward the terms tend to zero as: $((2M+1)/N)^q$, for $N \rightarrow \infty$, $q = 2, 3, \dots$. Assuming that $(2M+1)/N \rightarrow 0$ we neglect all terms which tend to zero faster than $(2M+1)/N$. Consequently, `gend:seccom1` becomes:

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} \approx & \\ E_{\mathcal{T}} \left\{ \text{tr} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) \varepsilon(k_1) \boldsymbol{\psi}^{\top}(k_2; \mathbf{w}^*) \varepsilon(k_2) \mathbf{H}^{-1}(\mathbf{w}^*) \right] \right\}. \end{aligned} \quad (\text{A.106})$$

¹⁷ Define, in accordance with App. C: $\mathbf{Y}_1(k_1) = \boldsymbol{\psi}(k_1; \mathbf{w}^*) \varepsilon(k_1)$, $\mathbf{Y}_2(k_2) = \boldsymbol{\psi}^{\top}(k_2; \mathbf{w}^*) \varepsilon(k_2)$ and $\mathbf{Y}_i(k_i) = \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_i)$, $i \geq 3$ which all have zero mean since

- $E\{\boldsymbol{\psi}(k; \mathbf{w}^*) \varepsilon(k)\}$ (which defines the (half of) gradient of the cost function) per definition is equal to the zero vector.
- $E\{\boldsymbol{\theta}(k_i)\} = \mathbf{0}$ according to `gend:thedef`

As \mathbf{H}^{-1} does not depend on the training set we proceed as in the preceding paragraph¹⁸. The result is:

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} \approx \frac{\sigma_{\varepsilon}^2}{N} \cdot \text{tr} \left[\frac{1}{N} \sum_{k=1}^N E_{\mathbf{x}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) \boldsymbol{\psi}^{\top}(k; \mathbf{w}^*) \right\} \mathbf{H}^{-1}(\mathbf{w}^*) \right]. \quad (\text{A.107})$$

From Def. A.7

$$\begin{aligned} \mathbf{H}^{-1}(\mathbf{w}^*) &= \left[E_{\mathbf{x}_t, \varepsilon_t} \left\{ \boldsymbol{\psi}_t(\mathbf{w}^*) \boldsymbol{\psi}_t^{\top}(\mathbf{w}^*) - \boldsymbol{\Psi}_t(\mathbf{w}^*) \varepsilon_t \right\} \right]^{-1} \\ &= \left[E_{\mathbf{x}_t} \left\{ \boldsymbol{\psi}_t(\mathbf{w}^*) \boldsymbol{\psi}_t^{\top}(\mathbf{w}^*) \right\} \right]^{-1}. \end{aligned} \quad (\text{A.108})$$

Finally¹⁹,

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} \approx \frac{m}{N} \sigma_{\varepsilon}^2. \quad (\text{A.109})$$

Next the second addend of `gend:eg` is evaluated by using `gend:thterm`, (A.101), and (A.104). As above we neglect terms which tend to zero faster than $(2M+1)/N$, i.e.,

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \right\} &\approx \\ \text{tr} \left[E_{\mathcal{T}} \left\{ \mathbf{H}(\mathbf{w}^*) \mathbf{H}^{-1}(\mathbf{w}^*) \cdot \right. \right. \\ &\left. \left. \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) \varepsilon(k_1) \boldsymbol{\psi}^{\top}(k_2; \mathbf{w}^*) \varepsilon(k_2) \mathbf{H}^{-1}(\mathbf{w}^*) \right\} \right]. \end{aligned} \quad (\text{A.110})$$

Reducing, performing the expectation, using `gend:invhstar` the matrix within the square brackets is the identity matrix times σ_{ε}^2/N . Accordingly:

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \right\} \approx \frac{m}{N} \sigma_{\varepsilon}^2. \quad (\text{A.111})$$

If `gend:seccomfn` and (A.111) is substituted into `gend:es` and (A.52) then

$$\sigma_{\varepsilon}^2 \approx E_{\mathcal{T}} \{ S_N(\hat{\mathbf{w}}) \} + \frac{m}{N} \sigma_{\varepsilon}^2 \quad (\text{A.112})$$

$$\Gamma \approx \sigma_{\varepsilon}^2 + \frac{m}{N} \sigma_{\varepsilon}^2. \quad (\text{A.113})$$

Estimating σ_{ε}^2 by `gend:sigfirst`, i.e.,

$$\sigma_{\varepsilon}^2 \approx \frac{E_{\mathcal{T}} \{ S_N(\hat{\mathbf{w}}) \}}{1 - \frac{m}{N}}, \quad N > m. \quad (\text{A.114})$$

Provided that $\sigma_{\varepsilon}^2 \neq 0$ substitution of this result into `gend:gammares`, and further using `gend:snapprox` gives:

$$GEN = \frac{N+m}{N-m} S_N(\hat{\mathbf{w}}). \quad (\text{A.115})$$

This result is equivalent with the *FPE*-estimate [Akaike 69] and ends the proof of Th. A.6. Clearly, the estimator is consistent for $N \rightarrow \infty$ as

$$\lim_{N \rightarrow \infty} \frac{N+m}{N-m} = 1, \quad \lim_{N \rightarrow \infty} S_N(\hat{\mathbf{w}}) = G(\mathbf{w}^*) \quad (\text{A.116})$$

¹⁸That is, first taking the expectation w.r.t. ε subsequently w.r.t. \mathbf{x} .

¹⁹Note, that if the model is an LX-model this result is not an approximation as the derivation in the preceding paragraph still holds. That is, it is not necessary to assume $N \rightarrow \infty$.

The right limit transition is obtained by Th. A.1 and by the fact that

$$\lim_{N \rightarrow \infty} S_N(\mathbf{w}) = G(\mathbf{w}), \quad \forall \mathbf{w}. \quad (\text{A.117})$$

By estimating σ_ε^2 as in `gend:sigapp` we introduce terms which tend to zero faster than $1/N$ as:

$$\frac{1}{1 - \frac{m}{N}} = 1 + \frac{m}{N} - \left(\frac{m}{N}\right)^2 + \dots. \quad (\text{A.118})$$

This is in principle inconsistent since we elsewhere have neglected terms of orders $1/N^q$, $q \geq 2$. If we consider the two first terms in `gend:mnapp` only then the estimate is:

$$GEN = \left(1 + 2\frac{m}{N}\right) S_N(\hat{\mathbf{w}}). \quad (\text{A.119})$$

Therefore in principle we should prefer this result. However, recall from the preceding paragraph that within LX-models the *FPE*-estimate is a better approximation than using `gend:gencomapp`. This leads to an argument for choosing the *FPE*-estimate instead of using the estimate in `gend:gencomapp`. This, of course, presumes that the ranking in `gend:rank` still holds within general complete NN-models. It has not been possible to show this; but it may be an interesting topic of future research.

Incomplete Models The essential change in the derivation when dealing with incomplete models originates from the fact that the optimal error, $e(k; \mathbf{w}^*)$, depends both on the input and the inherent noise. This is e.g., seen from `gend:error`:

$$e(k; \mathbf{w}^*) = g(\mathbf{x}(k)) - f(\mathbf{x}(k); \mathbf{w}^*) + \varepsilon(k). \quad (\text{A.120})$$

One inferior exception is the case where the adjustment error ($g - f$) equals a constant (no \mathbf{x} dependence). When the model contains a bias term (which is the common case, see Ch. 4) this case never occurs. Consequently, it is not possible to carry out the expectations in the terms `gend:secterm`, (A.65) w.r.t. to the input and the inherent noise individually, as done in previous paragraphs. It is interesting to notice, that even when the model is complete the optimal error may depend both on the input and the inherent noise. This occurs when the weight estimation algorithm fails to reach the global optimum of the cost function. See further the preceding paragraph concerning complete LX-models.

In order to evaluate the expectations in `gend:secterm` and (A.65) we expand the inverse Hessian (cf. `gend:invhnstar`):

$$\begin{aligned} \mathbf{H}_N^{-1}(\mathbf{w}^*) = & \\ & \mathbf{H}^{-1}(\mathbf{w}^*) - \frac{1}{N} \sum_{k=1}^N \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k) \mathbf{H}^{-1}(\mathbf{w}^*) + \\ & \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_1) \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_2) \mathbf{H}^{-1}(\mathbf{w}^*) + \dots. \end{aligned} \quad (\text{A.121})$$

Substituting these expansions into `gend:secterm` gives (by omitting the expectation and trace operators):

$$\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \mathbf{H}_N^{-1}(\mathbf{w}^*)$$

$$\begin{aligned}
&= \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \mathbf{H}^{-1}(\mathbf{w}^*) \\
&+ \frac{1}{N^3} \sum_{k_1=1}^N \sum_{k_2=1}^N \sum_{k_3=1}^N \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \mathbf{H}^{-1}(\mathbf{w}^*) \boldsymbol{\theta}(k_3) \mathbf{H}^{-1}(\mathbf{w}^*) \\
&+ \text{higher order terms.} \tag{A.122}
\end{aligned}$$

Since the input $\mathbf{x}(k)$ and the inherent noise $\varepsilon(k)$ is assumed to be strictly stationary sequences cf. As. A.1 the results of App. C are applicable. See also footnote 17 on page 340. Taking the expectation of `gend:sectermin` and assuming that $N \gg 2M+1$ we neglect terms which decrease faster than $(2M+1)/N$. Inspection of the above equation yields that the expectation of all but the first term decreases faster than $(2M+1)/N$ as the remaining terms induced by expansion of \mathbf{H}_N^{-1} is a product-sum of three or more stochastic matrices which cf. App. C decreases as $((2M+1)/N)^q$, $q \geq 2$.

Analogous arguments lead to that the term `gend:tterm` is approximated similarly. In summary Γ cf. `gend:lam` is approximated by:

$$\begin{aligned}
\Gamma &\approx E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + \\
&2 \cdot \text{tr} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) \boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \right\} \mathbf{H}^{-1}(\mathbf{w}^*) \right]. \tag{A.123}
\end{aligned}$$

Applying the result of App. C and noting that the input is assumed M -dependent²⁰ cf. As. A.3 then

$$\Gamma \approx E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + \frac{2}{N} \cdot \text{tr} \left[\sum_{\tau=-M}^M \frac{N-|\tau|}{N} \boldsymbol{\Phi}(\tau) \mathbf{H}^{-1}(\mathbf{w}^*) \right] \tag{A.124}$$

where the correlation matrix

$$\boldsymbol{\Phi}(\tau) = E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) \boldsymbol{\psi}^\top(k+\tau; \mathbf{w}^*) e(k+\tau; \mathbf{w}^*) \right\}. \tag{A.125}$$

The correlation matrix meet the following identity:

$$\boldsymbol{\Phi}(\tau) = \boldsymbol{\Phi}^\top(-\tau). \tag{A.126}$$

To prove this we for simplicity define:

$$\mathbf{a}(k) = \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) \tag{A.127}$$

and get

$$\begin{aligned}
\boldsymbol{\Phi}(\tau) &= E_{\mathcal{T}} \left\{ \mathbf{a}(k) \mathbf{a}^\top(k+\tau) \right\} \\
&= E_{\mathcal{T}} \left\{ \mathbf{a}(k-\tau) \mathbf{a}^\top(k) \right\} \\
&= E_{\mathcal{T}} \left\{ \left(\mathbf{a}(k) \mathbf{a}^\top(k-\tau) \right)^\top \right\} \\
&= \boldsymbol{\Phi}^\top(-\tau). \tag{A.128}
\end{aligned}$$

Finally, the following estimations are made:

²⁰Alternatively we may assume that the input is a strongly mixing sequence (see further As. A.3). Popularly, that means that $\mathbf{x}(k)$, $\mathbf{x}(k+\tau)$ becomes independent as the lag $\tau \rightarrow \infty$. Therefore, M is determined so that the dependence is negligible (w.r.t. some criterion) as $\tau > M$.

- The first term in `gend:gamapin` is estimated via `gend:snapprox`.
- $\mathbf{H}(\mathbf{w}^*) \approx \mathbf{H}(\hat{\mathbf{w}})$ since the higher order terms in `gend:Gtaylor` is assumed to be negligible. Further, we estimate $\mathbf{H}^{-1}(\hat{\mathbf{w}})$ by $\mathbf{H}_N^{-1}(\hat{\mathbf{w}})$. This of course introduces an statistical error which can be characterized by e.g., the bias²¹,

$$B \left\{ \mathbf{H}_N^{-1} \right\} = E \left\{ \mathbf{H}_N^{-1} \right\} - \mathbf{H}^{-1} \quad (\text{A.129})$$

and the variance,

$$V \left\{ \mathbf{H}_N^{-1} \right\} = E \left\{ \left(\mathbf{H}_N^{-1} - \mathbf{H}^{-1} \right) \otimes \left(\mathbf{H}_N^{-1} - \mathbf{H}^{-1} \right) \right\} \quad (\text{A.130})$$

where \otimes denotes the Kronecker tensor product. Since \mathbf{H} is an $m \times m$ matrix the variance matrix becomes an $m^2 \times m^2$ matrix. However, due to the extend the variance will not be further elaborated.

Using Th. B.4 it is evident that the estimator is unbiased to zero order, i.e.,

$$B \left\{ \mathbf{H}_N^{-1} \right\} = \epsilon \left(\frac{2M+1}{N} \right) \quad (\text{A.131})$$

where $\epsilon \rightarrow \mathbf{0}$. This is sufficient as the term involving $\mathbf{H}^{-1}(\mathbf{w}^*)$ (cf. `gend:gamapin`) is proportional to $(2M+1)/N$ (see App. B for further details).

- The correlation matrix, $\Phi(\tau)$ is estimated in two steps.

1. First, the estimator

$$\hat{\Phi}(\tau) = E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}) | \hat{\mathbf{w}} \right\}. \quad (\text{A.132})$$

is employed. Note that it is necessary to condition on $\hat{\mathbf{w}}$ since the training only result in one weight estimate. The statistical error made by using $\hat{\Phi}(\tau)$ as an estimator of $\Phi(\tau)$ is e.g., expressed in terms of bias, i.e.,

$$B \left\{ \hat{\Phi}(\tau) \right\} = E \left\{ \hat{\Phi}(\tau) \right\} - \Phi(\tau). \quad (\text{A.133})$$

Consider the expansion of $\hat{\Phi}(\tau)$ as a function of \mathbf{w} around \mathbf{w}^* . This gives:

$$\hat{\Phi}(\tau) = \Phi(\tau) + \mathbf{T} \circ \Delta \mathbf{w} + \dots \quad (\text{A.134})$$

where \circ denotes a tensor product as \mathbf{T} is an $m \times m \times m$ tensor which represent the derivative of $\Phi(\tau)$ w.r.t. \mathbf{w} evaluated at \mathbf{w}^* . Furthermore, recall that $\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$. According to `gend:biascor` the bias becomes:

$$B \left\{ \hat{\Phi}(\tau) \right\} = \mathbf{T} \circ E \left\{ \Delta \mathbf{w} \right\} + \dots \quad (\text{A.135})$$

According to `gend:dw`, where $\mathbf{H}_N^{-1}(\mathbf{w}^*)$ is expanded to the first order (in $(2M+1)/N$), i.e., $\mathbf{H}_N^{-1}(\mathbf{w}^*) \approx \mathbf{H}^{-1}(\mathbf{w}^*)$, it is obvious that $E \left\{ \Delta \mathbf{w} \right\} = \mathbf{0}$ as

$$E \left\{ \frac{\partial S_N(\mathbf{w}^*)}{\partial \mathbf{w}} \right\} = \frac{\partial G(\mathbf{w}^*)}{\partial \mathbf{w}} = \mathbf{0}. \quad (\text{A.136})$$

In conclusion the estimator $\hat{\Phi}(\tau)$ is unbiased to zero order in $(2M+1)/N$.

²¹Since the second order expansions of the cost function is assumed to hold the Hessians are approximately equal when evaluated at $\hat{\mathbf{w}}$ and \mathbf{w}^* . The statistical error mentioned here is *not* induced by the fluctuations in $\hat{\mathbf{w}}$; on the contrary, by the fact that the “mean value” \mathbf{H}^{-1} is estimated by the single sample \mathbf{H}_N^{-1} . If more data are available, say QN , it is possible to get a better estimate by averaging Q estimates of \mathbf{H}_N^{-1} . Below the dependence on $\hat{\mathbf{w}}$ is omitted for simplicity.

2. The next step is to estimate $\widehat{\Phi}(\tau)$ by replacing the ensemble-averaging with a time-averaging. We employ the biased estimator²²:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \widehat{\boldsymbol{w}}) e(k; \widehat{\boldsymbol{w}}) \boldsymbol{\psi}^\top(k + \tau; \widehat{\boldsymbol{w}}) e(k + \tau; \widehat{\boldsymbol{w}}) \quad (\text{A.137})$$

where naturally $N > \max |\tau|$. The reason for preferring the biased estimator is that the variance of the individual matrix elements cf. [Papoulis 84b, Sec. 11-2] is $o(1/N)$, i.e., they tend to zero as N^{-1} whereas applying the unbiased estimator²³ the variance is $o(1/(N - |\tau|))$. The bias of the estimator is proportional to $|\tau|/N$ which implies that the assumption $N \gg 2M + 1$, used elsewhere, ensures a small bias (recall that M is the maximum lag). Next the noise on the estimator, $\mathbf{R}(\tau)$, considered as a matrix is treated. Define the error matrix:

$$\mathbf{E}(\tau) = \widehat{\Phi}(\tau) - \mathbf{R}(\tau), \quad (\text{A.138})$$

and the sumnorm $\|\cdot\|_1$ of a matrix $\mathbf{X} = \{x_{ij}\}$ as:

$$\|\mathbf{X}\|_1 = \sum_{i,j} |x_{ij}|. \quad (\text{A.139})$$

The quality of the estimator, $\mathbf{R}(\tau)$ is then evaluated in terms of the sumnorm of the bias and the variance²⁴, i.e., $\|E\{\mathbf{E}(\tau)\}\|_1$ and $\|V\{\mathbf{E}(\tau)\}\|_1$, respectively. Due to the definition of the sumnorm and the results concerning the individual matrix elements it is evident that:

$$\|E\{\mathbf{E}(\tau)\}\|_1 = o\left(\frac{m^2|\tau|}{N}\right) \quad (\text{A.140})$$

$$\|V\{\mathbf{E}(\tau)\}\|_1 = o\left(\frac{m^2}{N}\right) \quad (\text{A.141})$$

since $\mathbf{E}(\tau)$ is an $m \times m$ matrix. That is, one may expect that the error done when using $\mathbf{R}(\tau)$ as an estimator of $\widehat{\Phi}(\tau)$ is proportional to the number of elements in the matrices²⁵, i.e., m^2 . Furthermore, as a matter of fact, it is the fluctuations in the final *GEN*-estimate which should be considered; however, this issue is pretty hard to solve. The only simple fact to notice is that τ runs over $M + 1$ values. That is, we may expect the total statistical error to be proportional to M .

Using these result in `gend:gamapin` we finally get the estimate:

$$GEN = S_N(\widehat{\boldsymbol{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1}(\widehat{\boldsymbol{w}}) \right]. \quad (\text{A.142})$$

²²See e.g., [Papoulis 84b, Sec. 11-2]. Notice, that the estimator generalizes the usual estimator for scalar variables simply by treating each element in the matrix individually.

²³This unbiased estimator is achieved by normalizing with $N - |\tau|$ instead of N in `gend:cormatest`.

²⁴The expectation and variance operations are carried out w.r.t. the individual matrix elements.

²⁵Notice that this result may be somewhat pessimistic since e.g., statistical dependence among the individual matrix elements is not taken into account.

Obviously, the estimator is consistent for $N \rightarrow \infty$ as the last term vanishes and (cf. among other things, the result of Th. A.1)

$$\lim_{N \rightarrow \infty} S_N(\hat{\mathbf{w}}) = G(\mathbf{w}^*). \quad (\text{A.143})$$

Below we treat two special cases of this general result.

Independent Input If the input, $\mathbf{x}(k)$, is an independent sequence the instantaneous gradient, $\boldsymbol{\psi}(k)$, which is a nonlinear function of the input, also becomes independent. Furthermore, if the inherent noise, $\varepsilon(k)$, is second order white and independent of the input the estimate of Γ is obtained by setting $M = 0$ in `gend:gamapin` and thus neglecting the term containing lags, $\tau \geq 1$ in `gend:infin`.

LX-Models Within LX-models the instantaneous gradient, $\boldsymbol{\psi}$ and the Hessian do not depend on the weights. Consequently, the cost function is quadratic in the weights for which reason all Taylor series expansions to the second order, used above, are exact.

This ends the proof of the Theorems A.2 – A.5.

A.2.2 LS Cost Function with Regularization Term

In the subsection we derive the generalization error estimators when applying the cost function:

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa r(\mathbf{w}) \quad (\text{A.144})$$

where $\kappa \geq 0$ is the regularization parameter and $r(\cdot)$ is a regularization function which not depends on training data. The expected cost function yields cf. `gend:excst`:

$$C(\mathbf{w}) = G(\mathbf{w}) + \kappa r(\mathbf{w}). \quad (\text{A.145})$$

In analogy with Def. A.7, make the following definitions:

DEFINITION A.8

1. *The gradient vector of the cost function:*

$$\begin{aligned} \frac{\partial C_N(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial S_N(\mathbf{w})}{\partial \mathbf{w}} + \kappa \frac{\partial r(\mathbf{w})}{\partial \mathbf{w}} \\ &= -\frac{2}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \mathbf{w}) e(k; \mathbf{w}) + \kappa \frac{\partial r(\mathbf{w})}{\partial \mathbf{w}}. \end{aligned} \quad (\text{A.146})$$

2. *The gradient vector of the expected cost function:*

$$\begin{aligned} \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial G(\mathbf{w})}{\partial \mathbf{w}} + \kappa \frac{\partial r(\mathbf{w})}{\partial \mathbf{w}} \\ &= -2E_{\mathbf{x}_t, \varepsilon_t} \{ \boldsymbol{\psi}_t(\mathbf{w}) e_t(\mathbf{w}) \} + \kappa \frac{\partial r(\mathbf{w})}{\partial \mathbf{w}}. \end{aligned} \quad (\text{A.147})$$

3. *The Hessian matrix of the cost function:*

$$\mathbf{J}_N(\mathbf{w}) = \frac{1}{2} \frac{\partial^2 C_N(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} = \mathbf{H}_N(\mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (\text{A.148})$$

4. The Hessian matrix of the expected cost function:

$$\mathbf{J}(\mathbf{w}) = \frac{1}{2} \frac{\partial^2 C(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} = \mathbf{H}(\mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top}. \quad (\text{A.149})$$

Assumption A.11 is replaced by:

ASSUMPTION A.13 *Let the minimization of C_N on the training set \mathcal{T} result in the estimate: $\hat{\mathbf{w}}$. Assume the existence of an optimal weight vector \mathbf{w}^* such that the remainders of the second order Taylor series expansion of C around \mathbf{w}^* are negligible. That is: Let $\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$ then*

$$C(\hat{\mathbf{w}}) \approx C(\mathbf{w}^*) + \Delta \mathbf{w}^\top \mathbf{J}(\mathbf{w}^*) \Delta \mathbf{w} \quad (\text{A.150})$$

as $\partial C(\mathbf{w}^*)/\partial \mathbf{w} = \mathbf{0}$. Further assume that the remainders of expanding C_N around $\hat{\mathbf{w}}$ to the second order is negligible, i.e.,

$$C_N(\mathbf{w}^*) \approx C_N(\hat{\mathbf{w}}) + \Delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w}. \quad (\text{A.151})$$

Below are listed three theorems stating the GEN-estimate when using a cost function with a regularizer.

The two first theorems, Th. A.9 and A.10, differ mainly in the assumed dependence lag, M , of the input. Contrary to the previous section it is interesting to notice that without ado no special theorems result when the model is complete. The reason is that the optimal error (the error when employing the optimal weights) does not become independent of the input. However, introducing the additional condition that the regularization parameter is small leads to a more simple result which coincides with the GPE-estimate [Moody 91], [Moody 92].

THEOREM A.9 *Suppose that the nonlinear system is given by `gend:nonsys` and the model given by `gend:model` is either an NN- or an LX-model. The model may be **complete** as well as **incomplete**. Further, suppose that As. A.1, A.3 – A.9, A.12 and A.13 hold. The GEN-estimate is then given by:*

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (\text{A.152})$$

where $\text{tr}[\cdot]$ denotes the trace and the involved matrices are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}), \quad 0 \leq \tau \leq M, \quad (\text{A.153})$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^\top}. \quad (\text{A.154})$$

If considering a LX-model then $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$ is replaced by $\mathbf{z}(k)$ defined in `gend:linmod`.

THEOREM A.10 *Suppose that the nonlinear system is given by `gend:nonsys` and the model given by `gend:model` is either an NN- or an LX-model. The model may be **complete** as well as **incomplete**. Further, suppose that As. A.2 holds, and $\mathbf{x}(k)$ is an independent sequence. Finally, As. A.4 – A.9, A.12 and A.13 are supposed to hold. The GEN-estimate is then given by:*

$$\text{GEN} = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[(\mathbf{S} + \mathbf{R}(0)) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (\text{A.155})$$

The involved matrices are calculated as:

$$\mathbf{R}(0) = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) e^2(k; \hat{\mathbf{w}}), \quad (\text{A.156})$$

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^\top}. \quad (\text{A.157})$$

If considering a LX-model then $\boldsymbol{\psi}(k; \hat{\mathbf{w}})$ is replaced by $\mathbf{z}(k)$ defined in `gend:linmod`.

THEOREM A.11 Suppose that the nonlinear system is given by `gend:nonsys` and the model given by `gend:model` is **complete** and either an NN- or an LX-model. Further, suppose that *As. A.2, A.4 – A.9, A.12 and A.13* hold and that \mathbf{w}^* defined in *As. A.13* is the **global minimum**. Finally, the regularization parameter, κ , is required to be small (see `gend:smkap`). The GEN-estimate coincides with the GPE-estimate [Moody 91], [Moody 92]:

$$\text{GEN} = \text{GPE} = S_N(\hat{\mathbf{w}}) + \frac{2\hat{\sigma}_e^2 \hat{m}_{\text{eff}}}{N} \quad (\text{A.158})$$

where $\hat{\sigma}_e^2$ is an estimate of $\sigma_e^2 = E\{e^2(k; \mathbf{w}^*)\}$ (see [Moody 91]) and \hat{m}_{eff} is an estimate of the effective number of weights given by:

$$\hat{m}_{\text{eff}} = \text{tr} \left[\left(\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \hat{\mathbf{w}}) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right]. \quad (\text{A.159})$$

In addition, dealing with an LX-model and employing a weight decay regularizer, i.e., $r(\mathbf{w}) = \mathbf{w}^\top \mathbf{w}$, then

$$m_{\text{eff}} = \sum_{i=1}^m \frac{\lambda_i}{\lambda_i + \kappa} \quad (\text{A.160})$$

where λ_i is the i 'th eigenvalue of the Hessian \mathbf{H}_N .

A.2.2.1 Proof of Theorems

Throughout this subsection we implicitly presume that the conditions of the theorems listed above hold.

Taking the expectation w.r.t. the training set, \mathcal{T} , of the Taylor series expansions `gend:Gtaylorr` and (A.151) yields:

$$E_{\mathcal{T}} \{C_N(\mathbf{w}^*)\} \approx E_{\mathcal{T}} \{C_N(\hat{\mathbf{w}})\} + E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\}, \quad (\text{A.161})$$

$$E_{\mathcal{T}} \{C(\hat{\mathbf{w}})\} \approx E_{\mathcal{T}} \{C(\mathbf{w}^*)\} + E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}(\mathbf{w}^*) \Delta \mathbf{w} \right\}. \quad (\text{A.162})$$

The average generalization error, Γ , is cf. `gend:ccost`

$$\Gamma = E_{\mathcal{T}} \{G(\hat{\mathbf{w}})\} = E_{\mathcal{T}} \{C(\hat{\mathbf{w}})\} - \kappa E_{\mathcal{T}} \{r(\hat{\mathbf{w}})\}. \quad (\text{A.163})$$

The individual terms in these equation are now evaluated.

First Term of `gend:esr` The first term in `gend:esr` is likewise *Sec. A.2.1.1* estimated by `gend:snapprox`, i.e.,

$$\begin{aligned} E_{\mathcal{T}} \{C_N(\hat{\mathbf{w}})\} &= E_{\mathcal{T}} \{S_N(\hat{\mathbf{w}})\} + \kappa E_{\mathcal{T}} \{r(\hat{\mathbf{w}})\} \\ &\approx S_N(\hat{\mathbf{w}}) + \kappa E_{\mathcal{T}} \{r(\hat{\mathbf{w}})\}. \end{aligned} \quad (\text{A.164})$$

Approximate Expression for $\Delta \mathbf{w}$ In order to evaluate the remaining terms an approximate expression of $\Delta \mathbf{w}$ is required. As $\hat{\mathbf{w}}$ minimizes $C_N(\mathbf{w})$

$$\frac{\partial C_N(\hat{\mathbf{w}})}{\partial \mathbf{w}} = \mathbf{0}. \quad (\text{A.165})$$

Proceeding as in Sec. A.2.1.1 then

$$\begin{aligned} \Delta \mathbf{w} &\approx - \left[\frac{\partial^2 C_N(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right]^{-1} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \\ &= \mathbf{J}_N^{-1}(\mathbf{w}^*) \left(\frac{1}{N} \sum_{k=1}^N \psi(k; \mathbf{w}^*) e(k; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \end{aligned} \quad (\text{A.166})$$

where we used Def. A.8 and further that

$$\mathbf{J}_N(\mathbf{w}) \approx \mathbf{J}_N(\mathbf{w}^*). \quad (\text{A.167})$$

for all \mathbf{w} lying in (or at the boundary of) the hypersphere with centre in \mathbf{w}^* and radius $\Delta \mathbf{w}$.

Second Term of `gend:esr` The second term of `gend:esr` yields by applying `gend:dwr`:

$$\begin{aligned} E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^\top \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \right\} &= \text{tr} \left[E_{\mathcal{T}} \left\{ \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \Delta \mathbf{w}^\top \right\} \right] \\ &\approx \text{tr} \left[E_{\mathcal{T}} \left\{ \mathbf{J}_N(\mathbf{w}^*) \mathbf{J}_N^{-1}(\mathbf{w}^*) \frac{1}{2} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \frac{1}{2} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \mathbf{J}_N^{-1}(\mathbf{w}^*) \right\} \right] \\ &= \text{tr} \left[E_{\mathcal{T}} \left\{ \frac{1}{2} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}} \frac{1}{2} \frac{\partial C_N(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \mathbf{J}_N^{-1}(\mathbf{w}^*) \right\} \right] \\ &= \text{tr} \left[E_{\mathcal{T}} \left\{ \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \left(\psi(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \cdot \right. \right. \\ &\quad \left. \left. \left(\psi^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \right) \mathbf{J}_N^{-1}(\mathbf{w}^*) \right\} \right]. \end{aligned} \quad (\text{A.168})$$

The inverse Hessian, $\mathbf{J}_N^{-1}(\mathbf{w}^*)$, is expanded as done in `gend:invhnstar1` and subsequently substituted into `gend:fstofthd`. Using arguments similar to those used on page 342, as we neglect terms which tend to zero faster than $(2M+1)/N$, result in that `gend:fstofthd` reduces to:

$$\begin{aligned} \text{tr} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N E_{\mathcal{T}} \left\{ \left(\psi(k_1; \mathbf{w}) e(k_1; \mathbf{w}) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \cdot \right. \right. \\ \left. \left. \left(\psi^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \right) \mathbf{J}^{-1}(\mathbf{w}^*) \right\} \right]. \end{aligned} \quad (\text{A.169})$$

Applying App. C this equation equals:

$$\frac{1}{N} \cdot \text{tr} \left[\sum_{\tau=-M}^{\tau=M} \frac{N - |\tau|}{N} \tilde{\Phi}(\tau) \mathbf{J}^{-1}(\mathbf{w}^*) \right] \quad (\text{A.170})$$

where

$$\begin{aligned}
\tilde{\Phi}(\tau) &= E_{\mathcal{T}} \left\{ \left(\boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \right. \\
&\quad \left. \left(\boldsymbol{\psi}^{\top}(k + \tau; \mathbf{w}^*) e(k + \tau; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} \right) \right\} \\
&= E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) \boldsymbol{\psi}^{\top}(k + \tau; \mathbf{w}^*) e(k + \tau; \mathbf{w}^*) \right\} \\
&\quad - E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) \right\} \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} \\
&\quad - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} E_{\mathcal{T}} \left\{ \boldsymbol{\psi}^{\top}(k + \tau; \mathbf{w}^*) e(k + \tau; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} \right\}. \tag{A.171}
\end{aligned}$$

The first addend equals, $\tilde{\Phi}(\tau)$, defined in `gend:cormat`. Recall that $\tilde{\Phi}(\tau) = \tilde{\Phi}^{\top}(-\tau)$ and that $\tilde{\Phi}(\tau)$ for $0 \leq \tau \leq M$ (cf. `gend:cormatest`) is estimated by:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^{\top}(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}), \quad 0 \leq \tau \leq M. \tag{A.172}$$

The third addend of the expression for $\tilde{\Phi}(\tau)$ is easily seen to be equal to zero as the term in the expectation brackets is proportional to $\partial C(\mathbf{w}^*)/\partial \mathbf{w} = \mathbf{0}$, i.e.,

$$E_{\mathcal{T}} \left\{ \boldsymbol{\psi}^{\top}(k + \tau; \mathbf{w}^*) e(k + \tau; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} \right\} = \mathbf{0}. \tag{A.173}$$

Using this equation (for $\tau = 0$) we get

$$E_{\mathcal{T}} \left\{ \boldsymbol{\psi}^{\top}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) \right\} = \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}}, \tag{A.174}$$

and accordingly the second addend the expression of $\tilde{\Phi}(\tau)$ of equals:

$$-\frac{\kappa^2}{4} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}}. \tag{A.175}$$

An estimate of this addend, say \mathbf{S} , is thus:

$$\mathbf{S} = -\frac{\kappa^2}{4} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^{\top}}. \tag{A.176}$$

The estimator is approximately unbiased to zero order. This is seen by the following arguments: Since the expansion in `gend:Gtaylor` is assumed to hold then

$$\frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \approx \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} + \frac{\partial^2 r(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^{\top}} \Delta \mathbf{w}. \tag{A.177}$$

Applying this expansion results in:

$$\frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}} \frac{\partial r(\hat{\mathbf{w}})}{\partial \mathbf{w}^{\top}} = \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} + \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \Delta \mathbf{w}^{\top} \frac{\partial^2 r(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^{\top}} + \frac{\partial^2 r(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^{\top}} \Delta \mathbf{w} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} + \dots \tag{A.178}$$

Performing the expectation w.r.t \mathcal{T} and noting that $E\{\Delta \mathbf{w}\} = \mathbf{0}$ to zero order in $(2M + 1)/N$ (see page 343) the desired result follows.

In summary the second term of `gend:esr` is estimated by:

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{J}_N \Delta \mathbf{w} \right\} \approx \frac{1}{N} \cdot \text{tr} \left[\left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} \left(\mathbf{R}(\tau) + \mathbf{R}^{\top}(\tau) \right) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right]. \quad (\text{A.179})$$

First Term of `gend:egr` We notice that `gend:equal` still holds, i.e.,

$$E_{\mathcal{T}} \{S_N(\mathbf{w}^*)\} = E_{\mathcal{T}} \{G(\mathbf{w}^*)\}. \quad (\text{A.180})$$

Consequently, cf. `gend:encost` and (A.145)

$$E_{\mathcal{T}} \{C_N(\mathbf{w}^*)\} = E_{\mathcal{T}} \{C(\mathbf{w}^*)\}. \quad (\text{A.181})$$

Second Term of `gend:egr` The second term of `gend:egr` yields:

$$E_{\mathcal{T}} \left\{ \Delta \mathbf{w}^{\top} \mathbf{J}(\mathbf{w}^*) \Delta \mathbf{w} \right\}.$$

Proceeding as in the paragraph concerning the second term of `gend:esr` we note that the only difference appears in `gend:fstofthd` as the product $\mathbf{J}_N(\hat{\mathbf{w}}) \mathbf{J}_N^{-1}(\mathbf{w}^*)$ is replaced by the product $\mathbf{J}(\mathbf{w}^*) \mathbf{J}_N^{-1}(\mathbf{w}^*)$ which not immediately cancels to the identity matrix. However, due to fact that only the first term, that is $\mathbf{J}^{-1}(\mathbf{w}^*)$, in the expansion of $\mathbf{J}_N^{-1}(\mathbf{w}^*)$ is used when neglecting terms which tend to zero faster than $(2M+1)/N$ the product cancels after all. In consequence, the final result is identical to that presented in `gend:esthird`.

The Final Result We are now ready to state the final result of the *GEN* estimator by inserting the individual terms into `gend:esr` and (A.162) and further utilize `gend:gamdefr`. In that context note that the terms of the form $\kappa E_{\mathcal{T}} \{r(\hat{\mathbf{w}})\}$ vanishes. Accordingly:

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{S} + \mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} \left(\mathbf{R}(\tau) + \mathbf{R}^{\top}(\tau) \right) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right]. \quad (\text{A.182})$$

The comments done on page 345 – concerning the cases of LX-models and independent inputs – hold in the present case too.

This end the proof of Theorems A.9 – A.10.

Approach based on Independence In Sec. A.2.1.1, concerning the derivation of the *GEN*-estimate when using an ordinary LS cost function, we saw that when dealing with complete models (provided that \mathbf{w}^* is the global optimum) the fact that the optimal error, $e(k; \mathbf{w}^*)$, became equal to the inherent noise $\varepsilon(k)$ was exploited to carry out the expectation w.r.t. to the input, \mathbf{x} , and the noise, ε , independent of each other. This indeed leads to more simple results. However, when using the cost function with a regularization term the optimal error *does not* equals the inherent noise even when the model is complete and \mathbf{w}^* is the global optimum. This is quite obvious if we evaluate the expected cost function. Applying `gend:gcom` and (A.16) gives:

$$C(\mathbf{w}) = E \left\{ \varepsilon^2 \right\} + E \left\{ [f(\mathbf{x}; \mathbf{w}^{\circ}) - f(\mathbf{x}; \mathbf{w})]^2 \right\} + 2\kappa r(\mathbf{w}) E \left\{ f(\mathbf{x}; \mathbf{w}^{\circ}) - f(\mathbf{x}; \mathbf{w}) \right\} + \kappa^2 r^2(\mathbf{w}). \quad (\text{A.183})$$

Hence, $\mathbf{w} = \mathbf{w}^\circ$ does not in general minimize $C(\mathbf{w})$ and in consequence the adjustment error $f(\mathbf{x}; \mathbf{w}^\circ) - f(\mathbf{x}; \mathbf{w}^*) \neq 0$ (as $\mathbf{w}^* = \arg \min_{\mathbf{w}} C(\mathbf{w})$). We therefore conclude that the optimal error

$$e(k; \mathbf{w}^*) = \varepsilon(k) + f(\mathbf{x}(k); \mathbf{w}^\circ) - f(\mathbf{x}(k); \mathbf{w}^*) \quad (\text{A.184})$$

depends *both on the input and the inherent noise*.

However, when $\kappa \rightarrow 0$ then $\mathbf{w}^* \rightarrow \mathbf{w}^\circ$. This leads to an argument of treating the optimal error as independent of the input, and further being a white sequence, provided that the following conditions are met:

- The input and the inherent noise are independent and the inherent noise is a white sequence according to As. A.2.
- The model is complete.
- \mathbf{w}^* is the global optimum of $C(\mathbf{w})$ (for further reference see As. A.13).
- The regularization parameter, κ , is small²⁶

All terms involved in `gend:esr` and (A.162) but the second remain unchanged in this specific case. Therefore we focus on the second terms. To be more specific, recall the expression in `gend:fstofthd` where \mathbf{J}_N^{-1} is replaced by the \mathbf{J}^{-1} according to the fact that terms which tend to zero faster than $(2M + 1)/N$ is neglected. The post-multiplication with \mathbf{J}^{-1} is thus of no importance w.r.t. to the expectation. Consequently, for the sake of simplicity this matrix – including the trace operator – is omitted, i.e.,

$$E_{\mathcal{T}} \left\{ \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \left(\boldsymbol{\psi}(k_1; \mathbf{w}^*) e(k_1; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right) \cdot \left(\boldsymbol{\psi}^\top(k_2; \mathbf{w}^*) e(k_2; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \right) \right\}. \quad (\text{A.185})$$

As the optimal error, $e(k; \mathbf{w}^*)$, is assumed independent of the input it is independent of the instantaneous gradient, $\boldsymbol{\psi}^\top(k; \mathbf{w}^*)$, too. Furthermore, since the optimal error is supposed to be white the expectation only contributes when $k_1 = k_2$. Consequently, `gend:trdspe` gives:

$$\begin{aligned} & \frac{1}{N} E_{\mathcal{T}} \left\{ e^2(k; \mathbf{w}^*) \right\} E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) \boldsymbol{\psi}^\top(k; \mathbf{w}^*) \right\} \\ & - \frac{1}{N} E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) \right\} \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top} \\ & - \frac{\kappa}{2N} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} E_{\mathcal{T}} \left\{ \boldsymbol{\psi}(k; \mathbf{w}^*) e(k; \mathbf{w}^*) - \frac{\kappa}{2} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \right\}. \end{aligned} \quad (\text{A.186})$$

The third addend is obviously equal to zero cf. e.g., `gend:gradopt`. The second addend is like `gend:secthd` equal to

$$-\frac{\kappa^2}{4N} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^\top}. \quad (\text{A.187})$$

²⁶The adjective “small” is rather intractable to specify unless the distributions of the input and the inherent noise are known. Furthermore it depends on the specific structure of the model according to `gend:smkap`.

When κ is small, more accurately, when the second addend is “small” compared to the first, we may neglect this terms. That is,

$$\kappa \ll 2 \sqrt{\frac{E_{\mathcal{T}} \{e^2(k; \mathbf{w}^*)\} \left\| E_{\mathcal{T}} \{ \boldsymbol{\psi}(k; \mathbf{w}^*) \boldsymbol{\psi}^{\top}(k; \mathbf{w}^*) \} \right\|}{\left\| \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}} \frac{\partial r(\mathbf{w}^*)}{\partial \mathbf{w}^{\top}} \right\|}} \quad (\text{A.188})$$

where $\| \cdot \|$ denotes any matrix norm.

Finally, the second term becomes:

$$E_{\mathcal{T}} \{ \Delta \mathbf{w}^{\top} \mathbf{J}_N(\hat{\mathbf{w}}) \Delta \mathbf{w} \} \approx \frac{E_{\mathcal{T}} \{ e^2(k; \mathbf{w}^*) \}}{N} \text{tr} \left[E_{\mathcal{T}} \{ \boldsymbol{\psi}(k; \mathbf{w}^*) \boldsymbol{\psi}^{\top}(k; \mathbf{w}^*) \} \mathbf{J}^{-1}(\mathbf{w}^*) \right]. \quad (\text{A.189})$$

When $\kappa = 0$ and the model is complete the term within the trace brackets becomes the identity matrix which implies that the trace equals the number of weights, m (compare with `gend:seccom3` and (A.109)). Consequently, the factor $\text{tr}[\cdot]$ can be interpreted as the effective number of weights, denoted m_{eff} .

Proceeding as in the previous paragraph²⁷ the estimate becomes identical to the *GPE*-estimate [Moody 91], [Moody 92] and is given by:

$$GEN = GPE = S_N(\hat{\mathbf{w}}) + \frac{2\hat{\sigma}_e^2 \hat{m}_{\text{eff}}}{N} \quad (\text{A.190})$$

where $\hat{\sigma}_e^2$ is an estimate of $\sigma_e^2 = E\{e^2(k; \mathbf{w}^*)\}$. In [Moody 91] two proposals are given, e.g.,

$$\hat{\sigma}_e^2 = \frac{N}{N - m_{\text{eff}}} S_N(\hat{\mathbf{w}}). \quad (\text{A.191})$$

Furthermore, \hat{m}_{eff} is the estimate of the effective number of weights given by:

$$\hat{m}_{\text{eff}} = \text{tr} \left[\left(\frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^{\top}(k; \hat{\mathbf{w}}) \right) \mathbf{J}_N^{-1}(\hat{\mathbf{w}}) \right]. \quad (\text{A.192})$$

When dealing with an LX-model and employing the weight decay regularizer a simple expression for \hat{m}_{eff} appears. According to Def. A.8

$$\mathbf{J}_N(\mathbf{w}) = \mathbf{H}_N(\mathbf{w}) + \frac{\kappa}{2} \frac{\partial^2 r(\mathbf{w})}{\partial \boldsymbol{\theta} \mathbf{w}} \mathbf{w}^{\top}. \quad (\text{A.193})$$

Using the weight decay regularizer, $r(\mathbf{w}) = \mathbf{w}^{\top} \mathbf{w}$, and recalling that the Hessian does not depend on the weights then `gend:jureg` becomes:

$$\mathbf{J}_N = \mathbf{H}_N + \kappa \mathbf{I} \quad (\text{A.194})$$

where \mathbf{I} is the $m \times m$ identity matrix. Suppose that the eigenvalue decomposition of \mathbf{H}_N is given by:

$$\mathbf{H}_N = \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^{\top} \quad (\text{A.195})$$

²⁷That is adding up all terms and performing estimation of the involved quantities. In particular, the last addend of `gend:lastterm` vanishes.

where $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m]$ is the *unitary*²⁸ matrix of normalized eigenvectors, \mathbf{q}_i , $i = 1, 2, \dots, m$ and $\mathbf{\Lambda} = \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_m\}$ is the eigenvalue matrix. Applying this decomposition in `gend:jnwe` result in:

$$\begin{aligned} \mathbf{J}_N &= \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top + \kappa\mathbf{Q}\mathbf{Q}^\top \\ &= \mathbf{Q}(\mathbf{\Lambda} + \kappa\mathbf{I})\mathbf{Q}^\top. \end{aligned} \quad (\text{A.196})$$

Note that the sum of the eigenvalue matrix and the identity matrix is an diagonal matrix with elements: $\lambda_i + \kappa$. Consequently, the inverse of \mathbf{J}_N becomes:

$$\mathbf{J}_N^{-1} = \mathbf{Q} \cdot \text{diag}\left\{\left[(\lambda_1 + \kappa)^{-1}, (\lambda_2 + \kappa)^{-1}, \dots, (\lambda_m + \kappa)^{-1}\right]\right\} \cdot \mathbf{Q}^\top. \quad (\text{A.197})$$

Now, recall that in the LX-model case the first factor in `gend:mefhat` is identical to the Hessian, \mathbf{H}_N ; consequently²⁹,

$$\begin{aligned} \hat{m}_{\text{eff}} &= \text{tr}\left[\mathbf{H}_N\mathbf{J}_N^{-1}\right] \\ &= \text{tr}\left[\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top\mathbf{Q} \cdot \text{diag}\left\{\left[(\lambda_1 + \kappa)^{-1}, (\lambda_2 + \kappa)^{-1}, \dots, (\lambda_m + \kappa)^{-1}\right]\right\} \cdot \mathbf{Q}^\top\right] \\ &= \text{tr}\left[\mathbf{Q}^\top\mathbf{Q}\mathbf{\Lambda} \cdot \text{diag}\left\{\left[(\lambda_1 + \kappa)^{-1}, (\lambda_2 + \kappa)^{-1}, \dots, (\lambda_m + \kappa)^{-1}\right]\right\}\right] \\ &= \sum_{i=1}^m \frac{\lambda_i}{\lambda_i + \kappa}. \end{aligned} \quad (\text{A.198})$$

If $\kappa = 0$ then $m_{\text{eff}} = m$ and thus the *GEN*-estimate coincides with the estimates discussed in Sec. A.2.1. When $\kappa > 0$ we notice that $m_{\text{eff}} < m$ since the individual terms in `gend:mefwefi` are less than one. This is due to the fact that $\lambda_i > 0$ as the Hessian matrix is positive definite. A weight decay term thus *reduces* the effective number of weights. This may improve the generalization error as mentioned in Sec. 6.9.

This ends the proof of Th. A.11.

A.3 Summary

In this appendix we derived generalization error estimators for incomplete, XN-models. By incomplete models we mean models which not are able to model the present nonlinear system perfectly. The nonlinear model has the structural form:

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w})$$

where $y(k)$ is the output signal, $\mathbf{x}(k)$ is the input signal, $e(k)$ is the error, and \mathbf{w} the weights (parameters) of the model.

The generalization error is in this context defined as the expected squared prediction error, i.e., $E\{e^2\}$, on a sample $[\mathbf{x}; y]$ which is independent of those used for estimating the model parameters.

The derivation was done both when using the ordinary LS cost function and when using a modified LS cost function which includes a regularization term for the estimation of model parameters.

²⁸A matrix, \mathbf{Q} , is said to be unitary if $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$.

²⁹Below the rule: $\text{tr}[\mathbf{A}\mathbf{B}] = \text{tr}[\mathbf{B}\mathbf{A}]$ was used.

APPENDIX B

APPROXIMATION OF INVERSE STOCHASTIC MATRICES

In this appendix we derive an approximate expression for the inverse of a stochastic matrix

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{S}(k)$$

in the large sample limit, i.e., $N \rightarrow \infty$. Typically \mathbf{H}_N is the Hessian of the cost function.

The approximation is used e.g., to find the inverse Hessian matrix of the cost function when deriving generalization error estimates, see further App. A.

B.1 Approximation of \mathbf{H}_N^{-1}

Let the symmetric Hessian matrix $\mathbf{H}_N \in \mathbb{R}^{m \times m}$ be given by

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{S}(k) \tag{B.1}$$

where $\mathbf{S}(k) = \{S_{ij}(k)\}$. Assume that $S_{ij}(k)$ are mean-ergodic random sequences¹ and define the expected Hessian

$$\mathbf{H} = E\{\mathbf{H}_N\}, \tag{B.2}$$

then:

$$\mathbf{H} = \lim_{N \rightarrow \infty} \mathbf{H}_N. \tag{B.3}$$

\mathbf{H}_N is now expressed as a perturbation of \mathbf{H} :

$$\mathbf{H}_N = \mathbf{H} + \Theta \tag{B.4}$$

where Θ is the perturbation matrix. According to Eq. (B.3) $\Theta \rightarrow \mathbf{0}$ as $N \rightarrow \infty$. Note that Θ is symmetric as both \mathbf{H}_N and \mathbf{H} are symmetric.

¹A sufficient condition is that $S_{ij}(k)$ are weakly stationary sequences with finite variances and that the autocovariance $\gamma(\tau) = E\{(S_{ij}(k) - H_{ij})(S_{ij}(k + \tau) - H_{ij})\} \rightarrow 0$ as $|\tau| \rightarrow \infty$ where $H_{ij} = E\{S_{ij}(k)\}$ [Papoulis 84a, Ch. 9-5].

Assuming that both \mathbf{H}_N and \mathbf{H} are invertible the inverse of \mathbf{H}_N in terms of Θ is by Eq. (B.4)

$$\mathbf{H}_N^{-1} = \left(\mathbf{I} + \mathbf{H}^{-1}\Theta \right)^{-1} \mathbf{H}^{-1} \quad (\text{B.5})$$

where \mathbf{I} is the $m \times m$ identity matrix.

Next, we concentrate on finding a series expansion of $(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}$ in terms of Θ . This is guided by the *Matrix Inversion Lemma* (MIL):

LEMMA B.1 (MATRIX INVERSION LEMMA) *If $\mathbf{A} \in \mathbb{R}^{m \times m}$ and $\mathbf{C} \in \mathbb{R}^{p \times p}$ are invertible matrices, $\mathbf{B} \in \mathbb{R}^{m \times p}$, and $\mathbf{D} \in \mathbb{R}^{p \times m}$ then (e.g., [Ljung 87, p. 306]):*

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B} \left(\mathbf{DA}^{-1}\mathbf{B} + \mathbf{C}^{-1} \right)^{-1} \mathbf{DA}^{-1}. \quad (\text{B.6})$$

Setting $\mathbf{A} = \mathbf{C} = \mathbf{I}$, $\mathbf{B} = \mathbf{H}^{-1}$, and $\mathbf{D} = \Theta$ the MIL gives

$$\left(\mathbf{I} + \mathbf{H}^{-1}\Theta \right)^{-1} = \mathbf{I} - \mathbf{H}^{-1} \left(\Theta\mathbf{H}^{-1} + \mathbf{I} \right)^{-1} \Theta, \quad (\text{B.7})$$

and setting $\mathbf{A} = \mathbf{C} = \mathbf{I}$, $\mathbf{B} = \Theta$, and $\mathbf{D} = \mathbf{H}^{-1}$ gives

$$\left(\Theta\mathbf{H}^{-1} + \mathbf{I} \right)^{-1} = \mathbf{I} - \Theta \left(\mathbf{I} + \mathbf{H}^{-1}\Theta \right)^{-1} \mathbf{H}^{-1}. \quad (\text{B.8})$$

We notice that $(\Theta\mathbf{H}^{-1} + \mathbf{I})^{-1} (\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}$ appears both on the right hand side of ih:first and on the left hand side of Eq. (B.8) and further that $(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}$ appears on the right hand side of ih:second and on the left hand side of Eq. (B.7). By successively applying Eq. (B.7), (B.8) we get:

$$\left(\mathbf{I} + \mathbf{H}^{-1}\Theta \right)^{-1} = \mathbf{I} + \sum_{i=1}^q (-1)^i \left(\mathbf{H}^{-1}\Theta \right)^i + \mathbf{R}_q \quad (\text{B.9})$$

where \mathbf{R}_q is the remainder

$$\mathbf{R}_q = \begin{cases} \left(\mathbf{H}^{-1}\Theta \right)^{\frac{q+1}{2}} \left(\mathbf{I} + \mathbf{H}^{-1}\Theta \right)^{-1} \left(\mathbf{H}^{-1}\Theta \right)^{\frac{q+1}{2}} & , q \text{ odd} \\ -\mathbf{H}^{-1} \left(\Theta\mathbf{H}^{-1} \right)^{\frac{q}{2}} \left(\mathbf{I} + \mathbf{H}^{-1}\Theta \right)^{-1} \left(\Theta\mathbf{H}^{-1} \right)^{\frac{q}{2}} \Theta & , q \text{ even} \end{cases}, \quad (\text{B.10})$$

and

$$\left(\mathbf{H}^{-1}\Theta \right)^i = \underbrace{\left(\mathbf{H}^{-1}\Theta \right) \dots \left(\mathbf{H}^{-1}\Theta \right)}_{i \text{ terms}}. \quad (\text{B.11})$$

We are now capable of formulating two theorems concerning the series expansion in ih:serie.

THEOREM B.1 *The remainder \mathbf{R}_q in Eq. (B.10) satisfy*

$$\frac{\mathbf{R}_q}{\|\Theta\|^q} \rightarrow \mathbf{0}, \text{ as } \Theta \rightarrow \mathbf{0}. \quad (\text{B.12})$$

That is, $\mathbf{R}_q = o(\|\Theta\|^q)$ where $\|\cdot\|$ denotes any matrix norm.

PROOF Assume q is odd (similar arguments are applicable for q even) then taking the norm of Eq. (B.10) and using the inequalities of calculus with matrix norms gives:

$$\begin{aligned}\|\mathbf{R}_q\| &= \left\| (\mathbf{H}^{-1}\Theta)^{\frac{q+1}{2}} (\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1} (\mathbf{H}^{-1}\Theta)^{\frac{q+1}{2}} \right\| \\ &\leq \|\mathbf{H}^{-1}\|^{q+1} \cdot \|(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}\| \cdot \|\Theta\|^{q+1}.\end{aligned}\quad (\text{B.13})$$

Using Eq. (B.7) and (B.8) successively we get

$$\|(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}\| = \|\mathbf{I} - \mathbf{H}^{-1}\Theta + \mathbf{H}^{-1}\Theta(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}\mathbf{H}^{-1}\Theta\|. \quad (\text{B.14})$$

↓

$$\|(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}\| \leq 1 + \|\mathbf{H}^{-1}\| \|\Theta\| + \|\mathbf{H}^{-1}\|^2 \|\Theta\|^2 \|(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}\|. \quad (\text{B.15})$$

↓

$$\|(\mathbf{I} + \mathbf{H}^{-1}\Theta)^{-1}\| \leq \frac{1 + \|\mathbf{H}^{-1}\| \|\Theta\|}{1 - \|\mathbf{H}^{-1}\|^2 \|\Theta\|^2}. \quad (\text{B.16})$$

Now substituting Eq. (B.16) into Eq. (B.13) we finally get:

$$\frac{\|\mathbf{R}_q\|}{\|\Theta\|^q} \leq \frac{\|\mathbf{H}^{-1}\|^{q+1} (1 + \|\Theta\| \|\mathbf{H}^{-1}\|) \|\Theta\|}{1 - \|\mathbf{H}^{-1}\|^2 \|\Theta\|^2} \rightarrow 0, \text{ as } \Theta \rightarrow \mathbf{0}. \quad (\text{B.17})$$

↓

$$\frac{\mathbf{R}_q}{\|\Theta\|^q} \rightarrow \mathbf{0}, \text{ as } \Theta \rightarrow \mathbf{0}. \quad (\text{B.18})$$

■

Next, a condition for the convergence of the series in ih:serie is provided.

DEFINITION B.1 (THE PRINCIPLE OF CONVERGENCE) *The series*

$$\sum_{i=1}^{\infty} \mathbf{B}_i$$

is convergent, i.e.,

$$\lim_{n \rightarrow \infty} \left\{ \sum_{i=1}^n \mathbf{B}_i \right\} = \mathbf{B}. \quad (\text{B.19})$$

if and only if, $\forall \epsilon \in \mathbb{R}_+, \exists i_0 \in \mathbb{N}$ such that:

$$\|\mathbf{B}_{i+1} + \dots + \mathbf{B}_{i+p}\| < \epsilon, \quad \forall i > i_0, \forall p \in \mathbb{N}. \quad (\text{B.20})$$

THEOREM B.2 *If the absolute series*

$$\sum_{i=0}^{\infty} \|\mathbf{B}_i\| \quad (\text{B.21})$$

is convergent then the series

$$\sum_{i=0}^{\infty} \mathbf{B}_i \quad (\text{B.22})$$

is convergent.

PROOF According to the principle of convergence, $\forall \epsilon \in \mathbb{R}_+, \exists i_0 \in \mathbb{N}$ such that

$$\| \mathbf{B}_{i+1} \| + \dots + \| \mathbf{B}_{i+p} \| < \epsilon \quad \forall i > i_0, \forall p \in \mathbb{N}. \quad (\text{B.23})$$

↓

$$\| \mathbf{B}_{i+1} + \dots + \mathbf{B}_{i+p} \| \leq \| \mathbf{B}_{i+1} \| + \dots + \| \mathbf{B}_{i+p} \| < \epsilon \quad \forall i > i_0, \forall p \in \mathbb{N}. \quad (\text{B.24})$$

■

THEOREM B.3 *The series*

$$\mathbf{I} + \sum_{i=1}^{\infty} (-1)^i (\mathbf{H}^{-1} \Theta)^i \quad (\text{B.25})$$

is convergent cf. Eq. (B.1) if $\| \mathbf{H}^{-1} \| \| \Theta \| < 1$.

PROOF We apply the quotient criterion on the absolute series of ih:infserie

$$\| \mathbf{I} \| + \sum_{i=1}^{\infty} \| \mathbf{H}^{-1} \Theta \|^i. \quad (\text{B.26})$$

That is, the absolute series is convergent if $c < 1$ where

$$\frac{\| \mathbf{H}^{-1} \Theta \|^{i+1}}{\| \mathbf{H}^{-1} \Theta \|^i} \rightarrow c, \text{ as } i \rightarrow \infty. \quad (\text{B.27})$$

As

$$\frac{\| \mathbf{H}^{-1} \Theta \|^{i+1}}{\| \mathbf{H}^{-1} \Theta \|^i} \leq \frac{\| \mathbf{H}^{-1} \Theta \|^i \| \mathbf{H}^{-1} \| \| \Theta \|}{\| \mathbf{H}^{-1} \Theta \|^i} \rightarrow \| \mathbf{H}^{-1} \| \| \Theta \| \text{ as } i \rightarrow \infty \quad (\text{B.28})$$

the absolute series is convergent if $\| \mathbf{H}^{-1} \| \| \Theta \| < 1$ and according to Th. B.2 the series ih:infserie is convergent ■

COROLLARY B.1 *According to Th. B.3 and ih:invh, (B.9) the expansion of \mathbf{H}_N^{-1} is convergent and given by*

$$\mathbf{H}_N^{-1} = \left(\mathbf{I} + \sum_{i=1}^{\infty} (-1)^i (\mathbf{H}^{-1} \Theta)^i \right) \mathbf{H}^{-1} \quad (\text{B.29})$$

if $\| \mathbf{H}^{-1} \| \| \Theta \| < 1$.

B.2 Approximation of $E\{\mathbf{H}_N^{-1}\}$

Only in simple, trivial cases it is possible to find an exact expression for the expectation of the inverse Hessian. This is due to the facts: 1. The distribution of \mathbf{H}_N has to be known, and 2. it has to be feasible to carry out the expectation analytically. Therefore, in general we may resort to approximate results. A case in point of an exact expression is given in the following

EXAMPLE B.1

Let $\mathbf{x}(k) \in \mathcal{N}(\mathbf{0}, \mathbf{H})$, $\mathbf{x}(k_1)$ independent of $\mathbf{x}(k_2)$ as $k_1 \neq k_2$ and define the $m \times m$ Hessian matrix as

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{x}(k) \mathbf{x}^\top(k). \quad (\text{B.30})$$

According to [Anderson 84, Lemma 7.7.1]²

$$E\{\mathbf{H}_N^{-1}\} = \frac{N}{N-m-1} \mathbf{H}^{-1}, \quad N > m+1 \quad (\text{B.31})$$

□

The series expansion in ih:serie is generally used to obtain approximate results. Employing the expansion and ih:invh the expectation of the inverse Hessian matrix yields:

$$\begin{aligned} E\{\mathbf{H}_N^{-1}\} &= E\left\{\left(\mathbf{I} + \sum_{i=1}^{\infty} (-1)^i (\mathbf{H}^{-1} \boldsymbol{\Theta})^i\right) \mathbf{H}^{-1}\right\} \\ &= \left(\mathbf{I} + \sum_{i=1}^{\infty} (-1)^i E\{(\mathbf{H}^{-1} \boldsymbol{\Theta})^i\}\right) \mathbf{H}^{-1}. \end{aligned} \quad (\text{B.32})$$

Above all expectations are assumed to exist. By using ih:expec, (B.4) the first order term in ih:exinv gives:

$$-E\{\mathbf{H}^{-1} \boldsymbol{\Theta}\} = -\mathbf{H}^{-1} E\{\mathbf{H}_N - \mathbf{H}\} = \mathbf{0}. \quad (\text{B.33})$$

The second order term gives:

$$\begin{aligned} E\{\mathbf{H}^{-1} \boldsymbol{\Theta} \mathbf{H}^{-1} \boldsymbol{\Theta}\} &= \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N E\{\mathbf{H}^{-1} \boldsymbol{\theta}(k_1) \mathbf{H}^{-1} \boldsymbol{\theta}(k_2)\} \\ &= \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \mathbf{R}(k_1, k_2) \end{aligned} \quad (\text{B.34})$$

where

$$\boldsymbol{\Theta} = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\theta}(k). \quad (\text{B.35})$$

Recall that \mathbf{H}_N is assumed to be a weakly stationary sequence; consequently, $\boldsymbol{\Theta}$ cf. ih:pertub is a weakly stationary sequence. The expectation in ih:meansec depends therefore only on the difference $\tau = k_2 - k_1$, i.e., $\mathbf{R}(k_1, k_2) = \mathbf{R}(\tau)$. If $\mathbf{S}(k)$ is M -dependent, i.e., $\mathbf{S}(k)$ and $\mathbf{S}(k+\tau)$ are independent for $|\tau| > M$ then the number of terms involved in ih:meansec for τ fixed is $N - |\tau|$ as shown in App. C. Consequently, ih:meansec becomes:

$$E\{\mathbf{H}^{-1} \boldsymbol{\Theta} \mathbf{H}^{-1} \boldsymbol{\Theta}\} = \frac{1}{N} \sum_{\tau=-M}^M \frac{N - |\tau|}{N} \mathbf{R}(\tau). \quad (\text{B.36})$$

An upper bound of the norm of ih:secfin yields:

$$\begin{aligned} \|E\{\mathbf{H}^{-1} \boldsymbol{\Theta} \mathbf{H}^{-1} \boldsymbol{\Theta}\}\| &\leq \frac{1}{N} \sum_{\tau=-M}^M \frac{N - |\tau|}{N} \|\mathbf{R}(\tau)\| \\ &< \frac{2M+1}{N} \max_{-M \leq \tau \leq M} \|\mathbf{R}(\tau)\|. \end{aligned} \quad (\text{B.37})$$

²Note, as we know that the mean of $\mathbf{x}(k)$ is equal to the zero vector, \mathbf{H}_N is Wishart distributed with N degrees of freedom.

That is,

$$\|E\{\mathbf{H}^{-1}\Theta\mathbf{H}^{-1}\Theta\}\| \propto \frac{2M+1}{N}. \quad (\text{B.38})$$

We are now able to state the following

THEOREM B.4 *The expectation of the inverse Hessian is approximated by*

$$E\{\mathbf{H}_N^{-1}\} = \mathbf{H}^{-1} + \epsilon \left(\frac{2M+1}{N} \right) \quad (\text{B.39})$$

where $\epsilon((2M+1)/N) \rightarrow \mathbf{0}$, $(2M+1)/N \rightarrow 0$. That is if $N \gg 2M+1$

$$E\{\mathbf{H}_N^{-1}\} \approx \mathbf{H}^{-1}. \quad (\text{B.40})$$

PROOF Use the fact that norms of the terms of order greater than 1 in the series expansion of the expectation of the inverse Hessian, `ih:exinv`, are negligible if $N \gg 2M+1$ (Recall that the norm of terms have to negligible compared to $\|\mathbf{I}\| = 1$ cf. `ih:exinv`). This is substantiated by using `ih:secapprox` and by noting that the norms of the higher order terms in `ih:exinv` tends to zero faster than $(2M+1)/N$, i.e., they are $o((2M+1)/N)$, cf. App. C ■

In the following two subsections we study the validity of the approximation in Th. B.4. First we elaborate on the condition $N \gg 2M+1$ by studying a numerical example. Secondly, we show that – within LX-models – the approximation “under estimates” the expectation as $E\{\mathbf{H}_N^{-1}\}$ is shown to be greater than³ \mathbf{H}^{-1} .

B.2.1 On the Large N Assumption

Although the norm of the second order term in `ih:secapprox` is proportional to $(2M+1)/N$ it is not possible to state anything about the magnitude of N necessary to ensure the validity of the approximation in `ih:ihapp` without making additional assumptions concerning the distribution and structure of the Hessian. Below we give two examples which show some of the problems with the suggested approximation.

EXAMPLE B.2

The first example is inspired by Volterra filters (see Ch. 3). The intention is to show how N scales with m in order to ensure a close approximation.

Let $x(k)$ be a independent sequence which are marginal $\mathcal{N}(0,1)$ distributed. Consequently, $M=0$ in this case. Further define

$$\mathbf{x}(k) = [x(k), x^2(k), \dots, x^m(k)]^\top, \quad (\text{B.41})$$

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{x}(k)\mathbf{x}^\top(k) \quad (\text{B.42})$$

³Let \mathbf{A} and \mathbf{B} be $m \times m$ matrices then the inequality operator for matrices is defined as:

$$\mathbf{A} > \mathbf{B} \Leftrightarrow \mathbf{a}^\top(\mathbf{A} - \mathbf{B})\mathbf{a} > 0$$

where \mathbf{a} is any non-zero vector with dimension m . That is, \mathbf{A} is greater than \mathbf{B} if the difference $\mathbf{A} - \mathbf{B}$ is positive definite.

where the expected Hessian is given as the Hankel matrix

$$\begin{aligned} \mathbf{H} &= E\{\mathbf{x}(k)\mathbf{x}^\top(k)\} \\ &= \begin{bmatrix} E\{x^2(k)\} & E\{x^3(k)\} & \cdots & E\{x^{m+1}(k)\} \\ E\{x^3(k)\} & E\{x^4(k)\} & \cdots & E\{x^{m+2}(k)\} \\ \vdots & \vdots & \ddots & \vdots \\ E\{x^{m+1}(k)\} & E\{x^{m+2}(k)\} & \cdots & E\{x^{2m}(k)\} \end{bmatrix}. \end{aligned} \quad (\text{B.43})$$

Since $x(k)$ is Gaussian distributed (with variance 1) we have [Bendat & Piersol 86, Ch. 3.3.3]

$$E\{x^q(k)\} = \begin{cases} 0 & q \text{ odd} \\ \prod_{i=1}^{q/2} 2i - 1 & q \text{ even} \end{cases}. \quad (\text{B.44})$$

The 2-norm (i.e., the largest eigenvalue) of the second order term in `ih:secapprox` is estimated by simulation. The estimate is

$$\begin{aligned} \|E\{\mathbf{H}^{-1}\Theta\mathbf{H}^{-1}\Theta\}\|_2 &\approx \|\langle \mathbf{H}^{-1}\Theta\mathbf{H}^{-1}\Theta \rangle\|_2 \\ &= \|\langle \mathbf{H}^{-1}(\mathbf{H}_N - \mathbf{H})\mathbf{H}^{-1}(\mathbf{H}_N - \mathbf{H}) \rangle\|_2 \end{aligned} \quad (\text{B.45})$$

where $\langle \cdot \rangle$ denotes the average with respect to Q independent realization of \mathbf{H}_N . In Fig. B.1 the logarithm (base 10) of the norm of the second order term is shown as a function of dimension, m , and for different values of N . As mentioned in the proof of Th. B.4 the norm of the second order term has to be small compared to 1, i.e., 0 in logarithmic depiction. We notice that the norm scales approximately exponentially with m and even when $N = 500$ the second order term can not be neglected if $m > 3$. This is due the fact that the 2-norm condition number, i.e., the eigenvalue spread, of \mathbf{H} scales approximately exponentially with the dimension, m , as shown in Fig. B.2. The $1/N$ -scaling of the norm is easily verified. Typically the Hessian matrix is badly conditioned so we expect that N has to be rather large in order to ensure an accurate approximation. However, in the simple case discussed in example B.1 the scaling with m is linear as cf. `ih:exphes`

$$\begin{aligned} E\{\mathbf{H}_N^{-1}\} &= \frac{N}{N - m - 1} \mathbf{H}^{-1} \\ &= \frac{1}{1 - \frac{m+1}{N}} \mathbf{H}^{-1} \\ &\approx \mathbf{H}^{-1}, \quad \frac{m}{N} \ll 1. \end{aligned} \quad (\text{B.46})$$

□

EXAMPLE B.3

This example serves the purpose of showing how the correlation (dependence) of the matrix sequence $\mathbf{S}(k)$ influences the approximation in Th. B.4. Consider a 3-dimensional ($m = 3$) vector signal

$$\mathbf{x}(k) = [x_1(k), x_2(k), x_3(k)]^\top \quad (\text{B.47})$$

and the Hessian matrix

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{x}(k)\mathbf{x}^\top(k). \quad (\text{B.48})$$

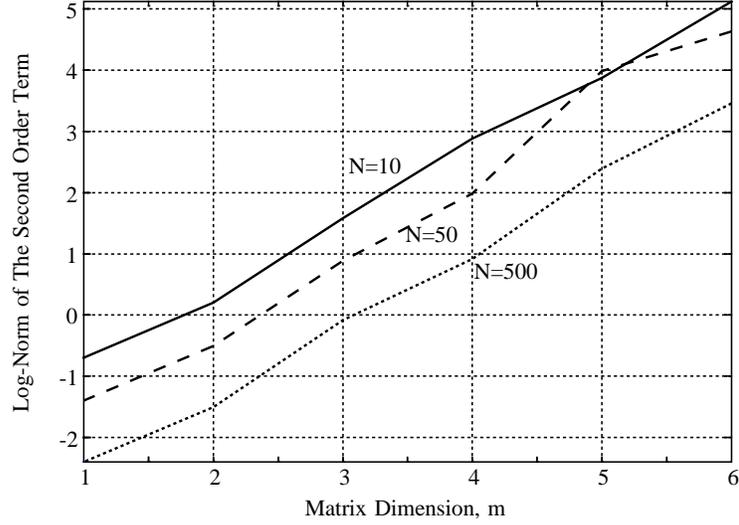


Figure B.1: The logarithm (base 10) of the norm of the second order term estimated due to `ih:normest`. Solid line: $N = 10$, dashed line: $N = 50$, and dotted line: $N = 500$. Average was performed on $Q = 60000$ examples.

We will treat four different $\mathbf{x}(k)$ vector signals: Gaussian and uniformly distributed signals and white as well as colored signals. Let $u_i(k)$, $i = 1, 2, 3$, be independent signals with zero mean and unit variance. In the Gaussian case $u_i(k) \in \mathcal{N}(0, 1)$ and in the uniform case $u_i(k)$ are distributed over the interval $[-\sqrt{3}; \sqrt{3}]$. In the white case, i.e.,

$$E \left\{ \mathbf{x}(k) \mathbf{x}^\top(k + \tau) \right\} = \begin{cases} \mathbf{H} & , \tau = 0 \\ \mathbf{0} & , \tau \neq 0 \end{cases} \quad (\text{B.49})$$

the vector signal $\mathbf{x}(k)$ is generated via the following equations:

$$x_1(k) = -2.77 \cdot \xi_1(k), \quad (\text{B.50})$$

$$x_2(k) = -2.52 \cdot \xi_2(k) - 1.85 \cdot x_1(k), \quad (\text{B.51})$$

$$x_3(k) = -0.212 \cdot \xi_3(k) + 2.40 \cdot x_1(k) + 0.573 \cdot x_2(k). \quad (\text{B.52})$$

Consequently, the expected Hessian becomes:

$$\mathbf{H} = E \left\{ \mathbf{x}(k) \mathbf{x}^\top(k) \right\} = \begin{bmatrix} -7.67 & -14.2 & 10.3 \\ -14.2 & 32.6 & -15.4 \\ 10.3 & -15.4 & 16.0 \end{bmatrix}. \quad (\text{B.53})$$

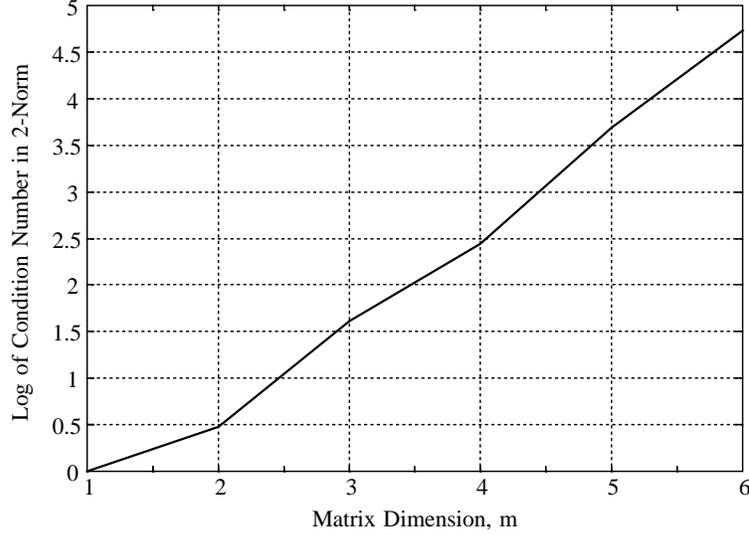


Figure B.2: The logarithm (base 10) of the condition number (in 2-norm) of \mathbf{H} as a function of matrix dimension m .

The colored vector signals are generated simply by filtering the individual components with a FIR-filter with impulse response $h(k)$. That is,

$$x_{c,i}(k) = h(k) * x_i(k) = \sum_{n=0}^{L-1} h(n)x_i(k-n), \quad i = 1, 2, 3 \quad (\text{B.54})$$

where $x_{c,i}(k)$ are the colored signals⁴. The filter is designed⁵ to implement a low-pass filter with normalized cut-off frequency equal to 0.01 and the filter order, L , was set to 11. The $x_{c,i}(k)$ signals thus become correlated over a time lag, M , equal to 10. The expected Hessian of the colored vector signal $\mathbf{x}_c(k)$ is easily calculated as:

$$\begin{aligned} E \{x_{c,i_1}(k)x_{c,i_2}(k)\} &= \sum_{n_1=0}^{L-1} \sum_{n_2=0}^{L-1} h(n_1)h(n_2)E \{x_{i_1}(k-n_1)x_{i_2}(k-n_2)\} \\ &= E \{x_{i_1}(k)x_{i_2}(k)\} \cdot \sum_{n=0}^{L-1} h^2(n) \end{aligned} \quad (\text{B.55})$$

⁴Due to the central limit theorem the $x_{c,i}(k)$ signals will be approximately Gaussian distributed when the $x_i(k)$ signals are independent uniformly distributed. However, we still prefer to denote the case: The colored uniform case.

⁵The design is performed with the MATLAB [MATLAB 94] routine “fir1” which uses a Hamming windowed ideal low-pass filter impulse response (the sinc-function).

where the result of `ih:hescor` was applied. Consequently,

$$\mathbf{H}_c = \mathbf{H} \sum_{n=0}^{L-1} h^2(n) \quad (\text{B.56})$$

Using the filter mentioned above we get:

$$\mathbf{H}_c = 0.1326 \cdot \mathbf{H}. \quad (\text{B.57})$$

Within each case we perform Q independent realizations of the vector signal sequence: $\{\mathbf{x}(k)\}$, $k = 1, 2, \dots, N$. The expectation of the inverse Hessian is now estimated by⁶:

$$E \left\{ \mathbf{H}_N^{-1} \right\} \approx \left\langle \mathbf{H}_N^{-1} \right\rangle \quad (\text{B.58})$$

where $\langle \cdot \rangle$ denotes the average w.r.t. the Q realizations. In this simulation we used:

$$Q = \begin{cases} 109000 & 5 \leq N \leq 10 \\ 54500 & 15 \leq N \leq 50 \end{cases} \quad (\text{B.59})$$

It turned out that – in all cases – it is possible to express the expectation of the inverse Hessian as:

$$E \left\{ \mathbf{H}_N^{-1} \right\} = \varphi(N) \mathbf{H}^{-1}. \quad (\text{B.60})$$

Since $\lim_{N \rightarrow \infty} \mathbf{H}_N = \mathbf{H}$ it is obvious that the prefactor $\varphi(N)$ has to satisfy:

$$\lim_{N \rightarrow \infty} \varphi(N) = 1. \quad (\text{B.61})$$

Recall that in the case of white Gaussian sequences the prefactor (cf. `ih:exphes`) becomes:

$$\varphi(N) = \frac{N}{N - m - 1}, \quad N > m + 1. \quad (\text{B.62})$$

In Fig. B.3 the prefactors obtained by the numerical study are shown⁷. We notice that the prefactors are dominant as long as the inequality $N \gg 2M + 1$ not is satisfied. Furthermore, the prefactors have significantly greater magnitude when the signals are colored. In consequence we may expect that the approximate result of Th. B.4 is quite poor when N is small compared to $2M + 1$.

Only in the simple case of a white Gaussian vector signal we are able to derive an expression for the shape of the prefactor cf. `ih:prefg` as a function of N . However, it is possible to parameterize the shape of the prefactors and perform a numerical fit to the measured prefactors. In the table below the result of the fit using the MATLAB [MATLAB 94] routine “`fmins`” are shown: In the case of white signals the shape is approximated closely by a form equivalent to `ih:prefg`, whereas dealing with colored signals requires a zero order term in the numerator in order to get a close fit \square .

⁶In the colored case consider instead the Hessian, $\mathbf{H}_{c,N}$.

⁷The prefactors are calculated for each sample size, N , according to the equation:

$$\varphi(N) = \frac{1}{9} \sum_{i=1}^3 \sum_{j=1}^3 \frac{\langle H_{N,ij}^{-1} \rangle}{H_{ij}^{-1}}$$

where the subindices denote the matrix element in the i 'th row and the j 'th column.

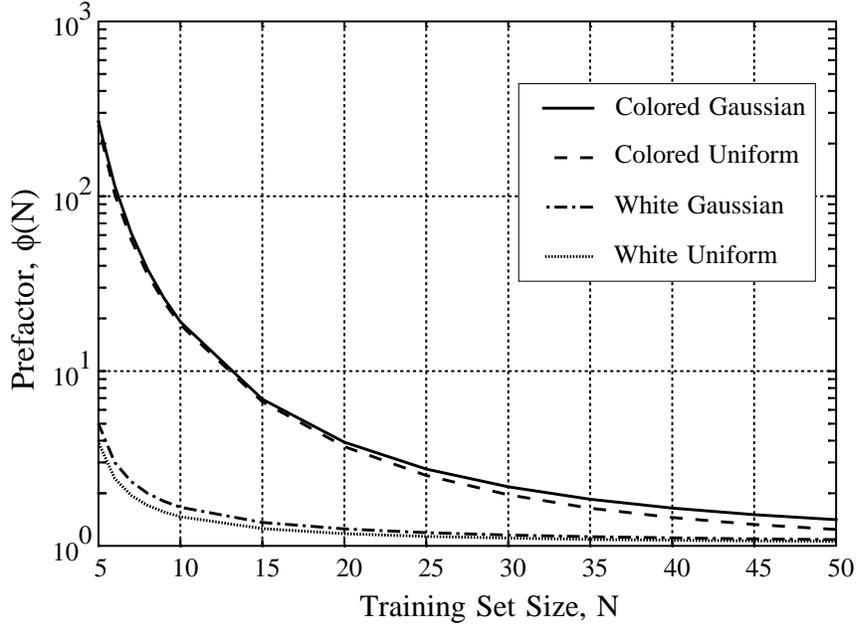


Figure B.3: Prefactors as a function of the number samples, N . It is noted that the prefactors are of major importance especially in the colored cases when $N \gg 2M + 1$ is not fulfilled.

	Gaussian	Uniform
White	$\frac{N - 0.1166}{N - 4.052}$	$\frac{N - 1.235}{N - 4.037}$
Colored	$\frac{N + 150.0}{N - 4.423}$	$\frac{N + 133.5}{N - 4.461}$

Table B.1: The shape of the prefactors in the various cases found by numerical fits.

B.2.2 LX-models

In this subsection we show that the approximation Th. B.4 within LX-models so to speak “under determines” the true expectation. This result is used in App. A.

THEOREM B.5 Consider LX-models, i.e., the Hessian matrix is given as (see e.g., App. A)

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \quad (\text{B.63})$$

where $\mathbf{z}(k)$ is the input vector. The approximation in Th. B.4 “under estimates” the true expectation, $E\{\mathbf{H}_N^{-1}\}$, as

$$E\{\mathbf{H}_N^{-1}\} > \mathbf{H}^{-1}. \quad (\text{B.64})$$

It is assumed that both \mathbf{H}_N and \mathbf{H} are positive definite matrices.

PROOF First consider the perturbation matrix $\Theta = \mathbf{H}_N - \mathbf{H}$ which by ih:H, (B.4) and (B.63) can be written as:

$$\Theta = - \lim_{n \rightarrow \infty} \frac{1}{n - N} \sum_{k=N+1}^n \mathbf{z}(k) \mathbf{z}^\top(k). \quad (\text{B.65})$$

Θ is negative definite which is perceived by:

$$\begin{aligned} \mathbf{a}^\top \Theta \mathbf{a} &= - \lim_{n \rightarrow \infty} \frac{1}{n - N} \sum_{k=N+1}^n \mathbf{a}^\top \mathbf{z}(k) \mathbf{z}^\top(k) \mathbf{a} \\ &= - \lim_{n \rightarrow \infty} \frac{1}{n - N} \sum_{k=N+1}^n \left(\mathbf{a}^\top \mathbf{z}(k) \right)^2. \end{aligned} \quad (\text{B.66})$$

Since, $\left(\mathbf{a}^\top \mathbf{z}(k) \right)^2 \geq 0 \forall \mathbf{a} \neq \mathbf{0}$ $\mathbf{a}^\top \Theta \mathbf{a}$ is negative definite⁸.

This result is used to show that

$$\mathbf{H}_N^{-1} > \mathbf{H}^{-1}, \quad (\text{B.67})$$

or equivalently,

$$\mathbf{a}^\top (\mathbf{H}_N^{-1} - \mathbf{H}^{-1}) \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}. \quad (\text{B.68})$$

Applying ih:pertub and the MIL on $\mathbf{H}^{-1} = (\mathbf{H}_N - \Theta)^{-1}$ with $\mathbf{A} = \mathbf{H}_N$, $\mathbf{C} = -\Theta$, and $\mathbf{B} = \mathbf{D} = \mathbf{I}$ we get:

$$\begin{aligned} \mathbf{a}^\top (\mathbf{H}_N^{-1} - \mathbf{H}^{-1}) \mathbf{a} &= \mathbf{a}^\top \left(\mathbf{H}_N^{-1} - (\mathbf{H}_N - \Theta)^{-1} \right) \mathbf{a} \\ &= \mathbf{a}^\top \mathbf{H}_N^{-1} \left(\mathbf{H}_N^{-1} - \Theta^{-1} \right)^{-1} \mathbf{H}_N^{-1} \mathbf{a}. \end{aligned} \quad (\text{B.69})$$

The last term is positive $\forall \mathbf{a} \neq \mathbf{0}$ due to the the following facts:

- Define $\mathbf{a}_1 = \mathbf{H}_N^{-1} \mathbf{a}$. Since \mathbf{H}_N is positive definite by assumption, all possible $\mathbf{a}_1 \neq \mathbf{0}$ is obtainable by varying \mathbf{a} .
- If a matrix is positive (negative) definite also the inverse of the matrix is positive (negative) definite⁹.
- $-\Theta^{-1}$ is positive definite according to ih:theta and the preceding item. Further, as \mathbf{H}_N^{-1} is positive definite the inverse of the sum, $\mathbf{H}_N^{-1} + (-\Theta^{-1})$, is positive definite.

Finally, taking the expectation of ih:invpso (as \mathbf{H} is not stochastic) the result follows \blacksquare

⁸Assuming $\mathbf{z}(k)$ to be a random sequence we tacitly exclude the zero probability case of $\mathbf{z}(k_1) \equiv \mathbf{z}(k_2) \forall k_1, k_2 \in [N+1; \infty[$.

⁹If \mathbf{Q} and $\mathbf{\Lambda}$ are the matrices of eigenvectors and eigenvalues of a matrix \mathbf{A} , respectively, then

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{\Lambda}$$

This implies $\mathbf{A}^{-1}\mathbf{Q} = \mathbf{Q}\mathbf{\Lambda}^{-1}$ for the inverse matrix \mathbf{A}^{-1} . Now, if \mathbf{A} is positive definite all eigenvalues λ_i are positive. Consequently, the eigenvalues of the inverse matrix, $1/\lambda_i$ are also positive, thereby ensuring the positive definiteness of \mathbf{A}^{-1} .

B.3 Summary

In this appendix a series expansion of the inverse $m \times m$ Hessian matrix is given in terms of $\Theta = \mathbf{H}_N - \mathbf{H}$. Θ is the difference between the the estimated Hessian \mathbf{H}_N based on N examples and the expectation $\mathbf{H} = E\{\mathbf{H}_N\}$. It is shown that the series is convergent if $\|\mathbf{H}^{-1}\|\|\Theta\| < 1$.

Furthermore, we showed that the expectation of the inverse Hessian $E\{\mathbf{H}_N^{-1}\}$ is approximated by \mathbf{H}^{-1} if $N \gg 1$. However, N may be extremely large in order to ensure a close approximation when \mathbf{H} is badly conditioned. This is exemplified by a numerical study. In addition, the influence of the dependence (put by the dependence lag, M) among the individual samples $\mathbf{S}(k)$ is studied numerically. It is shown that the dependence cause the approximation of the expectation of the inverse Hessian, stated above, to be rather poor when N is small compared to $2M + 1$.

Within LX-models we finally showed that the approximation of $E\{\mathbf{H}_N^{-1}\}$ is an “under estimate” since $E\{\mathbf{H}_N^{-1}\} > \mathbf{H}^{-1}$.

APPENDIX C

EXPECTATION OF PRODUCT-SUMS OF STOCHASTIC MATRICES

This appendix treats evaluation of the expectation of product-sums of stochastic matrices which is used in App. A, App. B, App. F and App. G.

Consider the expectation of the product of q stochastic matrices¹

$$E \left\{ \prod_{i=1}^q \mathbf{r}_i \right\} \quad (\text{C.1})$$

where

$$\mathbf{r}_i = \frac{1}{N} \sum_{k_i=1}^N \mathbf{Y}_i(k_i). \quad (\text{C.2})$$

It is assumed that the matrices have zero means, i.e., $E\{\mathbf{Y}_i(k_i)\} = \mathbf{0}$.

Substituting exp:sum into (C.1) and expanding yield:

$$E \left\{ \prod_{i=1}^q \mathbf{r}_i \right\} = \frac{1}{N^q} \sum_{k_1=1}^N \sum_{k_2=1}^N \cdots \sum_{k_q=1}^N E \{ \mathbf{Y}_1(k_1) \mathbf{Y}_2(k_2) \cdots \mathbf{Y}_q(k_q) \}. \quad (\text{C.3})$$

Assume that all matrix sequences, $\mathbf{Y}_i(k_i)$, originate from a number of strictly stationary sequences which are gathered in the vector $\mathbf{z}(k)$, i.e.,

$$\mathbf{Y}_i(k_i) = \mathbf{F}_i(\mathbf{z}(k_i)) \quad (\text{C.4})$$

where $\mathbf{F}_i(\cdot)$ are continuous mappings. Hence, the product of these matrices is a strictly stationary sequence and exp:expand depends consequently on the lags² $\tau_i = k_{i+1} - k_1$, $i = 1, 2, \dots, q - 1$ only.

Assume that the sequence \mathbf{z} is M -dependent, i.e., $\mathbf{z}(k)$ and $\mathbf{z}(k + \tau)$ are independent when $|\tau| > M$ and further that the dependence lag, $M < N$. It is obvious that

$$-M \leq \tau_i \leq M, \quad 1 \leq i \leq q - 1. \quad (\text{C.5})$$

¹The matrices are assumed to be of appropriate dimensions and the expectation is assumed to exist.

²This is per definition. The lags could be defined w.r.t. an arbitrary of the k_i 's.

The expectation in exp:expand now becomes:

$$E \{ \mathbf{Y}_1(k_1) \mathbf{Y}_2(k_2) \cdots \mathbf{Y}_q(k_q) \} = \mathbf{\Phi}(\tau_1, \tau_2, \cdots, \tau_{q-1}) = \mathbf{\Phi}(\boldsymbol{\tau}) \quad (\text{C.6})$$

where $\mathbf{\Phi}(\boldsymbol{\tau})$ is a correlation matrix, $\boldsymbol{\tau} = [\tau_1, \tau_2, \cdots, \tau_{q-1}]$.

For $\boldsymbol{\tau}$ fixed the number of terms involved in the summation over k_1, k_2, \cdots, k_q in exp:expand is calculated. Apply the following inequalities:

$$\left\{ \begin{array}{l} 1 \leq k_1 \leq N \\ 1 \leq k_2 \leq N \\ \vdots \\ 1 \leq k_q \leq N \end{array} \right\} \quad (\text{C.7})$$

\Leftrightarrow

$$\left\{ \begin{array}{l} 1 \leq k_1 \leq N \\ 1 \leq \tau_1 + k_1 \leq N \\ \vdots \\ 1 \leq \tau_{q-1} + k_1 \leq N \end{array} \right\} \quad (\text{C.8})$$

\Leftrightarrow

$$\left\{ \begin{array}{l} 1 \leq k_1 \leq N \\ 1 - \tau_1 \leq k_1 \leq N - \tau_1 \\ \vdots \\ 1 - \tau_{q-1} \leq k_1 \leq N - \tau_{q-1} \end{array} \right\}. \quad (\text{C.9})$$

That is, $\mathbf{\Phi}(\boldsymbol{\tau})$ appears

$$K(\boldsymbol{\tau}) = \min\{N, N - \tau_1, N - \tau_2, \cdots, N - \tau_{q-1}\} - \max\{1, 1 - \tau_1, 1 - \tau_2, \cdots, 1 - \tau_{q-1}\} + 1 \quad (\text{C.10})$$

times in the sum over k_1, k_2, \cdots, k_q . If $q = 2$ exp:K yields

$$K(\tau_1) = N - |\tau_1|. \quad (\text{C.11})$$

Accordingly, substituting exp:R and (C.10) into exp:expand result in:

$$E \left\{ \prod_{i=1}^q \mathbf{r}_i \right\} = \frac{1}{N^{q-1}} \sum_{\boldsymbol{\tau}} \frac{K(\boldsymbol{\tau})}{N} \mathbf{\Phi}(\boldsymbol{\tau}) \quad (\text{C.12})$$

where the sum is w.r.t. to legal values of $\boldsymbol{\tau}$. This legal values are found be using the fact that $\forall i: 1 \leq k_i \leq N$ and exp:tau; however, the result is irrelevant within this thesis and therefore omitted. When $q = 2$ exp:tau1 shows that legal values of τ_1 is: $-M \leq \tau_1 \leq M$.

The fraction in exp:result,

$$\frac{K(\boldsymbol{\tau})}{N} \leq 1$$

according to exp:K and (C.5). Equality is obtained for $\boldsymbol{\tau} = \mathbf{0}$.

Since the lags, τ_i , individually runs over the range $[-M; M]$ the maximum number of terms in the sum exp:result over legal values³ of $\boldsymbol{\tau}$ is $(2M + 1)^{q-1}$

Consequently, the expectation in exp:result will tend to zero faster or equal to

$$\left(\frac{2M + 1}{N} \right)^{q-1}.$$

³If $q = 2$ the sum runs over $2M + 1$ values. However, if $q > 2$ the sum runs over a number of lag values less than $(2M + 1)^{q-1}$ because not all $[\tau_1, \tau_2, \cdots, \tau_{q-1}]$ combinations are legal.

APPENDIX D

EVALUATION OF GAUSSIAN INTEGRALS

In this appendix we evaluate certain Gaussian integrals involved in the calculation of the generalization error of a simple neural network. The network is used in a simulation study of the *GEN*-estimate, cf. Ch. 6 and App. I.

D.1 One and Two-dimensional Gaussian Integrals

Consider the one-dimensional Gaussian integral:

$$I_1(\alpha, \beta, \gamma) = \int_{-\infty}^{+\infty} \exp\left(-(\alpha z^2 + \beta z + \gamma)\right) dz, \quad \alpha > 0. \quad (\text{D.1})$$

In order to evaluate this integral we use the fact that a p.d.f. always integrates to 1. In particular, if z is Gaussian distributed with mean μ and variance σ^2 then

$$\frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} \exp\left(-\frac{(z - \mu)^2}{2\sigma^2}\right) dz = 1. \quad (\text{D.2})$$

By setting $c = 1/(2\sigma^2)$ we get

$$\int_{-\infty}^{+\infty} \exp\left(-c(z - \mu)^2\right) dz = \sqrt{\frac{\pi}{c}}. \quad (\text{D.3})$$

Rewriting (D.1) to comply with the form of (D.3) and setting $c = \alpha$ results in:

$$\begin{aligned} I_1(\alpha, \beta, \gamma) &= \int_{-\infty}^{+\infty} \exp\left(-(\alpha z^2 + \beta z + \gamma)\right) dz \\ &= e^{-\gamma} \int_{-\infty}^{+\infty} \exp\left(-(\alpha z^2 + \beta z)\right) dz \\ &= \exp\left(\frac{\beta^2}{4\alpha} - \gamma\right) \int_{-\infty}^{+\infty} \exp\left(-\alpha\left(z + \frac{\beta}{2\alpha}\right)^2\right) dz \\ &= \sqrt{\frac{\pi}{\alpha}} \exp\left(\frac{\beta^2}{4\alpha} - \gamma\right), \quad \alpha > 0. \end{aligned} \quad (\text{D.4})$$

Next, consider the two-dimensional Gaussian integral:

$$\begin{aligned}
I_2(A, B, C, D, E, F) &= \\
&\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \exp\left(-\left(Az_1^2 + Bz_1z_2 + Cz_2^2 + Dz_1 + Ez_2 + F\right)\right) dz_1 dz_2, \\
&A > 0, \quad 4AC - B^2 > 0.
\end{aligned} \tag{D.5}$$

Rewriting gi:i2 gives

$$I_2 = e^{-F} \int_{-\infty}^{+\infty} \exp\left(-\left(Cz_2^2 + Ez_2\right)\right) \left[\int_{-\infty}^{+\infty} \exp\left(-\left(Az_1^2 + (Bz_2 + D)z_1\right)\right) dz_1 \right] dz_2. \tag{D.6}$$

where the integral w.r.t. z_1 is denoted \tilde{I}_2 . Using gi:i1res with $\alpha = A$, $\beta = Bz_2 + D$ and $\gamma = 0$ results in:

$$\begin{aligned}
\tilde{I}_2 &= \sqrt{\frac{\pi}{A}} \exp\left(\frac{(Bz_2 + D)^2}{4A}\right) \\
&= \sqrt{\frac{\pi}{A}} \exp\left(\frac{B^2z_2^2 + 2BDz_2 + D^2}{4A}\right), \quad A > 0.
\end{aligned} \tag{D.7}$$

Substituting gi:i2tilde into (D.6) and using gi:i1res with

$$a = C - \frac{B^2}{4A}, \quad \beta = E - \frac{2BD}{4A}, \quad \gamma = 0$$

give

$$\begin{aligned}
I_2(A, B, C, D, E, F) &= \\
&= \sqrt{\frac{\pi}{A}} \exp\left(\frac{D^2}{4A} - F\right) \int_{-\infty}^{+\infty} \exp\left(-\left(C - \frac{B^2}{4A}\right)z_2^2 - \left(E - \frac{2BD}{4A}\right)z_2\right) dz_2 \\
&= \sqrt{\frac{\pi}{A}} \exp\left(\frac{D^2}{4A} - F\right) \sqrt{\frac{\pi}{C - \frac{B^2}{4A}}} \exp\left(\frac{\left(E - \frac{2BD}{4A}\right)^2}{4\left(C - \frac{B^2}{4A}\right)}\right) \\
&= \frac{\pi \exp\left(\frac{AE^2 + CD^2 - BDE}{4AC - B^2} - F\right)}{\sqrt{AC - \frac{B^2}{4}}}, \\
&A > 0, \quad 4AC - B^2 > 0.
\end{aligned} \tag{D.8}$$

D.2 Generalization Error in a Simple Neural Model

Consider a nonlinear system consisting of a single neuron given by:

$$\begin{aligned}
y(k) &= g(\mathbf{z}(k)) + \varepsilon(k) \\
&= h(w_1^\circ z_1(k) + w_2^\circ z_2(k)) + \varepsilon(k)
\end{aligned} \tag{D.9}$$

where

- $\mathbf{z}(k) = [z_1(k), z_2(k)]^\top$ is the two-dimensional input vector which is assumed to be marginally Gaussian distributed with zero mean vector and covariance matrix $\mathbf{\Sigma}$, i.e.,

$$\mathbf{z}(k) \in \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}), \quad \forall k \quad (\text{D.10})$$

with

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}.$$

- w_1°, w_2° are the true weights which – in this context – are assumed to be known.
- $h(\cdot)$ is the activation function of the neuron which is assumed to be:

$$h(u) = \exp\left(-\frac{(u - \nu)^2}{\eta^2}\right) - \exp\left(-\frac{(u + \nu)^2}{\eta^2}\right) \quad (\text{D.11})$$

where ν is a position parameter and η is a scale parameter.

- $\varepsilon(k)$ is the inherent noise with zero mean and variance σ_ε^2 . We assume that $\varepsilon(k)$ is independent of the input $\mathbf{z}(k)$.

The NN-model of gi:system is *incomplete*¹ and given by

$$y(k) = h(wz_1(k)) + e(k; w). \quad (\text{D.12})$$

The generalization error is by Ch. 6 given by

$$G(w) = E_{\mathbf{z}_t, \varepsilon_t} \left\{ e_t^2(w) \right\} \quad (\text{D.13})$$

where the expectation is w.r.t. $[\mathbf{z}_t, \varepsilon_t]$, i.e., a test sample independent of the training set. cf. gi:system and (D.12)²

$$\begin{aligned} G(w) &= E \left\{ e^2(w) \right\} \\ &= E \left\{ [\varepsilon + h(w_1^\circ z_1 + w_2^\circ z_2) - h(wz_1)]^2 \right\} \\ &= \sigma_\varepsilon^2 + E \left\{ [h(w_1^\circ z_1 + w_2^\circ z_2) - h(wz_1)]^2 \right\} \\ &= \sigma_\varepsilon^2 + \underbrace{E \left\{ h^2(w_1^\circ z_1 + w_2^\circ z_2) \right\}}_{G_1} + \underbrace{E \left\{ h^2(wz_1) \right\}}_{G_2} \\ &\quad - \underbrace{2E \left\{ h(w_1^\circ z_1 + w_2^\circ z_2) \cdot h(wz_1) \right\}}_{G_3}. \end{aligned} \quad (\text{D.14})$$

The individual terms $G_i, i \in \{1, 2, 3\}$ are now evaluated.

¹This is due to the fact that the model does not depend on $z_2(k)$.

²We omit the subindex t and the subindices of the expectation operator. Furthermore, we note that the model parameter w depends only on the training set *not* on the independent test sample.

D.2.1 The term G_1

Let $u = w_1^\circ z_1 + w_2^\circ z_2$ then:

$$G_1 = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h^2(u) p(z_1, z_2) dz_1 dz_2 \quad (\text{D.15})$$

where $p(z_1, z_2)$ is the joint p.d.f. of \mathbf{z} which according to the assumption gi:xprop is given by

$$p(z_1, z_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)} \left[\frac{z_1^2}{\sigma_1^2} - 2\frac{\rho}{\sigma_1\sigma_2} z_1 z_2 + \frac{z_2^2}{\sigma_2^2} \right]\right) \quad (\text{D.16})$$

where $\rho = \sigma_{12}/(\sigma_1\sigma_2)$. By defining

$$\begin{aligned} K &= \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \quad , \quad c_1 = \frac{1}{2(1-\rho^2)\sigma_1^2} \\ c_2 &= \frac{1}{2(1-\rho^2)\sigma_2^2} \quad , \quad c_{12} = \frac{-\rho}{(1-\rho^2)\sigma_1\sigma_2}. \end{aligned} \quad (\text{D.17})$$

gi:pdf is rewritten as:

$$p(z_1, z_2) = K \exp\left(-\left(c_1 z_1^2 + c_{12} z_1 z_2 + c_2 z_2^2\right)\right). \quad (\text{D.18})$$

Next we evaluate $h^2(u)$.

$$\begin{aligned} h^2(u) &= \left[\exp\left(-\frac{(u-\nu)^2}{\eta^2}\right) - \exp\left(-\frac{(u+\nu)^2}{\eta^2}\right) \right]^2 \\ &= \exp\left(-\frac{2(u-\nu)^2}{\eta^2}\right) + \exp\left(-\frac{2(u+\nu)^2}{\eta^2}\right) - 2\exp\left(-\frac{2(u^2+\nu^2)}{\eta^2}\right) \\ &= \sum_{n \in \{-1,1\}} \underbrace{\exp\left(-\frac{2(u+n\nu)^2}{\eta^2}\right)}_{T_{11}(n)} - 2 \underbrace{\exp\left(-\frac{2(u^2+\nu^2)}{\eta^2}\right)}_{T_{12}}. \end{aligned} \quad (\text{D.19})$$

Above the integer $n \in \{-1, 1\}$ is introduced for convenience. Substituting gi:g1terms and (D.18) into (D.15) results in that G_1 is expressed as a sum of three terms which all are two-dimensional Gaussian integrals. That is, by proper settings of the parameters in gi:i2res G_1 finally results. The parameters involving the terms $T_{11}(n)$ - which we will denote $\text{Par}[T_{11}(n)]$ - are given by:

$$\begin{aligned} A &= \frac{2(w_1^\circ)^2}{\eta^2} + c_1 \quad , \quad B = \frac{4w_1^\circ w_2^\circ}{\eta^2} + c_{12} \\ C &= \frac{2(w_2^\circ)^2}{\eta^2} + c_2 \quad , \quad D = \frac{4n\nu w_1^\circ}{\eta^2} \\ E &= \frac{4n\nu w_2^\circ}{\eta^2} \quad , \quad F = 2\frac{\nu^2}{\eta^2}. \end{aligned} \quad (\text{D.20})$$

and the parameters involving T_{12} , $\text{Par}[T_{12}]$, are the same as above except that $D = E = 0$. Finally,

$$G_1 = K \cdot \sum_{n \in \{-1,1\}} I_2(\text{Par}[T_{11}(n)]) - 2I_2(\text{Par}[T_{12}]). \quad (\text{D.21})$$

D.2.2 The term G_2

Let $u = wz_1$ then

$$G_2 = \int_{-\infty}^{+\infty} h^2(u)p(z_1) dz_1 \quad (\text{D.22})$$

where $p(z_1)$ is the marginal p.d.f. of z_1 which according to `gi:xprop` is given by

$$p(z_1) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{z_1^2}{2\sigma_1^2}\right). \quad (\text{D.23})$$

The result of `gi:g1terms` is valid for any definition of u . Consequently, using this expression (where the terms now are denoted $T_{21}(n)$ and T_{22}) and `gi:pdf1`, G_2 is given as a sum of three one-dimensional Gaussian integrals. The settings of the parameters in `gi:ilres` involving the terms $T_{21}(n)$, i.e., `Par[T11(n)]`, are:

$$\begin{aligned} \alpha &= \frac{2(w_1^\circ)^2}{\eta^2} + \frac{1}{2\sigma_1^2}, \\ \beta &= \frac{4n\nu w_1^\circ}{\eta^2}, \\ \gamma &= \frac{2\nu^2}{\eta^2}. \end{aligned} \quad (\text{D.24})$$

The parameters involving the term T_{22} is the same as above except that $\beta = 0$. Accordingly,

$$G_2 = \frac{1}{\sqrt{2\pi}\sigma_1} \cdot \sum_{n \in \{-1,1\}} I_1(\text{Par}[T_{21}(n)]) - 2I_1(\text{Par}[T_{22}]). \quad (\text{D.25})$$

D.2.3 The term G_3

Defining

$$\begin{aligned} u_1 &= w_1^\circ z_1 + w_2^\circ z_2, \\ u_2 &= wz_1 \end{aligned} \quad (\text{D.26})$$

G_3 becomes:

$$G_3 = -2 \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(u_1)h(u_2)p(z_1, z_2) dz_1 dz_2. \quad (\text{D.27})$$

Now the product $h(u_1)h(u_2)$ is evaluated:

$$\begin{aligned} h(u_1)h(u_2) &= \\ & \left[\exp\left(-\frac{(u_1 - \nu)^2}{\eta^2}\right) - \exp\left(-\frac{(u_1 + \nu)^2}{\eta^2}\right) \right] \cdot \left[\exp\left(-\frac{(u_2 - \nu)^2}{\eta^2}\right) - \exp\left(-\frac{(u_2 + \nu)^2}{\eta^2}\right) \right] \\ &= \sum_{n_1 \in \{-1,1\}} \sum_{n_2 \in \{-1,1\}} n_1 n_2 \underbrace{\exp\left(-\frac{(u_1 + n_1\nu)^2 + (u_2 + n_2\nu)^2}{\eta^2}\right)}_{T_3(n_1, n_2)}. \end{aligned} \quad (\text{D.28})$$

Substituting `gi:pdfre` and (D.28) into (D.27) result in two-dimensional Gaussian integrals with parameters, $\text{Par}[T_3(n_1, n_2)]$ (cf. `gi:i2res`):

$$\begin{aligned}
A &= \frac{(w_1^\circ)^2 + w^2}{\eta^2} + c_1 & , & & B &= \frac{2w_1^\circ w_2^\circ}{\eta^2} + c_{12} \\
C &= \frac{(w_2^\circ)^2}{\eta^2} + c_2 & , & & D &= \frac{2\nu(n_1 w_1^\circ + n_2 w)}{\eta^2} \\
E &= \frac{2n_1 \nu w_2^\circ}{\eta^2} & , & & F &= \frac{2\nu^2}{\eta^2}.
\end{aligned} \tag{D.29}$$

We note that only D and E depend on n_1 and n_2 . According to the expression of I_2 in `gi:i2res` the terms involving D and E are: D^2 and BDE . As $n_i^2 = 1$, $i \in \{1, 2\}$ then both terms depend on the product $n_1 n_2$ only. Consequently,

$$\begin{aligned}
G_3 &= -2K \cdot \sum_{n_1 \in \{-1, 1\}} \sum_{n_2 \in \{-1, 1\}} n_1 n_2 \cdot I_2(\text{Par}[T_3(n_1, n_2)]) \\
&= -4K \cdot \sum_{k \in \{-1, 1\}} n \cdot I_2(\text{Par}[T_3(n)])
\end{aligned} \tag{D.30}$$

where $n = n_1 n_2$.

D.3 Summary

In this appendix we presented an evaluation of one and two-dimensional Gaussian integrals for the purpose of calculating the generalization error. It was shown that it is possible to calculate the generalization error of a simple neural network model consisting of one neuron. The crucial assumptions made in the context were:

1. The input is Gaussian.
2. The activation function is a sum of two Gaussian functions.
3. The nonlinear system under consideration consists of a single neuron with known weights and with the activation function mentioned in item 2.

APPENDIX E

MOMENTS OF GAUSSIAN STOCHASTIC VECTORS

This appendix is devoted to evaluating the Hessian matrix of the expected cost function within a polynomial filter with Gaussian input vector signal. The result is used in Ch. 7 in conjunction with validation of the *GEN*-estimate.

E.1 The Hessian of a Polynomial Filter

Consider a l 'th order polynomial filter (cf. nf:polfilgen)

$$\begin{aligned} \hat{y}(k) = & h_0 + \sum_{n_1=1}^p h_1(n_1)P_1(z_{n_1}(k)) + \sum_{n_1=1}^p \sum_{\substack{n_2=1 \\ n_2 \neq n_1}}^p h_2(n_1, n_2)P_1(z_{n_1}(k))P_1(z_{n_2}(k)) + \\ & + \sum_{n_1=1}^p h_2(n_1, n_1)P_2(z_{n_1}(k)) + \cdots + \sum_{n_1=1}^p h_l(n_1, \dots, n_1)P_l(z_{n_1}(k)) \end{aligned} \quad (\text{E.1})$$

where $\mathbf{z}(k) = [z_1(k), z_2(k), \dots, z_p(k)]^\top$ is the input vector signal, $P_r(\cdot)$ is a polynomial of order r , and $h_r(n_1, n_2, \dots, n_r)$ is the r 'th order finite time-domain kernel. The filter is in a canonical filter representation simply expressed as:

$$\hat{y}(k) = \mathbf{w}^\top \mathbf{v}(k) \quad (\text{E.2})$$

where

$$\mathbf{w} = [h_0, h_1(1), h_2(1, 1), \dots, h_l(1, 1, \dots, 1), h_1(2), h_2(2, 1), \dots, h_l(p, p, \dots, p)]^\top, \quad (\text{E.3})$$

and (the time index k is omitted for simplicity)

$$\begin{aligned} \mathbf{v} = & [1, P_1(z_1), P_2(z_1), \dots, P_l(z_1), \\ & P_1(z_2), P_1(z_2)P_1(z_1), \dots, P_1(z_2)P_{l-1}(z_1), \\ & P_1(z_3), P_1(z_3)P_1(z_1), \dots, P_1(z_3)P_{l-1}(z_1), \\ & \vdots \\ & P_{l-1}(z_{p-1})P_1(z_p), \\ & P_l(z_p)]^\top \end{aligned} \quad (\text{E.4})$$

with dimension $q = C_{l+p,l}$ (cf. Ch. 3).

Since the polynomial filter is an LN-filter the Hessian matrix of the expected cost function is cf. Ch. 6 given by:

$$\mathbf{H} = E \left\{ \mathbf{v} \mathbf{v}^\top \right\} \quad (\text{E.5})$$

where the expectation is performed w.r.t. \mathbf{z} . The elements in this matrix are cf. gm:veqn of the form:

$$E \left\{ \prod_{i=1}^p P_{r_{i,1}}(z_i) \cdot P_{r_{i,2}}(z_i) \right\}$$

where

$$0 \leq \sum_{i=1}^p r_{i,n} \leq l, \quad n = 1, 2.$$

That is, the essential expectations which have to be calculated are:

$$E \left\{ \prod_{i=1}^p z_i^{r_i} \right\}$$

subject to the constraint:

$$0 \leq \sum_{i=1}^p r_i \leq 2l.$$

In general it is difficult to carry out these expectations; however, since \mathbf{z} is distributed according to a multi-dimensional Gaussian distribution the calculations turn out to be handy as shown below.

Provided that the essential expectations have been carried out it is possible to formulate an algorithm based on matrix operations which perform the construction of the Hessian, \mathbf{H} ; however, the details are omitted since they seem irrelevant in the present context.

E.2 Moment Calculations

Consider the case when \mathbf{z} is Gaussian distributed, i.e., $\mathbf{z} \in \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the covariance matrix. Accordingly, the p.d.f. becomes:

$$p(\mathbf{z}) = \frac{1}{\sqrt{(2\pi)^p \det(\boldsymbol{\Sigma})}} \exp \left(-\frac{1}{2} (\mathbf{z} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{z} - \boldsymbol{\mu}) \right). \quad (\text{E.6})$$

Using the properties of *moment generating functions*, e.g., [Bendat & Piersol 86, Ch. 3.2.3], gives:

$$E \left\{ \prod_{i=1}^p z_i^{r_i} \right\} = \left. \frac{\partial^{r_1} \partial^{r_2} \dots \partial^{r_p} m(\mathbf{t})}{\partial t_1^{r_1} \partial t_2^{r_2} \dots \partial t_p^{r_p}} \right|_{\mathbf{t}=\mathbf{0}} \quad (\text{E.7})$$

where $m(\mathbf{t})$ is the moment generating function defined by

$$\begin{aligned} m(\mathbf{t}) &= E \left\{ \exp(\mathbf{t}^\top \mathbf{z}) \right\} \\ &= \int_{\mathbb{R}^p} \exp(\mathbf{t}^\top \mathbf{z}) p(\mathbf{z}) d\mathbf{z}. \end{aligned} \quad (\text{E.8})$$

where $\mathbf{t} = [t_1, t_2, \dots, t_p]^\top$ is a p -dimensional auxiliary vector variable. Due to the exponential shape of the Gaussian distribution function gm:gaus the integral gm:gi is fairly easy to calculate (see also App. D).

Now consider the special case of a two-dimensional input, i.e., $p = 2$ and an $l = 4$ order polynomial filter; that is, $q = C_{6,4} = 15$. Assuming that the mean vector equals zero (i.e., $\boldsymbol{\mu} = \mathbf{0}$) the moment generating function becomes [Bendat & Piersol 86, Ch. 3.3]:

$$m(\mathbf{t}) = \exp\left(\frac{\sigma_1^2 t_1^2}{2} + \sigma_{12} t_1 t_2 + \frac{\sigma_2^2 t_2^2}{2}\right) \quad (\text{E.9})$$

where we used the notation:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}. \quad (\text{E.10})$$

When $r_1 + r_2$ is odd it is evident that

$$E\{z_1^{r_1} z_2^{r_2}\} = \int_{\mathbb{R}^2} z_1^{r_1} z_2^{r_2} p(\mathbf{z}) d\mathbf{z} = 0 \quad (\text{E.11})$$

since the integrand is an odd function (recall that $p(\mathbf{z})$ is an even function). As $0 \leq r_1 + r_2 \leq 2l = 8$ the remaining non-zero terms become due to gm:prod and (E.9)¹:

Zero Order

$$E\{1\} = 1.$$

Second Order

$$\begin{aligned} E\{z_1^2\} &= \sigma_1^2, \\ E\{z_1 z_2\} &= \sigma_{12}, \\ E\{z_2^2\} &= \sigma_2^2. \end{aligned}$$

Fourth Order

$$\begin{aligned} E\{z_1^4\} &= 3\sigma_1^4, \\ E\{z_1^3 z_2\} &= 3\sigma_1^2 \sigma_{12}, \\ E\{z_1^2 z_2^2\} &= 2\sigma_{12}^2 + \sigma_1^2 \sigma_2^2, \\ E\{z_1 z_2^3\} &= 3\sigma_{12} \sigma_2^2, \\ E\{z_2^4\} &= 3\sigma_2^4. \end{aligned}$$

Sixth Order

$$\begin{aligned} E\{z_1^6\} &= 15\sigma_1^6, \\ E\{z_1^5 z_2\} &= 15\sigma_1^4 \sigma_{12}, \\ E\{z_1^4 z_2^2\} &= 12\sigma_1^2 \sigma_{12}^2 + 3\sigma_1^4 \sigma_2^2, \\ E\{z_1^3 z_2^3\} &= 6\sigma_{12}^3 + 9\sigma_1^2 \sigma_{12} \sigma_2^2, \\ E\{z_1^2 z_2^4\} &= 12\sigma_{12}^2 \sigma_2^2 + 3\sigma_1^2 \sigma_2^4, \\ E\{z_1 z_2^5\} &= 15\sigma_{12} \sigma_2^4, \\ E\{z_2^6\} &= 15\sigma_2^6. \end{aligned}$$

¹The calculations are performed with the software package: "Mathematica" from Wolfram Research Inc., 100 Trade Center Drive, Champaign, Illinois, 61820-7237, USA.

Eighth Order

$$\begin{aligned}E\{z_1^8\} &= 105\sigma_1^8, \\E\{z_1^7 z_2\} &= 105\sigma_1^6 \sigma_{12}, \\E\{z_1^6 z_2^2\} &= 90\sigma_1^4 \sigma_{12}^2 + 15\sigma_1^6 \sigma_2^2, \\E\{z_1^5 z_3^3\} &= 60\sigma_1^2 \sigma_{12}^3 + 45\sigma_1^4 \sigma_{12} \sigma_2^2, \\E\{z_1^4 z_2^4\} &= 24\sigma_{12}^4 + 72\sigma_1^2 \sigma_{12}^2 \sigma_2^2 + 9\sigma_1^4 \sigma_2^4, \\E\{z_1^3 z_2^5\} &= 60\sigma_{12}^3 \sigma_2^2 + 45\sigma_1^2 \sigma_{12} \sigma_2^4, \\E\{z_1^2 z_2^6\} &= 90\sigma_{12}^2 \sigma_2^4 + 15\sigma_1^2 \sigma_2^6, \\E\{z_1 z_2^7\} &= 105\sigma_{12} \sigma_2^6, \\E\{z_2^8\} &= 105\sigma_2^8.\end{aligned}$$

APPENDIX F

STUDIES OF THE WEIGHT FLUCTUATION PENALTY

In this appendix the weight fluctuation penalty (*WFP*), mentioned in Ch. 6, will be studied in two simple cases. The aim is in the first case to show that the *WFP* of a complex model may be less than the *WFP* of a simple model. This fact contradicts the immediate intuition that *WFP* increases with model complexity.

The aim of the second case is to demonstrate that if some weights in a model optimally (i.e., applying a infinite training set) are equal to zero then it should be better to omit those weights as this reduces the *WFP*. A fundamental prerequisite if of course that it is possible to detect which of the weights that should be set to zero. This is treated in Sec. 6.6 and Sec. 6.7.

F.1 On the Changes in *WFP* Due to Model Complexity

Consider the linear system:

$$y(k) = \mathbf{z}^\top(k)\mathbf{w}^\circ + \varepsilon(k) \quad (\text{F.1})$$

where $\mathbf{z}(k) = [z_1(k), z_2(k)]^\top$ is a two-dimensional stochastic Gaussian i.i.d.¹ signal with zero mean and covariance matrix

$$\mathbf{H} = E \left\{ \mathbf{z}(k)\mathbf{z}^\top(k) \right\} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}. \quad (\text{F.2})$$

$\varepsilon(k)$ is an i.i.d. inherent noise sequence with zero mean and variance σ_ε^2 . In addition, $\varepsilon(k)$ is independent of $\mathbf{z}(k)$. Finally, $\mathbf{w}^\circ = [w_1^\circ, w_2^\circ]^\top$ is the true weight vector.

Now consider two models of the linear system: The incomplete model (IM) (the $z_2(k)$ signal does not enter the model)

$$y(k) = w_1 z_1(k) + e(k), \quad (\text{F.3})$$

and the complete model (CM)

$$y(k) = \mathbf{z}^\top(k)\mathbf{w} + e(k). \quad (\text{F.4})$$

The aim is to show that the *WFP* may be less when dealing with the complete model (the most complex model) than dealing with the incomplete model.

¹That is the samples of the vector signal are independent, i.e., $\mathbf{z}(k_1)$ is independent of $\mathbf{z}(k_2)$, $\forall k_1 \neq k_2$.

The weights are estimated on a training set, $\mathcal{T} = \{z(k); y(k)\}$, with N examples using the LS cost function. Since the models are LL-models the weight estimates result from solving the normal equations, cf. Ch. 5.

The IM Case The estimated weight, \hat{w}_1 , and the optimal weight, w_1^* , are in the IM case given by:

$$\begin{aligned}
\hat{w}_1 &= \left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^{-1} \frac{1}{N} \sum_{k=1}^N z_1(k)y(k) \\
&= \left[\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right]^{-1} \frac{1}{N} \sum_{k=1}^N z_1(k) (w_1^\circ z_1(k) + w_2^\circ z_2(k) + \varepsilon(k)) \\
&= w_1^\circ + w_2^\circ \left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^{-1} \frac{1}{N} \sum_{k=1}^N z_1(k)z_2(k) \\
&\quad + \left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^{-1} \frac{1}{N} \sum_{k=1}^N z_1(k)\varepsilon(k), \tag{F.5}
\end{aligned}$$

$$\begin{aligned}
w_1^* &= \left(E \{ z_1^2 \} \right)^{-1} E \{ z_1 y \} \\
&= \frac{1}{\sigma_1^2} \cdot E \{ z_1 (w_1^\circ z_1 + w_2^\circ z_2 + \varepsilon) \} \\
&= w_1^\circ + w_2^\circ \frac{\sigma_{12}}{\sigma_1^2}. \tag{F.6}
\end{aligned}$$

According to sa:wfplin the *WFP* becomes²

$$WFP = E \{ z_1^2 \} E_{\mathcal{T}} \{ (\Delta w_1)^2 \} = \sigma_1^2 \cdot E_{\mathcal{T}} \{ (\Delta w_1)^2 \}.$$
 \tag{F.7}

Using wf:ex1whatim and (F.6) we get:

$$\begin{aligned}
\Delta w_1 &= \hat{w}_1 - w_1^* \\
&= w_2^\circ \left(\frac{\frac{1}{N} \sum_{k=1}^N z_1(k)z_2(k)}{\frac{1}{N} \sum_{k=1}^N z_1^2(k)} - \frac{\sigma_{12}}{\sigma_1^2} \right) + \frac{\frac{1}{N} \sum_{k=1}^N z_1(k)\varepsilon(k)}{\frac{1}{N} \sum_{k=1}^N z_1^2(k)}, \tag{F.8}
\end{aligned}$$

and consequently

$$(\Delta w_1)^2 = \left. \frac{\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N z_1(k_1)\varepsilon(k_1)z_1(k_2)\varepsilon(k_2)}{\left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^2} \right\} L_1$$

²Since the model is one-dimensional the matrices in sa:wfplin reduce to scalars and consequently, the trace operator is superfluous.

$$\begin{aligned}
&= \left. + (w_2^\circ)^2 \cdot \frac{\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N z_1(k_1)z_2(k_1)z_1(k_2)z_2(k_2)}{\left(\frac{1}{N} \sum_{k=1}^N z_1^2(k)\right)^2} \right\} L_2 \\
&= \left. -2(w_2^\circ)^2 \cdot \frac{\sigma_{12}}{\sigma_1^2} \cdot \frac{\frac{1}{N} \sum_{k=1}^N z_1(k)z_2(k)}{\frac{1}{N} \sum_{k=1}^N z_1^2(k)} \right\} L_3 \\
&= \left. + (w_2^\circ)^2 \frac{\sigma_{12}^2}{\sigma_1^4} \right\} L_4 \\
&= \left. +2w_2^\circ \left(\frac{\frac{1}{N} \sum_{k=1}^N z_1(k)z_2(k)}{\frac{1}{N} \sum_{k=1}^N z_1^2(k)} - \frac{\sigma_{12}}{\sigma_1^2} \right) \cdot \frac{\frac{1}{N} \sum_{k=1}^N z_1(k)\varepsilon(k)}{\frac{1}{N} \sum_{k=1}^N z_1^2(k)} \right\} L_5. \quad (\text{F.9})
\end{aligned}$$

The expectation w.r.t. the training set, \mathcal{T} , is now performed on the individual components, L_1, \dots, L_5 , in wf:dw2im:

1. The expectation w.r.t. the training set is carried out by noting that

$$E_{\mathcal{T}}\{\cdot\} = E_{\{\mathbf{z}(k), \varepsilon(k)\}}\{\cdot\} = E_{\{\mathbf{z}(k)\}} \left\{ E_{\{\varepsilon(k)\}} \left\{ \cdot \mid \{\mathbf{z}(k)\} \right\} \right\}. \quad (\text{F.10})$$

Since $\varepsilon(k)$ is an i.i.d. sequence the expectation w.r.t. $\varepsilon(k)$ in L_1 only contributes when $k_1 = k_2$. That is,

$$\frac{\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N z_1(k_1)\varepsilon(k_1)z_1(k_2)\varepsilon(k_2)}{\left(\frac{1}{N} \sum_{k=1}^N z_1^2(k)\right)^2} = \frac{\sigma_\varepsilon^2}{N} E_{\{\mathbf{z}_1(k)\}} \left\{ \left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^{-1} \right\}. \quad (\text{F.11})$$

The expectation of the inverse sum of the squared $z_1(k)$ signal is in principle possible to perform as $z_1(k)$ is assumed to be Gaussian distributed. However, we shall apply the approximate results given in App. B on expectations of this type. Accordingly:

$$E_{\{\mathbf{z}_1(k)\}} \left\{ \left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^{-1} \right\} \approx \sigma_1^{-2}, \quad N \rightarrow \infty. \quad (\text{F.12})$$

To sum up:

$$L_1 = \frac{\sigma_\varepsilon^2}{N\sigma_1^2}. \quad (\text{F.13})$$

2. The component L_2 depends only on the input signal $z_1(k)$. Now, applying the result of App. B the denominator of the second component yields:

$$\left(\frac{1}{N} \sum_{k=1}^N z_1^2(k) \right)^{-2} \approx \sigma_1^{-4}, \quad N \rightarrow \infty. \quad (\text{F.14})$$

That is, we concentrate on the expectation of the numerator; in particular, the expectation operator goes inside the summation signs. As $\mathbf{z}(k)$ is an i.i.d. signal then when $k_1 \neq k_2$ ³,

$$E \{z_1(k_1)z_2(k_1)z_1(k_2)z_2(k_2)\} = E \{z_1(k_1)z_2(k_1)\} E \{z_1(k_2)z_2(k_2)\} = \sigma_{12}^2, \quad (\text{F.15})$$

whereas when $k_1 = k_2$,

$$E \{z_1(k_1)z_2(k_1)z_1(k_2)z_2(k_2)\} = E \{z_1^2(k)z_2^2(k)\} = 2\sigma_{12}^2 + \sigma_1^2\sigma_2^2. \quad (\text{F.16})$$

The last equality is based on the Gaussian assumption on the input distribution and further elaborated in App. E.

Now summing over k_1 and k_2 in the interval $[1; N]$ result in $N^2 - N = N(N - 1)$ terms with $k_1 \neq k_2$ and N terms where $k_1 = k_2$. Consequently, the final expression of L_2 is:

$$\begin{aligned} L_2 &= (w_2^\circ)^2 \sigma_1^{-4} \cdot \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N E_{\{\mathbf{z}(k)\}} \{z_1(k_1)z_2(k_1)z_1(k_2)z_2(k_2)\} \\ &= (w_2^\circ)^2 \sigma_1^{-4} \left[\sigma_{12}^2 \frac{N(N-1)}{N^2} + \frac{N(2\sigma_{12}^2 + \sigma_1^2\sigma_2^2)}{N^2} \right] \\ &= (w_2^\circ)^2 \sigma_1^{-4} \left[\sigma_{12}^2 + \frac{\sigma_{12}^2 + \sigma_1^2\sigma_2^2}{N} \right]. \end{aligned} \quad (\text{F.17})$$

3. The component L_3 is easily treated since the denominator in the last factor as in the first item is approximated by σ_1^2 as $N \rightarrow \infty$. The expectation of the numerator of the last factor simply becomes: σ_{12} . In summary:

$$L_3 = -2(w_2^\circ)^2 \cdot \frac{\sigma_{12}^2}{\sigma_1^4}. \quad (\text{F.18})$$

4. The component L_4 does not depend on the training set at all.
5. The component L_5 vanishes since the expectation w.r.t. $\varepsilon(k)$ equals zero. This is due to the fact that only $\varepsilon(k)$ to the first power enters the expression and that the mean value is zero per definition.

Adding the individual contributions to wf:dw2im and substituting into wf:ex1wfpim result in:

$$WFP = \frac{\sigma_\varepsilon^2 + (w_2^\circ)^2 \cdot (\sigma_{12}^2 + \sigma_1^2\sigma_2^2)}{N}. \quad (\text{F.19})$$

Note that $WFP \rightarrow 0$ as $N \rightarrow \infty$ which stems from the fact that $\hat{w}_1 \rightarrow w_1^*$ as $N \rightarrow \infty$.

The CM Case In the CM case we proceed similarly. Define

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)\mathbf{z}^\top(k), \quad (\text{F.20})$$

³The subindex on the expectation operator is omitted for simplicity.

then in the CM case the estimated weight vector yields:

$$\begin{aligned}
\hat{\mathbf{w}} &= \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) y(k) \\
&= \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \left(\mathbf{z}^\top(k) \mathbf{w}^\circ + \varepsilon(k) \right) \\
&= \mathbf{w}^\circ + \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \varepsilon(k),
\end{aligned} \tag{F.21}$$

$$\begin{aligned}
\mathbf{w}^* &= \mathbf{H}^{-1} E \{ \mathbf{z} y \} \\
&= \mathbf{H}^{-1} E \left\{ \mathbf{z} \left(\mathbf{z}^\top \mathbf{w}^\circ + \varepsilon \right) \right\} \\
&= \mathbf{w}^\circ.
\end{aligned} \tag{F.22}$$

The last result is an obvious consequence of using a complete model.

According to sa:wfplin the *WFP* becomes:

$$WFP = \text{tr} \left[E \left\{ \mathbf{z} \mathbf{z}^\top \right\} E_T \left\{ \Delta \mathbf{w} \Delta \mathbf{w}^\top \right\} \right] = \text{tr} \left[\mathbf{H} E_T \left\{ \Delta \mathbf{w} \Delta \mathbf{w}^\top \right\} \right]. \tag{F.23}$$

From wf:ex1whatcm and (F.22)

$$\Delta \mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^* = \mathbf{H}_N^{-1} \sum_{k=1}^N \mathbf{z}(k) \varepsilon(k). \tag{F.24}$$

Accordingly:

$$\Delta \mathbf{w} \Delta \mathbf{w}^\top = \mathbf{H}_N^{-1} \left[\frac{1}{N^2} \sum_{k_1=1}^N \mathbf{z}(k_1) \varepsilon(k_1) \mathbf{z}(k_2) \varepsilon(k_2) \right] \mathbf{H}_N^{-1}. \tag{F.25}$$

By first performing the expectation of wf:dw2cm w.r.t. $\varepsilon(k)$ (cf. wf:expcnd) and noting that $\varepsilon(k)$ is an i.i.d. sequence then

$$\begin{aligned}
E_T \left\{ \Delta \mathbf{w} \Delta \mathbf{w}^\top \right\} &= \frac{\sigma_\varepsilon^2}{N} \cdot E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{H}_N^{-1} \left[\frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \right] \mathbf{H}_N^{-1} \right\} \\
&= \frac{\sigma_\varepsilon^2}{N} \cdot E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{H}_N^{-1} \right\}.
\end{aligned} \tag{F.26}$$

Applying the result of App. B we get

$$E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{H}_N^{-1} \right\} \approx \mathbf{H}^{-1}, \quad N \rightarrow \infty \tag{F.27}$$

Finally, substituting into wf:wfpcm

$$WFP = \frac{\sigma_\varepsilon^2}{N} \cdot \text{tr} \left[\mathbf{H} \mathbf{H}^{-1} \right] = \frac{2\sigma_\varepsilon^2}{N}. \tag{F.28}$$

Comparing the *WFP*'s in the IM and CM cases (cf. wf:wfpcmfin, (F.19)) we notice that if

$$\frac{2\sigma_\varepsilon^2}{N} < \frac{\sigma_\varepsilon^2 + (w_2^\circ)^2 \cdot (\sigma_{12}^2 + \sigma_1^2 \sigma_2^2)}{N} \tag{F.29}$$

⇕

$$\sigma_\varepsilon^2 < (w_2^\circ)^2 \cdot (\sigma_{12}^2 + \sigma_1^2 \sigma_2^2), \quad (\text{F.30})$$

then the *WFP* of the IM will be less than the *WFP* of the CM although the CM is more complex than the IM. It is worth noting that the result is highly dependent on a number of facts concerning the system and the models. For instance the noise variance, the input distribution, the true weights, etc. Consequently, in general, one is really not capable of providing any firm statements on this topic.

F.2 The *WFP* when Dealing with Insignificant Weights

Consider the linear system:

$$y(k) = w_1^\circ + w_2^\circ x(k) + \varepsilon(k) \quad (\text{F.31})$$

where $x(k) \in \mathcal{N}(0, \sigma_x^2)$ is an i.i.d. input signal, $\varepsilon(k)$ is an i.i.d. inherent noise sequence with zero mean and variance σ_ε^2 . In addition, $\varepsilon(k)$ is independent of $x(k)$. Finally, w_1°, w_2° are the true weights.

Now consider two incomplete models of the linear system. The first model, called the unrestricted model, obeys:

$$\begin{aligned} y(k) &= w_1 + w_3 x^2(k) + e(k) \\ &= \mathbf{w}^\top \mathbf{z}(k) + e(k; \mathbf{w}) \end{aligned} \quad (\text{F.32})$$

where $\mathbf{w} = [w_1, w_3]^\top$, $\mathbf{z} = [1, x^2(k)]^\top$, and $e(k; \mathbf{w})$ is the error signal. The model is obviously incomplete since no linear term, i.e., $x(k)$, is present. On the other hand the model contains a quadratic term, $x^2(k)$ which not is present in the system. Moreover, since $x(k)$ is a zero mean Gaussian signal $x(k)$ is uncorrelated with $x^2(k)$. This implies (as elaborated below) that the optimal setting of w_3 w.r.t. the LS cost function⁴ is $w_3^* = 0$. That is, none of the systematic error⁵ is explained by including the $x^2(k)$ term. Due to the finite training set w_3 is, in general, assigned a non-zero value which results in a contribution to the *WFP*. Intuitively it may be better then not including this term after all. This leads to the consideration of the restricted model given by:

$$y(k) = w_1 + e(k; w_1). \quad (\text{F.33})$$

Since the optimal setting of w_3 in the unrestricted model equals zero the *MSME*'s of the two models are identical. That is, in the limit of an infinite training set the generalization ability is equally good. The goal is now to demonstrate that WFP_u of the unrestricted model is larger than WFP_r of the restricted model.

F.2.1 *WFP* of the Unrestricted Model

According to sa:wfplin the *WFP* of a LX-model, $y(k) = \mathbf{w}^\top \mathbf{z}(k) + e(k; \mathbf{w})$, equals:

$$WFP = \text{tr}[\mathbf{H}\mathbf{V}] \quad (\text{F.34})$$

⁴Note that the optimal setting depends on the chosen cost function and the statistic of the input. That is, if $x(k)$ is not Gaussian one may profit by including the $x^2(k)$ term.

⁵That is, the part of the error signal $e(k; \mathbf{w})$ which depends on the input.

where the Hessian $\mathbf{H} = E\{\mathbf{z}(k)\mathbf{z}^\top(k)\}$ and the covariance matrix $\mathbf{V} = E_{\mathcal{T}}\{\Delta\mathbf{w}\Delta\mathbf{w}^\top\}$. Here \mathcal{T} denotes expectation w.r.t. the training set and $\Delta\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}^*$ is the difference between the estimated and the optimal weight vector.

According to wf:ex2m1 and the results in App. E concerning moments of Gaussian variables we have

$$\mathbf{H} = E\left\{[1, x^2(k)] \cdot [1, x^2(k)]^\top\right\} = \begin{bmatrix} 1 & \sigma_x^2 \\ \sigma_x^2 & 3\sigma_x^4 \end{bmatrix}. \quad (\text{F.35})$$

The inverse Hessian is accordingly:

$$\mathbf{H}^{-1} = \begin{bmatrix} \frac{3}{2} & -\frac{1}{2}\sigma_x^{-2} \\ -\frac{1}{2}\sigma_x^{-2} & \frac{1}{2}\sigma_x^{-4} \end{bmatrix}. \quad (\text{F.36})$$

Employing the LS cost function cf. Ch. 5 results in that the optimal weight vector becomes:

$$\begin{aligned} \mathbf{w}^* &= \mathbf{H}^{-1}E\{y(k)\mathbf{z}(k)\} \\ &= \mathbf{H}^{-1}E\{(w_1^\circ + w_2^\circ x(k) + \varepsilon(k)) [1, x^2(k)]^\top\} \\ &= \mathbf{H}^{-1} [w_1^\circ, w_1^\circ \sigma_x^2]^\top \\ &= [w_1^\circ, 0]^\top. \end{aligned} \quad (\text{F.37})$$

That is, the optimal setting of w_3 equals zero.

The estimated weight is given by:

$$\hat{\mathbf{w}} = \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N y(k)\mathbf{z}(k) \quad (\text{F.38})$$

where the Hessian $\mathbf{H}_N = N^{-1} \sum_{k=1}^N \mathbf{z}(k)\mathbf{z}^\top(k)$ and the training set equals $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, $k = 1, 2, \dots, N$. This equation is conveniently rewritten by replacing $y(k)$ with the expression obtained by evaluating the model wf:ex2m1 at the optimal weights, i.e.,

$$\begin{aligned} \hat{\mathbf{w}} &= \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N \left(\mathbf{z}^\top(k)\mathbf{w}^* + e(k; \mathbf{w}^*) \right) \mathbf{z}(k) \\ &= \mathbf{w}^* + \mathbf{H}_N^{-1} \frac{1}{N} \sum_{k=1}^N e(k; \mathbf{w}^*) \mathbf{z}(k). \end{aligned} \quad (\text{F.39})$$

Hence, the covariance matrix, \mathbf{V} , is given by

$$\begin{aligned} \mathbf{V} &= E_{\mathcal{T}}\{\Delta\mathbf{w}\Delta\mathbf{w}^\top\} \\ &= E_{\mathcal{T}} \left\{ \mathbf{H}_N^{-1} \left[\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \mathbf{z}(k_1)\mathbf{z}^\top(k_2) e(k_1; \mathbf{w}^*) e(k_2; \mathbf{w}^*) \right] \mathbf{H}_N^{-1} \right\}. \end{aligned} \quad (\text{F.40})$$

In order to evaluate the expectation we use the Hessian approximation $\mathbf{H}_N^{-1} \approx \mathbf{H}^{-1}$, $N \gg 1$ cf. App. B. Furthermore, noting that both $x(k)$ and $\varepsilon(k)$ are i.i.d. sequences results in that the expectation w.r.t. the double sum only contributes when $k_1 = k_2$. In consequence,

$$\begin{aligned} \mathbf{V} &= \frac{1}{N} \mathbf{H}^{-1} E_{\mathcal{T}} \left\{ \mathbf{z}(k)\mathbf{z}^\top(k) e^2(k; \mathbf{w}^*) \right\} \mathbf{H}^{-1} \\ &= \frac{1}{N} \mathbf{H}^{-1} \mathbf{R} \mathbf{H}^{-1}. \end{aligned} \quad (\text{F.41})$$

The optimal error $e(k; \mathbf{w}^*)$ is cf. wf:ex2sys, (F.32)

$$\begin{aligned} e(k; \mathbf{w}^*) &= y(k) - \mathbf{z}^\top(k) \mathbf{w}^* \\ &= w_1^\circ + w_2^\circ x(k) + \varepsilon(k) - w_1^\circ \\ &= w_2^\circ x(k) + \varepsilon(k). \end{aligned} \quad (\text{F.42})$$

The squared optimal error yields accordingly

$$e^2(k; \mathbf{w}^*) = (w_2^\circ)^2 x^2(k) + 2w_2^\circ x(k)\varepsilon(k) + \varepsilon^2(k). \quad (\text{F.43})$$

Recall from wf:expcnd that the expectation w.r.t. to the training set involves expectation w.r.t. to both $x(k)$ and $\varepsilon(k)$. According to wf:expcnd this expectation can be carried out in two steps. Now cf. wf:ex2m1v and (F.43) we have⁶

$$\begin{aligned} \mathbf{R} &= E_{\mathcal{T}} \left\{ \begin{bmatrix} 1 & x^2(k) \\ x^2(k) & x^4(k) \end{bmatrix} \cdot \left((w_2^\circ)^2 x^2(k) + 2w_2^\circ x(k)\varepsilon(k) + \varepsilon^2(k) \right) \right\} \\ &= E_{\{x(k)\}} \left\{ \begin{bmatrix} (w_2^\circ)^2 x^2(k) + \sigma_\varepsilon^2 & (w_2^\circ)^2 x^4(k) + \sigma_\varepsilon^2 x^2(k) \\ (w_2^\circ)^2 x^4(k) + \sigma_\varepsilon^2 x^2(k) & (w_2^\circ)^2 x^6(k) + \sigma_\varepsilon^2 x^4(k) \end{bmatrix} \right\} \\ &= \begin{bmatrix} (w_2^\circ)^2 \sigma_x^2 + \sigma_\varepsilon^2 & 3(w_2^\circ)^2 \sigma_x^4 + \sigma_x^2 \sigma_\varepsilon^2 \\ 3(w_2^\circ)^2 \sigma_x^4 + \sigma_x^2 \sigma_\varepsilon^2 & 15(w_2^\circ)^2 \sigma_x^6 + 3\sigma_x^4 \sigma_\varepsilon^2 \end{bmatrix}. \end{aligned} \quad (\text{F.44})$$

Substituting this result into wf:ex2m1v, applying wf:ex2invh and finally substituting the covariance expression into wf:wfpexp yields after some algebraic manipulations

$$WFP_u = \frac{6(w_2^\circ)^2 \sigma_x^2 + 2\sigma_\varepsilon^2}{N}. \quad (\text{F.45})$$

F.2.2 WFP of the Restricted Model

According to the restricted model wf:ex2m2 the Hessians, H_N , H (which are scalars in this case) equals unity. Consequently, the optimal weight estimate becomes:

$$w_1^* = E\{y(k)\} = w_1^\circ, \quad (\text{F.46})$$

and the estimated weight

$$\begin{aligned} \hat{w}_1 &= \frac{1}{N} \sum_{k=1}^N y(k) \\ &= \frac{1}{N} \sum_{k=1}^N w_1^\circ + w_2^\circ x(k) + \varepsilon(k). \end{aligned} \quad (\text{F.47})$$

The variance of the weight estimate thus equals:

$$\begin{aligned} V &= E_{\mathcal{T}}\{(\Delta w_1)^2\} \\ &= E_{\mathcal{T}} \left\{ \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N [w_2^\circ x(k_1) + \varepsilon(k_1)] \cdot [w_2^\circ x(k_2) + \varepsilon(k_2)] \right\} \end{aligned}$$

⁶Below the results concerning moments of Gaussian variables, cf. App. E, are used.

$$\begin{aligned}
&= \frac{1}{N} E_{\{x(k)\}} \left\{ (w_2^\circ)^2 x^2(k) + \sigma_\varepsilon^2 \right\} \\
&= \frac{(w_2^\circ)^2 \sigma_x^2 + \sigma_\varepsilon^2}{N}.
\end{aligned} \tag{F.48}$$

Finally according to wf:wpexp

$$WFP_r = V = \frac{(w_2^\circ)^2 \sigma_x^2 + \sigma_\varepsilon^2}{N}. \tag{F.49}$$

The result is then: $WFP_u \geq WFP_r$, since all involved terms are positive. Consequently, the average generalization error of the restricted model wf:ex2m2 is better than that of the unrestricted model wf:ex2m1. Consequently, if some weight elimination procedure cf. Sec. 6.6 and Sec. 6.7 leads to the conclusion that a weight (optimally) can be removed then the restricted model should be used.

APPENDIX G

REDUCING GENERALIZATION ERROR BY REGULARIZATION

In this appendix it will be demonstrated that a weight decay regularization can reduce the generalization error when employing a complete LX-model. The reason is that regularization controls the trade off between the mean square model error (*MSME*) and the weight fluctuation penalty *WFP*.

G.1 System and Model

Consider the linear system:

$$y(k) = \mathbf{z}^\top(k)\mathbf{w}^\circ + \varepsilon(k) \quad (\text{G.1})$$

where $\mathbf{z}(k)$ is the i.i.d. m -dimensional stochastic input vector signal, \mathbf{w}° is the m -dimensional true weight vector, and $\varepsilon(k)$ is the i.i.d. zero mean inherent noise with variance σ_ε^2 . It is assumed that $\varepsilon(k)$ is independent of $\mathbf{z}(k)$. The model is complete and given by:

$$y(k) = \mathbf{z}^\top(k)\mathbf{w} + e(k; \mathbf{w}) \quad (\text{G.2})$$

where $e(k; \mathbf{w})$ is the error signal and \mathbf{w} the m -dimensional weight vector. In order to estimate the model from the training set, $\mathcal{T} = \{\mathbf{z}(k); y(k)\}$, we employ the LS cost function, $S_N(\mathbf{w})$, with a *weight decay regularizer*, i.e., the cost function becomes:

$$C_N(\mathbf{w}) = S_N(\mathbf{w}) + \kappa \mathbf{w}^\top \mathbf{w}. \quad (\text{G.3})$$

According to `pa:wdestlin` the estimated weights are given by:

$$\hat{\mathbf{w}} = \mathbf{J}_N^{-1} \sum_{k=1}^N \mathbf{z}(k)y(k) \quad (\text{G.4})$$

where \mathbf{J}_N is the Hessian matrix w.r.t. the cost function, $C_N(\mathbf{w})$, given by

$$\mathbf{J}_N = \mathbf{H}_N + \kappa \mathbf{I} \quad (\text{G.5})$$

and where

$$\mathbf{H}_N = \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)\mathbf{z}^\top(k). \quad (\text{G.6})$$

By substituting the system equation re:wdsys into re:wdwhat1 we get:

$$\begin{aligned}\hat{\mathbf{w}} &= \mathbf{J}_N^{-1} \sum_{k=1}^N \mathbf{z}(k) \left[\mathbf{z}^\top(k) \mathbf{w}^\circ + \varepsilon(k) \right] \\ &= \mathbf{J}_N^{-1} \mathbf{H}_N \mathbf{w}^\circ + \mathbf{J}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k) \varepsilon(k).\end{aligned}\quad (\text{G.7})$$

The optimal weights, \mathbf{w}^* which minimize the expected cost, $C(\mathbf{w})$, are cf. pa:wdwoptlin:

$$\begin{aligned}\mathbf{w}^* &= \mathbf{J}^{-1} E_{\mathbf{z}_t, \varepsilon_t} \{ \mathbf{z}_t y_t \} \\ &= \mathbf{J}^{-1} E_{\mathbf{z}_t, \varepsilon_t} \left\{ \mathbf{z}_t \left[\mathbf{z}_t^\top \mathbf{w}^\circ + \varepsilon_t \right] \right\} \\ &= \mathbf{J}^{-1} \mathbf{H} \mathbf{w}^\circ\end{aligned}\quad (\text{G.8})$$

where

$$\mathbf{J} = \mathbf{H} + \kappa \mathbf{I}. \quad (\text{G.9})$$

and $\mathbf{H} = E\{ \mathbf{z}_t \mathbf{z}_t^\top \}$.

These expressions enable us to study the *MSME* and the *WFP* in the present case.

G.2 Mean Square Model Error

The *MSME* is cf. sa:moderr when using re:wdwopt given by¹:

$$\begin{aligned}MSME &= E_{\mathbf{x}_t} \left\{ [g(\mathbf{x}_t) - f(\mathbf{x}_t; \mathbf{w}^*)]^2 \right\} \\ &= E_{\mathbf{z}_t} \left\{ \left[\mathbf{z}_t^\top \mathbf{w}^\circ - \mathbf{z}_t^\top \mathbf{w}^* \right]^2 \right\} \\ &= \text{tr} \left[E_{\mathbf{z}_t} \left\{ \mathbf{z}_t \mathbf{z}_t^\top \right\} (\mathbf{w}^* - \mathbf{w}^\circ) (\mathbf{w}^* - \mathbf{w}^\circ)^\top \right] \\ &= \text{tr} \left[\mathbf{H} \left(\mathbf{J}^{-1} \mathbf{H} - \mathbf{I} \right) \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \left(\mathbf{J}^{-1} \mathbf{H} - \mathbf{I} \right)^\top \right].\end{aligned}\quad (\text{G.10})$$

Consider the eigenvalue decomposition of \mathbf{H} ,

$$\mathbf{H} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^\top \quad (\text{G.11})$$

where

- $\mathbf{\Lambda}$ is the diagonal eigenvalue matrix

$$\mathbf{\Lambda} = \text{diag} \{ [\lambda_1, \lambda_2, \dots, \lambda_m] \} \quad (\text{G.12})$$

with λ_i being the i 'th eigenvalue. It is assumed that \mathbf{H}_N is positive definite, q.e., $\lambda_i > 0$, $i \in [1; m]$.

- \mathbf{Q} is the unitary matrix² of normalized eigenvectors, \mathbf{q}_i , $i = 1, 2, \dots, m$,

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m]. \quad (\text{G.13})$$

¹Below the rule: $\text{tr}[\mathbf{AB}] = \text{tr}[\mathbf{BA}]$ was used.

²A matrix \mathbf{Q} is said to be unitary if $\mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$, q.e., $\mathbf{Q}^{-1} = \mathbf{Q}^\top$.

The inverse of the Hessian \mathbf{J} is then:

$$\begin{aligned}\mathbf{J}^{-1} &= (\mathbf{H} + \kappa \mathbf{I})^{-1} \\ &= (\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top + \kappa\mathbf{Q}\mathbf{Q}^\top)^{-1} \\ &= \mathbf{Q}(\mathbf{\Lambda} + \kappa\mathbf{I})^{-1}\mathbf{Q}^\top.\end{aligned}\tag{G.14}$$

Consider next the linear transformation of the true weights, \mathbf{w}° , into the basis where \mathbf{H} becomes diagonal³, i.e.,

$$\boldsymbol{\omega}^\circ = \mathbf{Q}^\top \mathbf{w}^\circ\tag{G.15}$$

where $\boldsymbol{\omega}^\circ$ are the transformed true weights. Now, substituting re:hdecom, (G.14) and (G.15) into re:wdmsme1 yields⁴:

$$\begin{aligned}MSME &= \text{tr} \left[\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top \left(\mathbf{Q}(\mathbf{\Lambda} + \kappa\mathbf{I})^{-1}\mathbf{Q}^\top\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top - \mathbf{I} \right) \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \cdot \right. \\ &\quad \left. \left(\mathbf{Q}(\mathbf{\Lambda} + \kappa\mathbf{I})^{-1}\mathbf{Q}^\top\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top - \mathbf{I} \right)^\top \right] \\ &= \text{tr} \left[\mathbf{Q}\mathbf{\Lambda} \left((\mathbf{\Lambda} + \kappa\mathbf{I})^{-1} - \mathbf{I} \right) \boldsymbol{\omega}^\circ (\boldsymbol{\omega}^\circ)^\top \left((\mathbf{\Lambda} + \kappa\mathbf{I})^{-1} - \mathbf{I} \right) \mathbf{Q}^\top \right] \\ &= \sum_{i=1}^m \lambda_i \left[\frac{\lambda_i}{\lambda_i + \kappa} - 1 \right]^2 (\omega_i^\circ)^2 \\ &= \sum_{i=1}^m \frac{(\omega_i^\circ)^2 \kappa^2 \lambda_i}{(\lambda_i + \kappa)^2}.\end{aligned}\tag{G.16}$$

Note that $MSME = 0$ when $\kappa = 0$ which stems from the fact that the employed model is complete.

G.3 Weight Fluctuation Penalty

Next the WFP is considered. Since the model cf. re:wmod is an LL-model the second term of the WFP sa:moderr vanishes according to Sec. 6.3.4 when $E_{\mathcal{T}}\{\Delta\mathbf{w}\} = E_{\mathcal{T}}\{\hat{\mathbf{w}} - \mathbf{w}^*\} = 0$. Using re:wdwhat2 and (G.8)

$$\Delta\mathbf{w} = \left(\mathbf{J}_N^{-1}\mathbf{H}_N - \mathbf{J}^{-1}\mathbf{H} \right) \mathbf{w}^\circ + \mathbf{J}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)\varepsilon(k).\tag{G.17}$$

Using the results of App. B – in particular Th. B.4 – then⁵

$$\mathbf{J}_N^{-1} \approx \mathbf{J}^{-1}, \quad N \rightarrow \infty.\tag{G.18}$$

³Since $E\{\mathbf{z}\mathbf{z}^\top\} = \mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$ then the variable $\boldsymbol{\zeta} = \mathbf{Q}^\top \mathbf{z}$ has a second order moment matrix:

$$E\{\boldsymbol{\zeta}\boldsymbol{\zeta}^\top\} = \mathbf{Q}^\top \mathbf{z}\mathbf{z}^\top \mathbf{Q} = \mathbf{\Lambda}$$

⁴In these calculations the commutative law regarding the product of diagonal matrices is used.

⁵In this case the dependence lag $M = 0$ since the input vector signal, $\mathbf{z}(k)$, is assumed to be an i.i.d. sequence.

Using this approximation (i.e., the remainder is $o(1/N)$) in re:wddelw we get:

$$E_{\mathcal{T}}\{\Delta\mathbf{w}\} = \mathbf{J}^{-1}E_{\mathcal{T}}\{\mathbf{H}_N - \mathbf{H}\} + \mathbf{J}^{-1}\frac{1}{N}\sum_{k=1}^N E_{\mathcal{T}}\{\mathbf{z}(k)\varepsilon(k)\} = \mathbf{0} \quad (\text{G.19})$$

as $E_{\mathcal{T}}\{\mathbf{H}_N\} = \mathbf{H}$ and $E\{\varepsilon(k)\} = 0^6$.

Hence, sa:moderr gives:

$$\begin{aligned} WFP &= E_{\mathcal{T}}\left\{E_{\mathbf{x}_t}\left\{[f(\mathbf{x}_t; \mathbf{w}^*) - f(\mathbf{x}_t; \hat{\mathbf{w}})]^2\right\}\right\} \\ &= E_{\mathcal{T}}\left\{E_{\mathbf{z}_t}\left\{[\mathbf{z}_t^\top \hat{\mathbf{w}} - \mathbf{z}_t^\top \mathbf{w}^*]^2\right\}\right\} \\ &= E_{\mathcal{T}}\left\{E_{\mathbf{z}_t}\left\{\text{tr}\left[\mathbf{z}_t \mathbf{z}_t^\top (\hat{\mathbf{w}} - \mathbf{w}^*) (\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right]\right\}\right\} \\ &= \text{tr}\left[\mathbf{H} \cdot E_{\mathcal{T}}\left\{\Delta\mathbf{w}\Delta\mathbf{w}^\top\right\}\right]. \end{aligned} \quad (\text{G.20})$$

Now the matrix $\Delta\mathbf{w}\Delta\mathbf{w}^\top$ is evaluated according to the expression re:wddelw:

$$\begin{aligned} \Delta\mathbf{w}\Delta\mathbf{w}^\top &= \left(\mathbf{J}_N^{-1}\mathbf{H}_N - \mathbf{J}^{-1}\mathbf{H}\right) \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \left(\mathbf{J}_N^{-1}\mathbf{H}_N - \mathbf{J}^{-1}\mathbf{H}\right)^\top \Big\} L_1 \\ &+ \left(\mathbf{J}_N^{-1}\mathbf{H}_N - \mathbf{J}^{-1}\mathbf{H}\right) \mathbf{w}^\circ \cdot \frac{1}{N} \sum_{k=1}^N \mathbf{z}^\top(k)\varepsilon(k) \cdot \mathbf{J}_N^{-1} \Big\} L_2 \\ &+ \mathbf{J}_N^{-1} \frac{1}{N} \sum_{k=1}^N \mathbf{z}(k)\varepsilon(k) \cdot \left(\mathbf{J}_N^{-1}\mathbf{H}_N - \mathbf{J}^{-1}\mathbf{H}\right)^\top \Big\} L_3 \\ &+ \mathbf{J}_N^{-1} \left(\frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N \mathbf{z}(k_1)\mathbf{z}^\top(k_2)\varepsilon(k_1)\varepsilon(k_2) \right) \mathbf{J}_N^{-1} \Big\} L_4. \end{aligned} \quad (\text{G.21})$$

The expectation w.r.t. the training set, \mathcal{T} , is now performed on the individual components in re:wddelw2:

1. If $\kappa = 0$ then the component L_1 equals zero since in that case $\mathbf{J}_N^{-1} = \mathbf{H}_N^{-1}$ and $\mathbf{J}^{-1} = \mathbf{H}^{-1}$. When $\kappa > 0$ we start by rewriting the first factor of L_1 by using the inverse Hessian approximation given in App. B by Co. B.1, i.e.,

$$\mathbf{J}_N^{-1} = \mathbf{J}^{-1} - \mathbf{J}^{-1}(\mathbf{J}_N - \mathbf{J})\mathbf{J}^{-1} + \dots \quad (\text{G.22})$$

Using re:jndef, (G.9) then a first order approximation⁷ of the inverse Hessian becomes

$$\mathbf{J}_N^{-1} \approx \mathbf{J}^{-1} - \mathbf{J}^{-1}(\mathbf{H}_N - \mathbf{H})\mathbf{J}^{-1} \quad (\text{G.23})$$

⁶The expectation w.r.t. the training set is performed in two steps:

$$E_{\mathcal{T}}\{\cdot\} = E_{\{\mathbf{z}(k), \varepsilon(k)\}}\{\cdot\} = E_{\{\mathbf{z}(k)\}}\left\{E_{\{\varepsilon(k)\}}\{\cdot|\{\mathbf{z}(k)\}\}\right\}$$

⁷A first order approximation turns out to be sufficient when the remainder of $E\{L_1\}$ is required to be $o(1/N)$.

and the first factor of L_1 equals:

$$\begin{aligned}
\mathbf{J}_N^{-1} \mathbf{H}_N - \mathbf{J}^{-1} \mathbf{H} &= \left(\mathbf{J}^{-1} - \mathbf{J}^{-1} (\mathbf{H}_N - \mathbf{H}) \mathbf{J}^{-1} \right) \mathbf{H}_N - \mathbf{J}^{-1} \mathbf{H} \\
&= \mathbf{J}^{-1} \left(\mathbf{H}_N - (\mathbf{H}_N - \mathbf{H}) \mathbf{J}^{-1} \mathbf{H}_N - \mathbf{H} \right) \\
&= \mathbf{J}^{-1} (\mathbf{H}_N - \mathbf{H}) \left(\mathbf{I} - \mathbf{J}^{-1} \mathbf{H}_N \right) \\
&= \mathbf{J}^{-1} (\mathbf{H}_N - \mathbf{H}) \mathbf{J}^{-1} (\mathbf{J} - \mathbf{H}_N) \\
&= \mathbf{J}^{-1} (\mathbf{H}_N - \mathbf{H}) \mathbf{J}^{-1} (\kappa \mathbf{I} + (\mathbf{H} - \mathbf{H}_N)). \quad (\text{G.24})
\end{aligned}$$

Neglecting all terms which decrease faster than $1/N$ is equivalent to considering expectations which contain at most two stochastic matrices⁸ according to App. C. Then substituting re:ffl1 into L_1 and keeping the essential terms yield

$$E_{\mathcal{T}} \{L_1\} = \kappa^2 \mathbf{J}^{-1} E_{\mathcal{T}} \left\{ ((\mathbf{H}_N - \mathbf{H})) \mathbf{J}^{-1} \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \mathbf{J}^{-1} ((\mathbf{H}_N - \mathbf{H})) \right\} \mathbf{J}^{-1}. \quad (\text{G.25})$$

The first addend of the WFP cf. re:wdwfp1 is by using the rule $\text{tr}[\mathbf{A}\mathbf{B}] = \text{tr}[\mathbf{B}\mathbf{A}]$ then given by:

$$\begin{aligned}
WFP_{L_1} &= \text{tr} [\mathbf{H} E_{\mathcal{T}} \{L_1\}] \\
&= \text{tr} \left[\kappa^2 \mathbf{J}^{-1} E_{\mathcal{T}} \left\{ ((\mathbf{H}_N - \mathbf{H})) \mathbf{J}^{-1} \mathbf{H} \mathbf{J}^{-1} ((\mathbf{H}_N - \mathbf{H})) \right\} \mathbf{J}^{-1} \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \right] \\
&= \text{tr} \left[\kappa^2 \mathbf{J}^{-1} \mathbf{A} \mathbf{J}^{-1} \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \right] \quad (\text{G.26})
\end{aligned}$$

where

$$\mathbf{A} = E_{\mathcal{T}} \left\{ ((\mathbf{H}_N - \mathbf{H})) \mathbf{J}^{-1} \mathbf{H} \mathbf{J}^{-1} ((\mathbf{H}_N - \mathbf{H})) \right\}. \quad (\text{G.27})$$

Writing out \mathbf{H}_N according to re:wdhndef:

$$\mathbf{A} = \frac{1}{N^2} \sum_{k_1=1}^N \sum_{k_2=1}^N E_{\mathcal{T}} \left\{ \left(\mathbf{z}(k_1) \mathbf{z}^\top(k_1) - \mathbf{H} \right) \mathbf{J}^{-1} \mathbf{H} \mathbf{J}^{-1} \left(\mathbf{z}(k_2) \mathbf{z}^\top(k_2) - \mathbf{H} \right) \right\}. \quad (\text{G.28})$$

Since $\mathbf{z}(k)$ is an i.i.d. sequence then $\mathbf{A} = \mathbf{0}$ as $k_1 \neq k_2$. This is easily seen by noting that $E_{\mathcal{T}} \{ \mathbf{z}(k) \mathbf{z}^\top(k) \} = \mathbf{H}$. When $k_1 = k_2$ ⁹ we find a non-zero contribution; consequently,

$$\mathbf{A} = \frac{\mathbf{K}}{N} \quad (\text{G.29})$$

where

$$\mathbf{K} = E_{\mathcal{T}} \left\{ \left(\mathbf{z}(k) \mathbf{z}^\top(k) - \mathbf{H} \right) \mathbf{J}^{-1} \mathbf{H} \mathbf{J}^{-1} \left(\mathbf{z}(k) \mathbf{z}^\top(k) - \mathbf{H} \right) \right\}. \quad (\text{G.30})$$

In general the evaluation of \mathbf{K} is problematic; even when $\mathbf{z}(k)$ is Gaussian distributed. However, it is worth noting that \mathbf{A} is inversely porportional to N and positive semidefinite¹⁰. That means, $WFP_{L_1} \geq 0$. Furthermore, note that $WFP_{L_1} = 0$

⁸These matrices are assumed to have zero means which is ensured in the present case since $E\{\mathbf{H}_N - \mathbf{H}\} = \mathbf{0}$.

⁹Note there are N terms with $k_1 = k_2$ in the sum over k_1, k_2 .

¹⁰A matrix is positive semidefinite when: $\forall \mathbf{v} \neq \mathbf{0} : \mathbf{v}^\top \mathbf{A} \mathbf{v} \geq 0$. \mathbf{A} can be expressed as $\mathbf{A} = \mathbf{A}^{\top/2} \mathbf{A}^{1/2}$ due to the fact that \mathbf{H} obeys the Cholesky decomposition:

$$\mathbf{H} = \mathbf{Q} \mathbf{A} \mathbf{Q}^\top = (\mathbf{Q} \mathbf{A}^{1/2}) (\mathbf{Q} \mathbf{A}^{1/2})^\top$$

. Now,

$$\mathbf{v}^\top \mathbf{A}^{\top/2} \mathbf{A}^{1/2} \mathbf{v} = \mathbf{v}_1^\top \mathbf{v}_1 \geq 0$$

where $\mathbf{v}_1 = \mathbf{A}^{1/2} \mathbf{v}$.

as $\kappa = 0$ and $\kappa \rightarrow \infty$ ¹¹.

2. The expectation of the components L_2 and L_3 equals zero since first fixing $\mathbf{z}(k)$ and performing the expectation w.r.t. $\varepsilon(k)$ involves only the mean of ε which per definition is zero.
3. The expectation w.r.t. \mathcal{T} is performed in two steps: First w.r.t. $\varepsilon(k)$ and then w.r.t. $\mathbf{z}(k)$. Noting that $\varepsilon(k)$ is an i.i.d. sequence then the expectation of L_4 becomes:

$$\begin{aligned} E_{\mathcal{T}} \{L_4\} &= E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{J}_N^{-1} \left(\frac{1}{N^2} \sum_{k=1}^N \mathbf{z}(k) \mathbf{z}^\top(k) \cdot \sigma_\varepsilon^2 \right) \mathbf{J}_N^{-1} \right\} \\ &= \frac{\sigma_\varepsilon^2}{N} \cdot E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{J}_N^{-1} \mathbf{H}_N \mathbf{J}_N^{-1} \right\}. \end{aligned} \quad (\text{G.31})$$

If $\kappa = 0$ then $\mathbf{J}_N^{-1} = \mathbf{H}_N^{-1}$ which results in

$$E_{\mathcal{T}} \{L_4\} = \frac{\sigma_\varepsilon^2}{N} \cdot E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{H}_N^{-1} \right\} \approx \frac{\sigma_\varepsilon^2}{N} \cdot \mathbf{H}^{-1}, \quad N \rightarrow \infty. \quad (\text{G.32})$$

according to Th. B.4. When $\kappa > 0$ the result of re:invjapp is used, i.e.,

$$\begin{aligned} E_{\mathcal{T}} \{L_4\} &= \frac{\sigma_\varepsilon^2}{N} \cdot E_{\{\mathbf{z}(k)\}} \left\{ \mathbf{J}_N^{-1} \mathbf{H}_N \mathbf{J}_N^{-1} \right\} \\ &\approx \frac{\sigma_\varepsilon^2}{N} \cdot \mathbf{J}^{-1} E_{\mathbf{z}(k)} \left\{ \mathbf{H}_N \right\} \mathbf{J}^{-1} \\ &= \frac{\sigma_\varepsilon^2}{N} \cdot \mathbf{J}^{-1} \mathbf{H} \mathbf{J}^{-1}. \end{aligned} \quad (\text{G.33})$$

Using the eigenvalue decompositions of \mathbf{H} and \mathbf{J}^{-1} cf. re:hdecom and (G.14), respectively, we get:

$$\begin{aligned} E_{\mathcal{T}} \{L_4\} &= \frac{\sigma_\varepsilon^2}{N} \cdot \mathbf{Q} (\mathbf{\Lambda} + \kappa \mathbf{I})^{-1} \mathbf{Q}^\top \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^\top \mathbf{Q} (\mathbf{\Lambda} + \kappa \mathbf{I})^{-1} \mathbf{Q}^\top \\ &= \frac{\sigma_\varepsilon^2}{N} \cdot \mathbf{Q} (\mathbf{\Lambda} + \kappa \mathbf{I})^{-2} \mathbf{\Lambda} \mathbf{Q}^\top. \end{aligned} \quad (\text{G.34})$$

Substituting this expression into re:wdwfp1 we get:

$$\begin{aligned} WFP_{L_4} &= \text{tr} [\mathbf{H} E_{\mathcal{T}} \{L_4\}] \\ &= \frac{\sigma_\varepsilon^2}{N} \cdot \text{tr} \left[\mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^\top \mathbf{Q} (\mathbf{\Lambda} + \kappa \mathbf{I})^{-2} \mathbf{\Lambda} \mathbf{Q}^\top \right] \\ &= \frac{\sigma_\varepsilon^2}{N} \cdot \text{tr} \left[(\mathbf{\Lambda} + \kappa \mathbf{I})^{-2} \mathbf{\Lambda}^2 \right] \\ &= \frac{\sigma_\varepsilon^2}{N} \sum_{i=1}^m \frac{\lambda_i^2}{(\lambda_i + \kappa)^2}. \end{aligned} \quad (\text{G.35})$$

¹¹This is due to the fact that \mathbf{J}^{-1} decreases to zero like $1/\kappa$.

Finally,

$$\begin{aligned} WFP &= WFP_{L_1} + WFP_{L_4} \\ &= \frac{1}{N} \cdot \text{tr} \left[\kappa^2 \mathbf{J}^{-1} \mathbf{K} \mathbf{J}^{-1} \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \right] + \frac{\sigma_\varepsilon^2}{N} \sum_{i=1}^m \frac{\lambda_i^2}{(\lambda_i + \kappa)^2}. \end{aligned} \quad (\text{G.36})$$

Note for $\kappa = 0$:

$$WFP = \frac{m\sigma_\varepsilon^2}{N}. \quad (\text{G.37})$$

G.4 Optimizing the Regularization Parameter

Note that $WFP \rightarrow 0$ as $N \rightarrow \infty$ while the $MSME \neq 0$ unless $\kappa = 0$. That is, the optimal value of κ , $\kappa_{\text{opt}} = 0$ as $N \rightarrow \infty$. Since the results above are based on an $o(1/N)$ approximation it seems natural to consider κ_{opt} to $o(1/N)$, i.e., $\kappa_{\text{opt}} \propto 1/N$.

The optimal κ is found by solving:

$$\frac{\partial MSME}{\partial \kappa} + \frac{\partial WFP}{\partial \kappa} = 0. \quad (\text{G.38})$$

According to re:wmsmef

$$\frac{\partial MSME}{\partial \kappa} = \sum_{i=1}^m \frac{2|\omega_i^\circ|^2 \kappa \lambda_i^2}{(\lambda_i + \kappa)^3}. \quad (\text{G.39})$$

Since $\kappa_{\text{opt}} \propto 1/N$ and the fact that N does not appear explicitly in re:msmele we expand to first order κ and get

$$\frac{\partial MSME}{\partial \kappa} = 2\kappa \sum_{i=1}^m \frac{(\omega_i^\circ)^2}{\lambda_i} + o(1/N). \quad (\text{G.40})$$

Note that $\partial MSME/\partial \kappa > 0$; consequently, $MSME$ increases with κ .

According to re:wdwfpf

$$\frac{\partial WFP}{\partial \kappa} = \frac{1}{N} \left[\frac{\partial \left[\kappa^2 \mathbf{J}^{-1} \mathbf{K} \mathbf{J}^{-1} \mathbf{w}^\circ (\mathbf{w}^\circ)^\top \right]}{\partial \kappa} + \sum_{i=1}^m \frac{-2\sigma_\varepsilon^2 \lambda_i^2}{(\lambda_i + \kappa)^3} \right]. \quad (\text{G.41})$$

Since $\partial WFP/\partial \kappa$ is inversely proportional to N we expand to zero order in κ only. The κ dependence of first term of re:wfp1e is given by $\kappa^2/(\lambda_i + \kappa)^4$ which is seen by re:wfp1l as $\mathbf{J}^{-1} = \mathbf{Q}(\mathbf{\Lambda} + \kappa \mathbf{I})^{-1} \mathbf{Q}^\top$ is inversely proportional to κ . Consequently, this term does not contribute to zero order. In summary,

$$\frac{\partial WFP}{\partial \kappa} = \frac{-2\sigma_\varepsilon^2}{N} \sum_{i=1}^m \frac{1}{\lambda_i} + o(1/N). \quad (\text{G.42})$$

$\partial WFP/\partial \kappa < 0$ which implies that WFP decreases with κ . Consequently, cf. re:kapeqn, the average generalization error is minimized by setting:

$$\kappa_{\text{opt}} = \frac{\sigma_\varepsilon^2 \sum_{i=1}^m \frac{1}{\lambda_i}}{N \sum_{i=1}^m \frac{(\omega_i^\circ)^2}{\lambda_i}} + o(1/N)$$

$$\begin{aligned}
&= \frac{\sigma_\varepsilon^2 \text{tr} \mathbf{\Lambda}^{-1}}{N (\mathbf{w}^\circ)^\top \mathbf{\Lambda}^{-1} \mathbf{w}^\circ} + o(1/N) \\
&= \frac{\sigma_\varepsilon^2 \text{tr} \mathbf{H}^{-1}}{N (\mathbf{w}^\circ)^\top \mathbf{H}^{-1} \mathbf{w}^\circ} + o(1/N). \tag{G.43}
\end{aligned}$$

Notice certain facts concerning this optimal value:

- It depends on the noise level, σ_ε . Clearly if no noise is present then $\kappa_{\text{opt}} = 0$ since the data in the training set perfectly describe the system as we note the model is complete. On the other hand, if noise is present¹² there will be a significant *WFP* which can be lowered by introducing a non-zero κ and thus introducing a *MSME*.
- To order $1/N$, κ_{opt} is independent of the fourth order statistic of the input vector signal, i.e., independent of \mathbf{K} .
- It is – loosely speaking – inversely proportional to the square of the optimal weight values. This is due to the fact that we regularize against the zero weight vector¹³.

It is important to point out that κ_{opt} can not be calculated in practical applications since it depends on the optimal weights which of course are unknown.

In this context we will show below that even when the optimal value, κ_{opt} , is not reached there may still be a reduction in the average generalization error. In fact, the average generalization error is reduced within an interval of κ values. To see this the change in the average generalization error, $\Delta\Gamma$, when employing a non-zero κ is calculated. Formally, $\Delta\Gamma = \Gamma(0) - \Gamma(\kappa)$ where $\Gamma(\kappa)$ is the average generalization error using the regularization parameter κ . According to sa:moderr, (G.37)

$$\Gamma(0) = \sigma_\varepsilon^2 + \frac{m\sigma_\varepsilon^2}{N}. \tag{G.44}$$

In order to evaluate $\Gamma(\kappa)$ we integrate the approximations re:msmefn, (G.42) w.r.t. κ and add suitable integration constants. This yields:

$$MSME \approx \kappa^2 \sum_{i=1}^m \frac{(\omega_i^\circ)^2}{\lambda_i} = \kappa^2 (\mathbf{w}^\circ)^\top \mathbf{H}^{-1} \mathbf{w}^\circ, \tag{G.45}$$

$$WFP \approx \frac{\sigma_\varepsilon^2}{N} \sum_{i=1}^m 1 - \frac{2\kappa}{\lambda_i} = \frac{m\sigma_\varepsilon^2}{N} - \frac{2\kappa\sigma_\varepsilon^2}{N} \text{tr} \mathbf{H}^{-1}. \tag{G.46}$$

Accordingly,

$$\Delta\Gamma = \kappa \left[-\kappa (\mathbf{w}^\circ)^\top \mathbf{H}^{-1} \mathbf{w}^\circ + \frac{2\kappa\sigma_\varepsilon^2}{N} \text{tr} \mathbf{H}^{-1} \right]. \tag{G.47}$$

Consequently, $\Delta\Gamma > 0$ for all $\kappa \in]0; 2\kappa_{\text{opt}}]$.

¹²Notice that the noise variance enters the *WFP* only.

¹³The regularizing term is zero when employing $\mathbf{w} = \mathbf{0}$.

APPENDIX H

PAPER 1: A NEURAL ARCHITECTURE FOR ADAPTIVE FILTERING

This appendix contains a reprint¹ of the paper: “A Neural Architecture for Nonlinear Adaptive Filtering of Time Series,” in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.), *Proceedings of the 1991 IEEE Workshop: Neural Networks for Signal Processing*, Piscataway, New Jersey: IEEE Service Center, 1991, pp. 533–542. This paper is written in cooperation with Ph.D. Nils Hoffmann.

¹The typesetting is slightly changed and a few misprints is rectified.

A Neural Architecture for Nonlinear Adaptive Filtering of Time Series

Nils Hoffmann and Jan Larsen
The Computational Neural Network Center
Electronics Institute, Building 349
Technical University of Denmark
DK-2800 Lyngby, Denmark

H.1 Introduction

The need for nonlinear adaptive filtering may arise in different types of filtering tasks such as prediction, system identification and inverse modeling [16]. The problem of predicting chaotic time series has been addressed by several authors [6], [11]. In the latter case a feed-forward neural network was used both for prediction and for identification of a simple, nonlinear transfer function (system identification). These filtering tasks can be solved using a filtering configuration shown in Fig. H.1 [16].

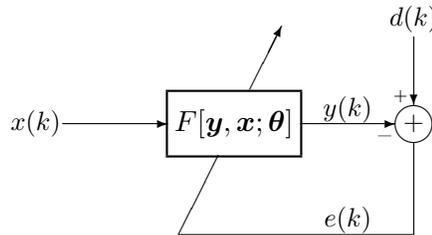


Figure H.1: Nonlinear adaptive filtering configuration. $x(k)$ is the input, $y(k)$ the output and $d(k)$ the desired signal. The filter is adapted in order to minimize the cost function: $\sum_{i=0}^k \lambda^{k-i} e^2(i)$, where $k = 1, 2, \dots, N$ and $0 < \lambda \leq 1$ is the forgetting factor [7].

The nonlinear filter may be designed to realize

$$y(k) = F[y(k-1), \dots, y(k-M), x(k), \dots, x(k-L+1); \theta] \quad (\text{H.1})$$

where $F[\cdot]$ is an unknown nonlinear function parameterized by θ , k is the discrete time index and L, M are filter orders.

The general structure of equation (H.1) enables one to model any nonlinear, discrete system. θ is assumed to be slowly time varying and consequently $y(k)$ is quasi stationary. The use of a recursive, nonlinear filter does, however, pose serious difficulties regarding stability. The filter may display limit cycles, chaotic behavior and unboundedness. The scope of this paper is to implement only the nonrecursive part of (H.1).

We propose a modularized architecture for the nonlinear filter in Fig. H.1 including algorithms for adapting the filter. Further we develop simple guidelines for selecting a specific filter design within the proposed architecture given a priori knowledge of the distribution and origin (type of modeling problem) of the input signal $x(k)$ and the desired

response $d(k)$. Finally we present simulations in order to further investigate the nature of the relations between filter design and the statistics of x and d .

H.2 Nonlinear Filter Architecture

The proposed filter architecture is shown in Fig. H.2. The filter, which may be viewed as a generalization of the Wiener Model [14], is divided into three partially independent (depending on specific design) sections: A preprocessing unit containing the filter memory, a **memoryless, multidimensional nonlinearity** (MMNL) and a linear combiner. The structure is selected in order to modularize the modeling problem which ensures a proper and sparse parameterization, and facilitates incorporation of a priori knowledge in contrast to the limited possibilities when using an ordinary feed-forward neural network. The

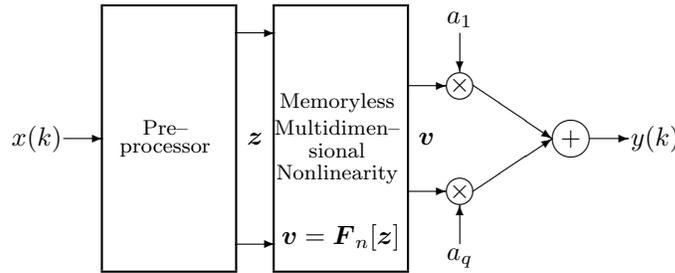


Figure H.2: Nonlinear filter architecture.

main objective of the preprocessor is to extract the essential information contained in $\mathbf{x}_k = [x(k), x(k-1), \dots, x(k-L+1)]'$ ensuring that \mathbf{z} has a dimension, $p \leq L$. The nonlinearity is memoryless and transforms the vector \mathbf{z} into the vector \mathbf{v} , and finally the linear combiner forms a weighted sum $y(k)$ of the terms in \mathbf{v} . This corresponds to rewriting (H.1) as:

$$y(k) = \mathbf{a}' \mathbf{F}_n[\mathbf{F}_p(x(k))] \quad (\text{H.2})$$

where $\mathbf{F}_p(\cdot)$ is the preprocessor, $\mathbf{F}_n[\cdot]$ is the MMNL and \mathbf{a} the weights of the linear combiner. The filter could be viewed as a heterogeneous multilayer neural network. All sections could be adapted, but this is not always necessary, as we shall see in the next section.

H.3 Filter Design

H.3.1 Signal Dependence

It seems reasonable that the specific design of the filter depends on the origin and distribution of the signals $x(k)$ and $d(k)$, and we will summarize some guidelines for choosing an appropriate design as follows:

Case	\mathbf{x}	\mathbf{d}	Model
1	Gaussian	Gaussian	$d(k) = \mathbf{a}' \mathbf{x}_k + \epsilon(k)$
2	Gaussian	Non-Gaussian	$d(k) = F[\mathbf{x}_k] + \epsilon(k)$
3	Non-Gaussian	Gaussian	$d(k) = F[\mathbf{x}_k] + \epsilon(k)$
4	Non-Gaussian	Non-Gaussian	$d(k) = F[\mathbf{x}_k] + \epsilon(k)$

1. The linear filter is optimal in this case, so $\mathbf{v} = [1 \ \mathbf{z}'']'$, see e.g., [12, Theorem 14.3].
2. The Wiener model, i.e., a bank of orthogonal linear filters in the preprocessing unit followed by a fixed (non-adaptive) polynomial nonlinearity, provides a filter which can be adapted in a very simple and fast way [14]. The linear filters may be determined using principal component analysis (PCA) on \mathbf{x} . This case is associated with the problem of nonlinear system identification where $d(k)$ is the output of an unknown system and $x(k)$ the input.
3. In this case there is no obvious choice of filter design. The case arises e.g., in inverse modeling where $d(k)$ is the driving signal of the unknown system and $x(k)$ the resulting output.
4. This case relates to prediction of nonlinear time series where $x(k) = d(k - \tau)$ is, in fact, a delayed version of $d(k)$. Previous simulation studies [11] indicate that the nonlinearity should be constructed from bounded functions (e.g., the commonly used $\tanh(\cdot)$) rather than polynomials, which have the inconvenient property of growing fast towards infinity.

H.3.2 Preprocessing Methods

We present two possible methods for dimensionality determination. If the unknown system being modeled can be described in the form of a nonlinear differential equation it is possible, in some cases, to determine dimensionality by letting $\mathbf{z} = [x(k), Dx(k), \dots, D^{p-1}x(k)]'$ where D is a discrete derivative operator. We denote this preprocessor: The **derivative preprocessor** (DPP). The following example (the pendulum) illustrates that we often have $p < L$ without losing information:

$$\frac{d^2x(t)}{dt^2} + \beta \frac{dx(t)}{dt} + \alpha \sin[x(t)] = d(t) \quad (\text{H.3})$$

$$D^2x(k) + \beta Dx(k) + \alpha \sin[x(k)] = F[D^2x(k), Dx(k), x(k)] = d(k) \quad (\text{H.4})$$

Equation (H.4) which is a discrete approximation of (H.3) clearly shows, that $d(k)$ may be expressed as a function of only $p = 3$ variables i.e., the derivatives of $x(k)$. To ensure that the approximation of the derivative operator D is accurate (i.e., approximates the continuous time derivative operator) over a broad range of frequencies it must be implemented using a linear filter with high order. A tapped delay line used without a preprocessing element would necessarily need the same length L to hold the same information so L is obviously greater than p in this example. In practice there exist two major problems with this kind of preprocessing: 1. Derivatives amplify noise (SNR is often low at high frequencies which are amplified the most) and 2. The optimal differentiating filter is non-causal. The first problem may be dealt with by noise reducing lowpass-filtering (see [3] for an optimal approach). The second obstacle may be circumvented by delaying $d(k)$ thus allowing for non-causal filtering i.e., estimating $d(k - r)$ ($r \geq 0$) using $x(k), x(k - 1), \dots$.

Another method is **principal component analysis** (PCA) which serves two purposes: 1. PCA makes the components in \mathbf{z} mutually uncorrelated (convergence speed-up for certain weight estimation algorithms, e.g., Backpropagation [8]) and 2. It determines the dimensionality which is done by removing the last $L - p$ **principal components** (PC's) with eigenvalues close to zero. The amount of lost information can be estimated as the total variance of the removed PC's divided by the total variance of \mathbf{x} , i.e., $L \cdot V\{x(k)\}$. The remaining PC's constitute the optimal *linear* projection (in the mean square sense) of \mathbf{x} on the space spanned by the first p eigenvectors of the covariance matrix of \mathbf{x} . A

theoretical well-founded procedure for on-line estimation of the PC's has recently been described [17]. Other schemes are given in [8, Chap. 8.3].

Further support for the use of PCA can be found in an information theoretical interpretation: Maximize the mutual information $I(\mathbf{x}; \mathbf{z})$ between \mathbf{x} and \mathbf{z} . It can be shown that if \mathbf{z} is Gaussian (also valid for certain similar probability density functions): $\max I(\mathbf{x}; \mathbf{z}) = \max H(\mathbf{z}) \Leftrightarrow \max \det V\{\mathbf{z}\}$ where $V\{\cdot\}$ is the variance and $H(\cdot)$ is the entropy. PCA is in fact done by maximizing $\det V\{\mathbf{z}\}$ [10, p. 682] which implicates that dimensionality determination by means of PCA is equivalent to maximizing $I(\mathbf{x}; \mathbf{z})$. When dealing with signals corrupted by noise PCA is not always preferable (especially if the signal to noise ratio is low) because the PC's then will reflect the noise. Furthermore using PCA when the spectral overlap of the signals $x(k)$ and $d(k)$ is small is not reasonable. This is due to the fact that the spectrum of the PC's corresponding to large eigenvalues mainly contains the dominating frequencies in $x(k)$ thus neglecting the frequencies that dominate the spectrum of $d(k)$.

H.3.3 Memoryless Multidimensional Nonlinearities

When approximating the MMNL, $F_n[\mathbf{z}]$, $\mathbf{z} \in \mathcal{I}$ where \mathcal{I} is the input space, we distinguish between *local* and *global* approximation methods [6]. In a local approximation context \mathcal{I} is divided into smaller domains. F is now approximated in each domain by separate nonlinearities. This results in a modularization of the MMNL which ensures a sparse parameterization. By global approximation is meant, that no dividing of \mathcal{I} is done at all. In general there is a trade off between the number of domains and the complexity of the subsequent nonlinearities.

H.3.3.1 Global Approximation Methods

In this case we deal with only one nonlinearity which must have the ability of approximating F arbitrarily accurate. We will discriminate between *fixed* and *adaptive* nonlinearities.

A natural choice of a fixed nonlinearity (FNL) is to let \mathbf{v} contain all possible products of z_i , e.g., terms of the form $\prod_{i=1}^p z_i^{s_i}$ up to some order $s = \sum_{i=1}^p s_i$. When these terms are added by the linear combiner it all amounts to a multidimensional Taylor expansion which combined with the linear filters in the preprocessor defines a discrete Volterra filter. Fréchet showed [14] that any continuous $F(x(t))$ can be represented by a Volterra filter with uniform convergence when $s \rightarrow \infty$ for $x(t) \in \mathcal{J}$, $\mathcal{J} \subseteq \mathcal{I}$. A convenient representation can be obtained by using a complete set of orthogonal polynomials P_i , where i is the order of the polynomial. If $z_i \in N(0, 1)$, P_i are identical to the Hermite polynomials. With these polynomials convergence in mean is assured over a suitable interval $[a; b]$. The generalization to the multidimensional case is done by forming all products of polynomials in different variables e.g., $\prod_{i=1}^p P_{s_i}(z_i)$ (see [14] for details). In general the probability density $f_{\mathbf{z}}(\mathbf{z})$ is unknown which makes it impossible to find the orthogonal polynomials. Instead we propose the use of Chebychev polynomials preceded by squashing functions that limits the z_i to the interval $] - 1; 1[$ thereby limiting the v_i to $] - 1; 1[$.

An obvious choice for an adaptive nonlinearity (ANL) is a layered feed-forward neural network composed of sigmoidal neurons. It is well-known (see e.g., [5]) that a two layer feed-forward network with a linear output neuron (under rather mild conditions on the activation function, g) can uniformly approximate any function as the number of neurons in the hidden layer reaches infinity. We suggest that the nonlinearity is composed of p layers. The first layer consists of p neurons which maps z_i , $1 \leq i \leq p$ into $g(z_i w_i^1 + w_i^0)$

(w_i^1 ensures a proper scaling, see below). The second layer consists of q neurons ($q \rightarrow \infty$ for an arbitrarily accurate approximation) and the outputs then form the nonlinear terms $[v_1, \dots, v_{q-p}]'$. It is further suggested to explicitly model the linear part by letting $[v_{q-p+1}, \dots, v_q]' = [z_1, \dots, z_p]'$.

H.3.3.2 Local Approximation Methods

A possible way to divide the input space is to use Localized Receptive Fields [13]. The output from each receptive field is then fed into separate nonlinearities. As above they could be either fixed or adaptive. Note that there is a trade-off between the number of domains in the input space and the complexity of the succeeding nonlinearities. Other local approximation schemes can be found in [6], [15], [14, Chap. 21].

H.3.3.3 Scaling of \mathbf{z}

Scaling of \mathbf{z} serves two purposes. First, we have to restrict z_i to an interval where the nonlinearity is slowly varying; i.e., neither growing towards infinity (as polynomials for large arguments) nor being constant (like $\tanh(\cdot)$ for large arguments). Secondly, we have to ensure that only the significant amplitude range of z_i (i.e., the interval where $f_{z_i}(z_i) > \varepsilon$, $0 < \varepsilon \ll 1$) is fed into the filter. Otherwise very unlikely values of z_i will be weighted too much in the cost function thus resulting in a poor performance. Scaling with a suitable measure of z_i , e.g., 2–3 standard deviations, serves this purpose.

H.3.4 Weight Estimation Algorithms

The task is to estimate the weights $\boldsymbol{\theta}$ so that the cost function $\sum_{i=0}^k \lambda^{k-i} e^2(i)$, $k = 1, 2, \dots, N$ is minimized [7] where e is the difference between the desired and the actual response and $0 < \lambda \leq 1$ is the forgetting factor.

H.3.4.1 Fixed Nonlinearity

In designs with a FNL it is only necessary to adapt the linear combiner. This is especially simple if \mathbf{x} is Gaussian and PCA is used as preprocessing making \mathbf{z} white (independent) and Gaussian. Now if z_i is scaled to unity variance and Hermite polynomials are used in the nonlinearity then the v_j will be uncorrelated and the weights may thus be updated using the crosscorrelation method proposed in [14]: $a_j = C\{v_j d\}/V\{v_j\}$, $1 \leq j \leq q$ where $C\{v_j d\}$ is the covariance between v_j and d and $V\{v_j\}$ is the variance of v_j . In most cases, however, \mathbf{v} is non-white but owing to the fact that $y(k)$ is linear in the weights, adaptive algorithms known from linear adaptive filtering such as the recursive least squares (RLS) [7, p. 385] or the least mean squares (LMS) [16, p. 99] are usable. The latter is perhaps the best choice for large values of q because it needs less computations and memory capacity while the major advantage of the RLS is the much faster convergence for highly correlated inputs (v_j).

H.3.4.2 Adaptive Nonlinearity

Designs with ANL implicate that estimation of the weights is a nonlinear optimization task which in general is hard to solve (global optimization) but local optimization schemes have been given, e.g., **Backpropagation** (BP). BP is known to have very slow convergence [4]. There is therefore a need for development of algorithms with faster convergence. Several

second-order algorithms (SOA) have been proposed, see e.g., [4]. A SOA incorporates the information contained in the Hessian (\mathbf{H}) of the cost function and the weights are updated according to the Newton-Raphson algorithm. In contrast to BP the SOA parameters are given a natural interpretation. $0 < \lambda \leq 1$ is the exponential forgetting factor, $0 < \mu \leq 1$ is the stepsize which normally is non-critical, and δ ($\mathbf{H}^{-1} = \delta \mathbf{I}$) is initially chosen large. We suggest a further development that takes the problems of nearly singular Hessian matrices into account. This problem arises in "flat" parts of the cost function. It is proposed to use the U-D factorization of \mathbf{H}^{-1} due to Bierman [2].

H.4 Simulations

H.4.1 Simulated Systems

In order to compare filter designs when filtering signals with different origin and distribution we study three systems covering the cases 2–4 on p. 401.

Example	x	d	Model
System Identification (SI)	Band-limited Gaussian noise	Non-Gaussian	Equation (H.5)
Inverse Modeling (IM)	Non-Gaussian	Gaussian lowpass filtered noise	Equation (H.6)
Prediction (P)	Non-gaussian	Non-gaussian	Equation (H.7)

H.4.1.1 System Identification

A simple system describing a wave force problem is given by Morison's equation [1, p. 234]. $x(t)$ is the wave velocity and $d(t)$ the wave force. The desired signal $d(k)$ is a discrete version of $d(t)$ sampled with the sampling period $\Delta T = 1$ and the same applies to $x(k)$.

$$d(t) = 0.2 \frac{dx(t)}{dt} + 0.8x(t)|x(t)| \quad (\text{H.5})$$

H.4.1.2 Inverse Modeling

We consider the pendulum where $x(t)$ is the angle deflection and $d(t)$ the force. The desired signal $d(k)$ is a discrete version of $d(t - \tau)$ (sampled with $\Delta T = 0.05$) where τ is a delay aiming to cancel both the delay between $d(t)$ and $x(t)$ and the delay in the preprocessing unit of the nonlinear filter. $x(k)$ corresponds to the angle $x(t)$.

$$\frac{d^2x(t)}{dt^2} + 0.2 \frac{dx(t)}{dt} + 4\pi^2 \sin[x(t)] = d(t) \quad (\text{H.6})$$

H.4.1.3 Prediction

The signal $x(t)$ is generated by the chaotic Mackey-Glass equation which often is used in a benchmark test for nonlinear predictors [6]. $d(k)$ is a discrete version of $x(t)$ and $x(k)$ a delayed, discrete version of $x(t - \tau)$ where τ signifies how far ahead we predict. Sampling the signal with $\Delta T = 1$ τ equals 100 time steps like in [6], [11].

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t - 17)}{1 + x(t - 17)^{10}} \quad (\text{H.7})$$

The systems mentioned above have all been simulated using discrete approximations of the derivatives. These discrete filters are all non-recursive which means that a non-recursive nonlinear adaptive filter is adequate. The actual training and cross validation signals have been obtained by decimating the input and output signals in order to avoid oversampling.

H.4.2 Numerical Results

In the table below are listed the main results. A measure of the filter performance is given by the error index: $E = \sigma_e/\sigma_d$, where σ_e , σ_d denote the standard deviation of the error and the desired signals respectively (cross validation). The number of parameters W gives an indication of the complexity.

Ex.	Prep.	L	p	Nonlinearity					
				Fixed		Adaptive		None	
				E	W	E	W	E	W
SI	PCA	14	4	0.263	94	0.215	158	0.417	5
SI	DPP	19	2	0.200	46	0.107	128	0.389	3
SI	None	14	-	-	-	-	-	0.382	15
IM	DPP	19	3	0.075	152	0.116	131	0.448	4
IM	None	19	-	-	-	-	-	0.402	20
P	PCA	20	4	0.152	170	-	-	0.539	5
P	None	20	-	-	-	-	-	0.539	21

In all simulations we have used 9000 samples for training and 8000 for cross validation. During training an algorithm based on a statistical test [9] was used to eliminate non-significant weights which accounts for the variations in W . The FNL consisted of bounded Chebychev polynomials and the ANL was implemented using multilayer neural net. In both cases we used 2. order algorithms for adapting the weights. In general the simulations indicate that the nonlinear filters are clearly superior to the linear with respect to E . This improvement is, however, gained at the expense of an increased complexity. The FNL and ANL's seems to show roughly equal performance on the selected examples, with the ANL having a better parameterization in the SI example (more significant weights and lower E) and vice versa in the IM example. The two preprocessing methods seem to complement each other. In the examples SI,IM, where the equations can be closely approximated with discrete derivative operators of low order, the use of discrete differentiating filters in the preprocessing unit yields a better performance with lower complexity than the use of PCA. This shows that the discrete derivatives are more informative than the PC's in the chosen examples. Using PCA in the example with the pendulum is in fact extremely bad because the PC's mainly reflect the low frequency components in $x(k)$ while the high frequencies carry most of the information about $d(k)$. In contrast, PCA works very well for the example of prediction whereas it is not easy to approximate the Mackey-Glass equation using only low-order derivatives (i.e., the use of a DPP is a bad choice). Finding an appropriate preprocessing method seems thus to require knowledge of an approximate mathematical model for the unknown system. Alternatively a rule of thumb saying that the preprocessor should make the spectrums of the $z_i(k)$ "close" to the spectrum of $d(k)$ could be used. In the prediction example we have used a PCA with $L = 20$ and $p = 4$ which allows us to compare our results with the ones obtained in [11] where a performance of $E = 0.054$ was found using an ANL and a total of 171 weights and $E = 0.28$ using a

6th order fixed polynomial nonlinearity and 210 weights. This indicates, that although the ANL still performs better on this example an increase in performance of the FNL has been gained by using bounded polynomials.

H.5 Conclusion

In this paper a neural architecture for adaptive filtering which incorporates a modularization principle is proposed. It facilitates a sparse parameterization, i.e., fewer parameters have to be estimated in a supervised training procedure. The main idea is to use a preprocessor which determine the dimension of the input space and further can be designed independent of the subsequent nonlinearity. Two suggestions for the preprocessor are presented: The derivative preprocessor and the principal component analysis. A novel implementation of fixed Volterra nonlinearities is given. It forces the boundedness of the polynomials by scaling and limiting the inputs signals. The nonlinearity is constructed from Chebychev polynomials. We apply a second-order algorithm for updating the weights for adaptive nonlinearities based on previous work of Chen *et al.* [4] and the U-D factorization of Bierman [2]. Finally the simulations indicate that the two kinds of preprocessing tend to complement each other while there is no obvious difference between the performance of the ANL and FNL.

H.6 Acknowledgments

We would like to thank Lars Kai Hansen, Peter Koefoed Møller, Klaus Bolding Rasmussen and John E. Aasted Sørensen for helpful comments on this paper.

References

- [1] J.S. Bendat: *Nonlinear Systems Analysis and Identification*, New York: John Wiley & Sons, 1990.
- [2] G.J. Bierman: "Measurement Updating using the U-D Factorization," in *Proceedings of the IEEE Int. Conf. on Decision and Control*, 1975, pp. 337–346.
- [3] B. Carlsson, A. Ahlén & M. Sternad: "Optimal Differentiation Based on Stochastic Signal Models," *IEEE Transactions on Signal Processing*, vol. 39, no. 2, 341–353, 1991.
- [4] S. Chen, C.F.N. Cowan, S.A. Billings & P.M. Grant: "Parallel Recursive Prediction Error Algorithm for Training Layered Neural Networks," *Int. Journal of Control*, vol. 51, no. 6, pp. 1215–1228, 1990.
- [5] G. Cybenko: "Approximation by Superposition of a Sigmoidal Function," *Math. Control Signal Systems*, no. 2, pp. 303–314, 1989.
- [6] J.D. Farmer & J.J. Sidorowich: "Exploiting Chaos to Predict the Future and Reduce Noise," *Technical Report LA-UR-88*, Los Alamos National Laboratory, 1988.
- [7] S. Haykin: *Adaptive filter theory*, Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

- [8] J. Hertz, A. Krogh & R.G. Palmer: *Introduction to the Theory of Neural Computation*, Redwood City, California: Addison-Wesley Publishing Company, 1991.
- [9] N. Hoffmann & J. Larsen: “An Algorithm for Parameter Reduction in Nonlinear Adaptive Filters,” *Technical Note, Electronics Institute, Technical University of Denmark*, February 1992.
- [10] P.R. Krishnaiah (ed.): *Multivariate Analysis 2*, New York: Academic Press, 1969.
- [11] A.S. Lapedes & R. Farber: “Nonlinear Signal Processing Using Neural Networks, Prediction and System Modeling,” *Technical Report LA-UR-87*, Los Alamos National Laboratory, 1987.
- [12] J.M. Mendel: *Lessons in Digital Estimation Theory* Englewood Cliffs, New Jersey: Prentice-Hall, 1987.
- [13] J. Moody & C.J. Darken: “Fast Learning in Networks of Locally-Tuned Processing Units,” *Neural Computation*, no. 1, pp. 281–294, 1989.
- [14] M. Schetzen: *The Volterra and Wiener Theories of Nonlinear Systems*, Malabar, Florida: Robert E. Krieger Publishing Company, 1989.
- [15] H. Tong & K.S. Lim: “Threshold Autoregression, Limit Cycles and Cyclical Data,” *Journal of the Royal Statistical Society*, vol. 42, no. 3, pp. 245–292, 1980.
- [16] B. Widrow & S.D. Stearns: *Adaptive Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1985.
- [17] Kai-Bor Yu: “Recursive Updating the Eigenvalue Decomposition of a Covariance Matrix,” *IEEE Transactions on Signal Processing*, vol. 39, no. 5, pp. 1136–1145, 1991.

APPENDIX I

PAPER 2: A GENERALIZATION ERROR ESTIMATE

This appendix contains a reprint¹ of the paper: “A Generalization Error Estimate for Nonlinear Systems,” in S.Y. Kung, F. Fallside, J. Aa. Sørensen & C.A. Kamm (eds.), *Proceedings of the 1992 IEEE-SP: Neural Networks for Signal Processing 2*, Piscataway, New Jersey: IEEE Service Center, 1992, pp. 29–38.

¹The typesetting is slightly changed and a few misprints is rectified.

A Generalization Error Estimate for Nonlinear Systems

Jan Larsen

The Computational Neural Network Center
Electronics Institute, Building 349
Technical University of Denmark
DK-2800 Lyngby, Denmark

I.1 Introduction

Evaluation of the quality of an estimated nonlinear model, e.g., a neural network, is important for the purpose of selecting a proper architecture. In this work the employed quality measure is the generalization error (expected squared prediction error). The topic of the paper is to derive an estimate of the generalization error for *incomplete* models, i.e., models which are not capable of modeling the present nonlinear relationship perfectly.

Consider the following discrete nonlinear system:

$$y(k) = g(\mathbf{x}(k)) + \varepsilon(k) \quad (\text{I.1})$$

where the scalar output, $y(k)$, (k is the discrete time index) is generated as the sum of a nonlinear mapping, $g(\cdot)$, of the input vector $\mathbf{x}(k)$ and the inherent noise $\varepsilon(k)$. In a signal processing context the input vector may e.g., represent a tapped delay line, i.e., $\mathbf{x}(k) = [x(k), x(k-1), \dots, x(k-L+1)]^\top$ (\top is the transpose operator).

ASSUMPTION I.1 The input $\mathbf{x}(k)$ is assumed to be a strictly stationary sequence and $\varepsilon(k)$ a white, strictly stationary sequence with zero mean and variance σ_ε^2 . Furthermore, $\mathbf{x}(k)$ is assumed independent of $\varepsilon(k)$, $\forall k$.

Let \mathcal{F} be a set of nonlinear functionals parameterized by an m -dimensional vector $\mathbf{w} = [w_1, w_2, \dots, w_m]^\top$. In general it is assumed that the functionals are nonlinear in \mathbf{w} . Feed-forward neural networks with hidden units are examples of \mathcal{F} . Let $f(\cdot) \in \mathcal{F}$. The model of Eq. (I.1) becomes:

$$y(k) = f(\mathbf{x}(k); \mathbf{w}) + e(k; \mathbf{w}) \quad (\text{I.2})$$

The prediction of $y(k)$, say $\hat{y}(k)$, is: $\hat{y}(k) = f(\mathbf{x}(k); \mathbf{w})$. When referring to a nonlinear model $\hat{y}(k)$ is considered to be nonlinear in \mathbf{w} .

DEFINITION I.1 If $\exists \mathbf{w}^\circ: g(\mathbf{x}(k)) \equiv f(\mathbf{x}(k); \mathbf{w}^\circ)$ the model is signified as complete otherwise as incomplete. \mathbf{w}° is denoted the true weight vector.

Usually the lack of knowledge concerning the structure of $g(\cdot)$ precludes the possibility of suggesting a complete model with a finite m . Consequently, it is claimed that incomplete models are the common case.

Given a training set: $\mathcal{T} = \{\mathbf{x}(k), y(k)\}$, $k = 1, 2, \dots, N$, where N is the training set size, the model is estimated by minimizing some cost function, say $S_N(\mathbf{w})$. In this work a

least squares (LS) cost is employed:

$$S_N(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N e^2(k; \mathbf{w}) = \frac{1}{N} \sum_{k=1}^N (y(k) - f(\mathbf{x}(k); \mathbf{w}))^2 \quad (\text{I.3})$$

The training performance $S_N(\hat{\mathbf{w}})$ is usually not a reliable measure of the quality of a model because it depends on the actual training set. A reliable quality measure is the *generalization error*, G , (e.g., [7]) which is defined as the expected, squared prediction error on a test sample, $\{\mathbf{x}_t, y_t\}$ (denoting t for test), which is *independent* of the training set but with identical distribution.

$$G(\mathbf{w}) = E_{\mathbf{x}_t, \varepsilon_t} \left\{ [y_t - f(\mathbf{x}_t; \mathbf{w})]^2 \right\} \quad (\text{I.4})$$

$E_{\mathbf{x}_t, \varepsilon_t} \{\cdot\}$ denotes expectation with respect to the joint p.d.f. of $[\mathbf{x}_t, \varepsilon_t]$. Note the dependence on both $f(\cdot)$ and \mathbf{w} .

In the literature several attempts have been made in order to estimate the generalization error of both linear and nonlinear models, for instance [1] and [3] which focus on complete models, while [5] and [7] focus on incomplete models which are claimed to be the most common.

In [5] a generalization error estimator for linear incomplete models is developed. The estimate requires knowledge of the estimated parameters $\hat{w}_i, i = m+1, m+2, \dots, m^\circ$ where m° denotes the dimension for which the model becomes complete. Unfortunately, these estimated parameters are not accessible when fitting with only m parameters. Therefore, the final result of [5] is essentially the *FPE*-criterion [1].

The *GPE* estimator [7] is claimed to estimate the generalization error for both nonlinear and incomplete (in [7] denoted biased) models when using the sum of S_N (LS-term) and a regularizing term as the cost function. However, in the next section, which deals with a new generalization error estimate with validity for both incomplete and nonlinear models, it is established that the error, $e(k; \mathbf{w})$, and the input, $\mathbf{x}(k)$, are *not* independent unless the model is complete. This dependence is not taken into account in [7].

I.2 Estimate for Incomplete, Nonlinear Models

In this section a new **g**eneralization **e**rror estimate for incomplete **n**onlinear models, called *GEN*, is introduced. The aim is to estimate $G(\hat{\mathbf{w}})$, i.e., how well the estimated model, $f(\mathbf{x}(k); \hat{\mathbf{w}})$, generalizes. In order to evaluate Eq. (I.4) the nonlinear system Eq. (I.1) must be known. Secondly, knowledge of the input and error distributions is required. However, these assumptions are not met in general; the only knowledge of the actual system is obtained implicitly from the acquired data. For that reason the presented generalization error estimate is based on training data solely.

To ensure the validity of the *GEN*-estimate the following assumptions must be satisfied:

ASSUMPTION I.2 Define Ω^m as the compact set which the weights minimizing the cost, S_N , belong to. Assume the existence of a covering of Ω^m in compact subsets, i.e., $\Omega^m = \bigcup_i \Omega^m(i)$, such that the estimator $\hat{\mathbf{w}}(i) \in \Omega^m(i)$ uniquely minimizes $S_N(\mathbf{w})$ within the partition $\Omega^m(i)$ and further

$$\frac{\partial S_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w}} = \mathbf{0}, \quad \mathbf{a}^\top \frac{\partial^2 S_N(\hat{\mathbf{w}}(i))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}, \quad \forall i \quad (\text{I.5})$$

Observe that $\{\widehat{\mathbf{w}}(i)\}$ may contain both local and global minima, even though the global minima are preferred. The occurrence of multiple minima is in evidence among feed-forward neural networks, due to e.g., symmetries which cause multiple minima in the cost function, see e.g., [4].

ASSUMPTION I.3 Assume a covering $\Omega^m = \bigcup_i \Omega_*^m(i)$, such that the optimal weights $\mathbf{w}^*(i) \in \Omega_*^m(i)$ uniquely minimize $G(\mathbf{w})$ within the partition $\Omega_*^m(i)$ and further

$$\frac{\partial G(\mathbf{w}^*(i))}{\partial \mathbf{w}} = \mathbf{0}, \quad \mathbf{a}^\top \frac{\partial^2 G(\mathbf{w}^*(i))}{\partial \mathbf{w} \partial \mathbf{w}^\top} \mathbf{a} > 0, \quad \forall \mathbf{a} \neq \mathbf{0}, \quad \forall i \quad (\text{I.6})$$

Note that the optimal weight vectors reflect the “best” models within the actual set \mathcal{F} . That is, the models obtained by training on an infinite training set corresponding to minimal generalization error as $\lim_{N \rightarrow \infty} S_N(\mathbf{w}_m) = G(\mathbf{w})$ (provided that $e^2(k; \mathbf{w})$ is mean-ergodic).

ASSUMPTION I.4 Let the minimization of S_N on the training set result in the estimate: $\widehat{\mathbf{w}}^2$. Assume the existence of an optimal weight vector \mathbf{w}^* such that the remainder of the following second order Taylor series expansion is negligible.

$$G(\widehat{\mathbf{w}}) \approx G(\mathbf{w}^*) + \Delta \mathbf{w}^\top \mathbf{H}(\mathbf{w}^*) \Delta \mathbf{w} \quad (\text{I.7})$$

where $\Delta \mathbf{w} = \widehat{\mathbf{w}} - \mathbf{w}^*$, $\mathbf{H}(\mathbf{w}^*)$ is the nonsingular (by Eq. (I.6)) Hessian matrix

$$\mathbf{H}(\mathbf{w}^*) = \frac{1}{2} \frac{\partial^2 G(\mathbf{w}^*)}{\partial \mathbf{w} \partial \mathbf{w}^\top} = E_{\mathbf{x}_t, \varepsilon_t} \left\{ \boldsymbol{\psi}_t(\mathbf{w}^*) \boldsymbol{\psi}_t^\top(\mathbf{w}^*) - \boldsymbol{\Psi}_t(\mathbf{w}^*) e_t(\mathbf{w}^*) \right\}, \quad (\text{I.8})$$

$\boldsymbol{\psi}_t(\mathbf{w}^*) = \partial f(\mathbf{x}_t; \mathbf{w}^*) / \partial \mathbf{w}$ and $\boldsymbol{\Psi}_t(\mathbf{w}^*) = \partial \boldsymbol{\psi}(\mathbf{x}_t; \mathbf{w}^*) / \partial \mathbf{w}^\top$. Note that $\partial G(\mathbf{w}^*) / \partial \mathbf{w} = \mathbf{0}$ according to Eq. (I.6).

Further assume that the remainder of expanding S_N around $\widehat{\mathbf{w}}(i)$ to the second order is negligible, i.e.,

$$S_N(\mathbf{w}^*) \approx S_N(\widehat{\mathbf{w}}) + \Delta \mathbf{w}^\top \mathbf{H}_N(\widehat{\mathbf{w}}) \Delta \mathbf{w} \quad (\text{I.9})$$

where $\mathbf{H}_N(\widehat{\mathbf{w}})$ is the nonsingular Hessian given by

$$\mathbf{H}_N(\widehat{\mathbf{w}}) = \frac{1}{2} \frac{\partial^2 S_N(\widehat{\mathbf{w}})}{\partial \mathbf{w} \partial \mathbf{w}^\top} = \frac{1}{N} \sum_{k=1}^N \boldsymbol{\psi}(k; \widehat{\mathbf{w}}) \boldsymbol{\psi}^\top(k; \widehat{\mathbf{w}}) - \boldsymbol{\Psi}(k; \widehat{\mathbf{w}}) e(k; \widehat{\mathbf{w}}), \quad (\text{I.10})$$

$\boldsymbol{\psi}(k; \widehat{\mathbf{w}}) = \partial f(\mathbf{x}(k); \widehat{\mathbf{w}}) / \partial \mathbf{w}$ and $\boldsymbol{\Psi}(k; \widehat{\mathbf{w}}) = \partial \boldsymbol{\psi}(\mathbf{x}(k); \widehat{\mathbf{w}}) / \partial \mathbf{w}^\top$. Note that $\partial S_N(\widehat{\mathbf{w}}) / \partial \mathbf{w} = \mathbf{0}$ according to Eq. (I.5).

ASSUMPTION I.5 $\mathbf{x}(k)$ is an M -dependent stationary sequence, i.e., $\mathbf{x}(k)$, $\mathbf{x}(k + \tau)$ are independent $\forall \tau > M$ (A weaker assumption aims at $\mathbf{x}(k)$ being a strongly mixing sequence [Rosenblatt 85, p. 62]).

ASSUMPTION I.6 Assume large training sets, i.e., $N \rightarrow \infty$. In particular: $N > M$. Further, assume that m is finite.

²Note that the weight estimate is highly dependent on the chosen weight estimation algorithm due to local optimization, initial conditions, etc. An alternative algorithm used on the same training set may therefore result in a different weight estimate.

DEFINITION I.2 *The generalization error estimate for nonlinear systems, GEN, is defined as a consistent ($N \rightarrow \infty$) estimator of Γ (the expectation of the generalization error w.r.t. the training set),*

$$\Gamma = E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\} \quad (\text{I.11})$$

where $\hat{\mathbf{w}}$ is the actual weight estimate and \mathcal{T} is the training set.

THEOREM I.1 *Assume that the nonlinear system is described by Eq. (I.1). If assumptions I.1 – I.6 hold and the model in Eq. (I.2) is **incomplete** then the GEN-estimate is given by:*

$$GEN = S_N(\hat{\mathbf{w}}) + \frac{2}{N} \cdot \text{tr} \left[\left(\mathbf{R}(0) + \sum_{\tau=1}^M \frac{N-\tau}{N} (\mathbf{R}(\tau) + \mathbf{R}^\top(\tau)) \right) \mathbf{H}_N^{-1}(\hat{\mathbf{w}}) \right] \quad (\text{I.12})$$

where the correlation matrices $\mathbf{R}(\tau)$, $\tau = 0, 1, \dots, M$, are calculated as:

$$\mathbf{R}(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} \boldsymbol{\psi}(k; \hat{\mathbf{w}}) e(k; \hat{\mathbf{w}}) \boldsymbol{\psi}^\top(k + \tau; \hat{\mathbf{w}}) e(k + \tau; \hat{\mathbf{w}}) \quad (\text{I.13})$$

Sketch of Proof The basis of the proof is the Taylor series expansions in Eq. (I.7), (I.9). Taking the expectation, $E_{\mathcal{T}}\{\cdot\}$ (i.e., w.r.t. the training set) of these equations it is possible to substitute Eq. (I.9) into (I.7) and thus express $E_{\mathcal{T}}\{G(\hat{\mathbf{w}})\}$ in terms of training data. This is due to the relation: $E_{\mathcal{T}}\{S_N(\mathbf{w}^*)\} = E_{\mathcal{T}}\{G(\mathbf{w}^*)\}$. When evaluating the expectations it is important to notice that the error (cf. Eq. (I.1) and (I.2))

$$e(k; \mathbf{w}) = \varepsilon(k) + g(\mathbf{x}(k)) - f(\mathbf{x}(k); \mathbf{w}) \quad (\text{I.14})$$

depends both on $\mathbf{x}(k)$ and $\varepsilon(k)$ unless the model is complete and \mathbf{w}^* is the global optimum since $g(\mathbf{x}) \equiv f(\mathbf{x}(k); \mathbf{w}^*)$ in this case³. In [6] the details of the proof are given and the estimate is further extended to treat other cost functions, for instance the LS-cost with inclusion of a weight decay term as in [7]. Note, that the derivation is valid even when dealing with noise free systems, i.e., $\sigma_\varepsilon^2 = 0$.

THEOREM I.2 *If the system in Eq. (I.1) is **linear**, the model Eq. (I.2) is linear and **complete**, \mathbf{w}^* in Assumption I.4 is the global minimum, and $\sigma_\varepsilon^2 \neq 0$ then the GEN-estimate coincides with the FPE-Criterion [1]:*

$$GEN = FPE = \frac{N+m}{N-m} S_N(\hat{\mathbf{w}}) \quad (\text{I.15})$$

Proof See the sketch above and [6].

I.3 Numerical Experiments

In this section the validity of the proposed generalization error estimate is tested by comparison with the *FPE*-estimate and the leave-one-out cross-validation technique. A linear system and a simple neural network is under consideration.

³Note that $g - f$ may be equal to a constant which is independent of \mathbf{x} . However, this case never occurs if the model contains a bias term.

I.3.1 Linear System

The linear system is given by:

$$y(k) = y^\circ(k) + \varepsilon(k) = [x(k), x^2(k)]\mathbf{w}^\circ + \varepsilon(k) \quad (\text{I.16})$$

where $\mathbf{w}^\circ = [1, 1]^\top$. The input $x(k) = \sum_{n=0}^{15} b_n u(k-n)$ where $u(k)$ is an i.i.d. Gaussian sequence with zero mean and unit variance. b_n is designed to implement a low-pass filter⁴ with normalized cutoff frequency 0.01. $x(k)$ is consequently colored and M-dependent (see Ass. I.5 above) with $M = 15$. $\varepsilon(k)$ is an i.i.d. Gaussian noise sequence with zero mean, $\sigma_\varepsilon^2 = 0.2 \cdot E_{x(k)}\{(y^\circ)^2(k)\}$, and independent of $u(k)$. The model used is incomplete and given by:

$$y(k) = wx(k) + e(k; w) \quad (\text{I.17})$$

GEN is compared to two different methods for estimating the generalization error. First, a comparison with the *FPE*-estimate which is much less computationally complex according to Eq. (I.12) and (I.15).

Secondly, comparison with the leave-one-out cross-validation method [2], [4] is performed. Within this method training is replicated N times. The j 'th training is performed on the data: $\{\mathbf{x}(k), y(k)\}$, $k = 1, 2, \dots, j-1, j+1, \dots, N$, $j = 1, 2, \dots, N$ ⁵ resulting in the estimate $\hat{\mathbf{w}}^{(j)}$. $e^2(j, \hat{\mathbf{w}}^{(j)})$ is a qualified estimate of the generalization error and consequently G is estimated as:

$$L = \frac{1}{N} \sum_{j=1}^N e^2(j, \hat{\mathbf{w}}^{(j)}) \quad (\text{I.18})$$

Knowing the details of the system Eq. (I.16) it is possible to compute the true generalization error $G(\hat{w})$ according to Eq. (I.4). Let $E\{\cdot\}$ denote expectation w.r.t. x_t and ε_t . Now, noting that x_t is Gaussian:

$$\begin{aligned} G(\hat{w}) &= E \left\{ \left[w_1^\circ x_t + w_2^\circ x_t^2 + \varepsilon_t - \hat{w} x_t \right]^2 \right\} \\ &= (w_1^\circ - \hat{w})^2 E\{x_t^2\} + 3(w_2^\circ E\{x_t^2\})^2 + \sigma_\varepsilon^2 \end{aligned} \quad (\text{I.19})$$

In order to simulate the statistical variations of the training sets Q independent training sets: $\{x^{(q)}(k), y^{(q)}(k)\}$, $q = 1, 2, \dots, Q$, is generated for every specific training set size, N . Next, Q models are estimated by (the LS-estimator):

$$\hat{w}^{(q)} = \left[\sum_{k=1}^N (x^{(q)}(k))^2 \right]^{-1} \cdot \sum_{k=1}^N x^{(q)}(k) y^{(q)}(k) \quad (\text{I.20})$$

Let \hat{G} be a specific generalization error estimator, i.e., *GEN*, *FPE*, *C* or *L*. For the purpose of comparison the *relative average deviation (RAD)* is defined as:

$$RAD = \frac{\langle G(\hat{\mathbf{w}}^{(q)}) \rangle - \langle \hat{G}(\hat{\mathbf{w}}^{(q)}) \rangle}{\langle G(\hat{\mathbf{w}}^{(q)}) \rangle} \quad (\text{I.21})$$

where $\langle \cdot \rangle$ denotes the average with respect to the Q realizations.

The result of comparing the *RAD*'s of *GEN* and *FPE* is shown in Fig. I.1. Averaging

⁴The design is performed by the MATLAB (The MathWorks, Inc.) M-file "fir1" which uses a Hamming windowed ideal impulse response (i.e., $\text{sinc}(x)$).

⁵Note that $\min(k) = 2$ for $j = 1$ and $\max(k) = N - 1$ for $j = N$.

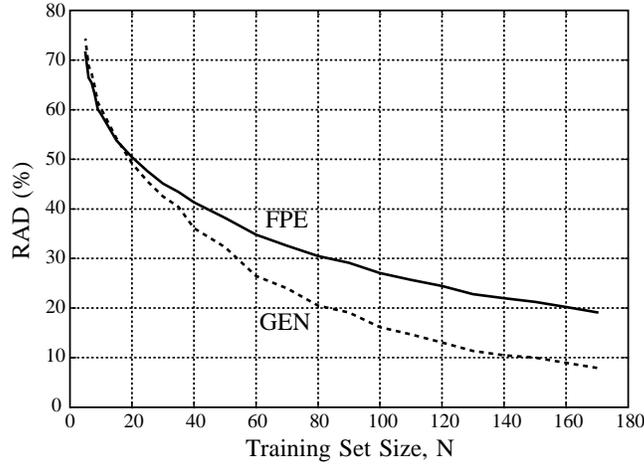


Figure I.1: Comparison of the RAD 's of GEN and FPE for the linear model Eq. (I.17).

was done with:

$$Q = \begin{cases} 30000 & 5 \leq N \leq 9 \\ 20000 & 10 \leq N \leq 170 \end{cases}$$

When N is small compared to M $\mathbf{R}(\tau)$ (cf. Eq. (I.13)) will be rather noisy. Therefore $\tau = 1, 2, \dots, \min\{M, [N/10]\}$ was used, where $[\cdot]$ denotes rounding to the nearest integer. Using a standard Gaussian test (details omitted) it is seen that the RAD of the GEN -estimate is significantly⁶ better than the RAD of the FPE -estimate for all $N > 15$ and roughly equal as $N \leq 15$. However, the RAD only shows the average performance. The estimator with the best RAD performance may still not be the preferred estimator. In order to elucidate the variations in the estimates the probability that GEN is closer than FPE to the average of the true generalization, $\langle G \rangle$, was estimated. That is,

$$\gamma = P\{|\langle G \rangle - GEN| < |\langle G \rangle - FPE|\} \quad (\text{I.22})$$

It was found that $\gamma > 0.5 \forall N \geq 25$ and $\gamma \approx 0.75$ when $N \geq 40$. Consequently, one may prefer the GEN -estimator when $N > 25$.

Next, GEN is tested against leave-one-out cross-validation and averaging was done with:

$$Q = \begin{cases} 10000 & 5 \leq N \leq 9 \\ 5000 & 10 \leq N \leq 100 \end{cases}$$

The result is shown in Fig. I.2. As $N > 15$ the GEN -estimate is significantly better than leave-one-out cross-validation as the RAD is lower. Further⁷, $\gamma > 0.5$ as $N \geq 30$ and $\gamma \approx 0.75$ for $N \geq 40$. This is in spite of the fact that the computational complexity of the L -estimate normally is greater than that of the GEN -estimate. The number of

⁶Here and in the following a 0.5% significance level is employed.

⁷ FPE is replaced by L in Eq. (I.22).

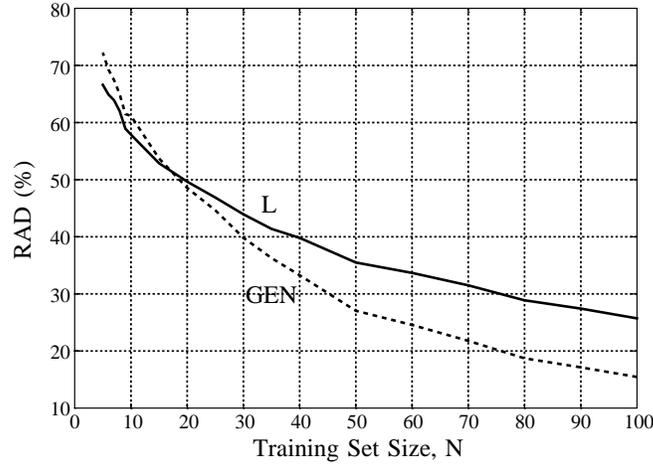


Figure I.2: Comparison of the *RAD*'s of *GEN* and leave-one-out cross-validation, *L*, for the linear model Eq. (I.17).

multiplications involved in the computation of the *GEN*-estimate is approximately MNm^2 whereas the *L*-estimate requires in the order of N^2m^2 multiplications (this is due to the fact that training is replicated N times).

I.3.2 Simple Neural Network

Consider a simple nonlinear system which consists of a single neuron:

$$y(k) = y^\circ(k) + \varepsilon(k) = h(\mathbf{x}^\top(k)\mathbf{w}^\circ) + \varepsilon(k), \quad (\text{I.23})$$

$$h(z) = \exp\left(-\left(\frac{z-\nu}{\eta}\right)^2\right) - \exp\left(-\left(\frac{z+\nu}{\eta}\right)^2\right) \quad (\text{I.24})$$

where $\mathbf{w}^\circ = [3, 3]^\top$. Let $\mathbf{u}(k)$ be a two-dimensional i.i.d. Gaussian sequence with zero mean and $E\{u_i^2(k)\} = 1$, $E\{u_1(k)u_2(k)\} = 0.5$. b_n is given as in the preceding subsection and $x_i(k) = \sum_{n=0}^{15} b_n u_i(k-n)$, $i = \{1, 2\}$. $\varepsilon(k)$ is an i.i.d. Gaussian noise sequence with zero mean, $\sigma_\varepsilon^2 = 0.1 \cdot E_{\mathbf{x}(k)}\{(y^\circ)^2(k)\}$, and independent of $u_i(k)$. The activation function $h(z)$ is chosen to be a sum of two Gaussian functions in order to enable the evaluation of the true generalization error Eq. (I.4). In this simulation: $\nu = 2$ and $\eta = 1$. The employed incomplete nonlinear model of Eq. (I.23) is:

$$y(k) = h(wx_1(k)) + e(k; w) \quad (\text{I.25})$$

According to Eq. (I.4), (I.23), and (I.25) ($E\{\cdot\}$ w.r.t. $[\mathbf{x}_t, \varepsilon_t]$):

$$\begin{aligned} G(\hat{w}) &= E\left\{\left[\varepsilon_t + h(\mathbf{x}^\top(k)\mathbf{w}^\circ) - h(\hat{w}x_1(k))\right]^2\right\} \\ &= E\left\{\left[h(\mathbf{x}^\top(k)\mathbf{w}^\circ) - h(\hat{w}x_1(k))\right]^2\right\} + \sigma_\varepsilon^2 \end{aligned} \quad (\text{I.26})$$

Evaluation of the first term in Eq. (I.26) is possible, however, due to the extent of the derivation it is omitted, see [6] for further details.

The parameter w in Eq. (I.25) is estimated using a modified Gauss-Newton algorithm [Seber & Wild 89, Ch. 14]. That is, for each training set $\{x_1^{(q)}(k), y^{(q)}(k)\}$, $q = 1, 2, \dots, Q$ (below the q index is omitted for simplicity):

$$w_{(i+1)} = w_{(i)} + \mu \widetilde{\mathbf{H}}_N^{-1}(w_{(i)}) \nabla(w_{(i)}), \quad (\text{I.27})$$

$$\widetilde{\mathbf{H}}_N(w_{(i)}) = \sum_{k=1}^N \left[h'(w_{(i)} x_1(k)) \cdot x_1(k) \right]^2, \quad (\text{I.28})$$

$$\nabla(w_{(i)}) = \sum_{k=1}^N h'(w_{(i)} x_1(k)) \cdot x_1(k) \cdot e(k; w_{(i)}) \quad (\text{I.29})$$

where $0 \leq \mu \leq 1$ is the step-size and $'$ denotes the derivative. For each iteration, i , μ is adjusted in order to ensure: $S_N(w_{(i+1)}) < S_N(w_{(i)})$. The employed stopping criterion [Seber & Wild 89, Sec. 14.4] was: $(S_N(w_{(i+1)}) - S_N(w_{(i)})) / S_N(w_{(i)}) < 10^{-12}$.

The result of comparing *GEN* to *FPE* is shown in Fig. I.3 ($Q = 5000$). It is observed

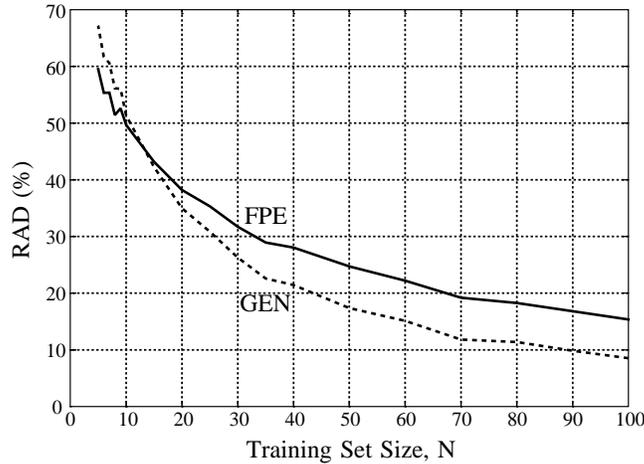


Figure I.3: Comparison of the *RAD*'s of *GEN* and *FPE* for the nonlinear model Eq. (I.25).

that the *RAD* of the *GEN*-estimate is significantly better than that of the *FPE*-estimate for all $N > 10$ and that (cf. Eq. (I.22)) $\gamma > 0.5$ as $N \geq 15$ and $\gamma \approx 0.6$ for $N > 15$.

I.4 Conclusion

In this paper a new estimate (*GEN*) of the generalization error is presented. The estimator is valid for both *incomplete* and *nonlinear* models. An incomplete model is characterized in that it does not model the actual nonlinear relationship perfectly. The estimator can be viewed as an extension of the *FPE* and *GPE* estimators [1], [7]. The *GEN*-estimator

has been evaluated by simulating incomplete models of linear and simple neural network systems respectively. Within the linear system *GEN* is compared to the Final Prediction Error (*FPE*) criterion and the leave-one-out cross-validation technique. It was found that the *GEN*-estimate of the true generalization error is less biased on the average. Further the probability, γ , of *GEN* being closer to the true generalization error than the other estimators was estimated, and it was found that $\gamma > 0.5$ within a large range of training set sizes. Comparing the *GEN*-estimate to *FPE* when simulating a simple neural network shows that *GEN* is less biased on the average and that $\gamma \approx 0.6$ when using training sets of sizes greater than 15. In summary it is concluded that *GEN* is an applicable alternative in estimating the generalization at the expense of an increased complexity. However, the leave-one-out cross-validation estimate which possess a higher complexity was not able to outperform *GEN* in the chosen example.

I.5 Acknowledgments

The author would like to thank Lars Kai Hansen, Nils Hoffmann, Peter Koefoed Møller and John Aasted Sørensen for helpfull comments on this paper. This work is partly supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center.

References

- [1] H. Akaike, "Fitting Autoregressive Models for Prediction," *Ann. Inst. Stat. Math.*, vol. 21, pp. 243–247, 1969.
- [2] N.R. Draper & H. Smith, *Applied Regression Analysis*, New York, New York: John Wiley & Sons, 1981.
- [3] D.B. Fogel, "An Information Criterion for Optimal Neural Network Selection," *IEEE Transactions on Neural Networks*, vol. 2, no. 5, Sept. 1991.
- [4] L.K. Hansen & P. Salomon, "Neural Network Ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, Oct. 1990.
- [5] R. Kannurpatti & G.W. Hart, "Memoryless Nonlinear System Identification With Unknown Model Order," *IEEE Transaction on Information Theory*, vol. 37, no. 5, Sept. 1991.
- [6] J. Larsen, *Design of Neural Network Filters*, Ph.D. Thesis, Electronics Institute, Technical University of Denmark, 1993.
- [7] J. Moody, "Note on Generalization, Regularization, and Architecture Selection in Nonlinear Learning Systems", in *Proceedings of the first IEEE Workshop on Neural Networks for Signal Processing*, Eds. B.H. Juang, S.Y. Kung & C.A. Kamm, Piscataway, New Jersey: IEEE, 1991, pp. 1–10.
- [8] M. Rosenblatt, *Stationary Sequences and Random Fields*, Boston, Massachusetts: Birkhäuser, 1985.
- [9] G.A.F. Seber & C.J. Wild, *Nonlinear Regression*, New York, New York: John Wiley & Sons, 1989.

BIBLIOGRAPHY

- M. ABRAMOWITZ & I.A. STEGUN (eds.): *Handbook of Mathematical Functions*, New York, New York: Dover Publications, Inc., 1972.
- H. AKAIKE: "Fitting Autoregressive Models for Prediction," *Annals of the Institute of Statistical Mathematics*, vol. 21, pp. 243–247, 1969.
- H. AKAIKE: "A New Look at the Statistical Model Identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, Dec. 1974.
- T.W. ANDERSON: *An Introduction to Multivariate Statistical Analysis*, New York, New York: John Wiley & Sons, 1984.
- A.D. BACK & A.C. TSOI: "FIR and IIR Synapses, A New Neural Network Architecture for Time Series Modeling," *Neural Computation*, vol. 3, pp. 375–385, 1991.
- A. BARRON: "Predicted Squared Error: A Criterion for Automatic Model Selection," in S. Farlow (ed.) *Self-Organizing Methods in Modeling*, New York, New York: Marcel Dekker, 1984.
- R. BATTITI: "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method," *Neural Computation*, vol. 4, pp. 141–166, 1992.
- J.S. BENDAT: *Nonlinear Systems Analysis and Identification*, New York, New York: John Wiley & Sons, 1990.
- J.S. BENDAT & A.G. PIERSOL: *Random Data, Analysis and Measurement Procedures*, New York, New York: John Wiley & Sons, 1986.
- G.J. BIERMAN: "Measurement Updating using the U-D Factorization," in *Proceedings of the IEEE International Conference on Decision and Control*, 1975, pp. 337–346.
- S.A. BILLINGS & W.S.F. VOON: "Structure Detection and Model Validity Tests in the Identification of Nonlinear Systems," *IEE Proceedings*, vol. 130, part D, no. 4, pp. 193–199, July 1983.
- S.A. BILLINGS & W.S.F. VOON: "A Prediction-Error and Stepwise-Regression Estimation Algorithm for Non-Linear Systems," *International Journal of Control*, vol. 44, no. 3, pp. 803–822, 1986.
- C. BISHOP: "Improving the Generalization Properties of Radial Basis Function Neural Networks," *Neural Computation*, vol. 3, pp. 579–588, 1991.
- D.S. BROOMHEAD & G.P. KING: "Extracting Qualitative Dynamics from Experimental Data," *Physica D*, vol. 20, p. 217, 1987.
- B. CARLSSON, A. AHLÉN & M. STERNAD: "Optimal Differentiation Based on Stochastic Signal Models," *IEEE Transactions on Signal Processing*, vol. 39, no. 2, 341–353, 1991.

- B. CHANANE & S.P. BANKS: "Rational Expansion for Non-linear Input-Output Maps," *International Journal of Control*, vol. 51, no. 6, pp. 1241–1250, 1990.
- S. CHEN & S.A. BILLINGS: "Representation of Non-linear Systems: The NARMAX model," *International Journal of Control*, vol. 49, no. 3, pp. 1013–1032, 1989.
- S. CHEN, S.A. BILLINGS & P.M. GRANT: "Non-Linear System Identification using Neural Networks," *International Journal of Control*, vol. 51, no. 6, pp. 1191–1214, 1990a.
- S. CHEN, C.F.N. COWAN, S.A. BILLINGS & P.M. GRANT: "Parallel Recursive Prediction Error Algorithm for Training Layered Neural Networks," *International Journal of Control*, vol. 51, no. 6, pp. 1215–1228, 1990b.
- S. CHEN, G.J. GIBSON, C.F.N. COWAN, & P.M. GRANT: "Adaptive Equalization of Finite Non-Linear Channels Using Multilayer Perceptrons," *Signal Processing*, vol. 20, pp. 107–119, 1990c.
- J. CHEN & J. VANDEWALLE: "Study of Adaptive Nonlinear Echo Canceler With Volterra Expansion," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1989*, Glasgow, Scotland, May 23–26, 1989, pp. 1376–1379.
- V. CHERKASSKY: "Neural Networks and Nonparametric Regression," in S.Y. Kung, F. Fallside, J. Aa. Sørensen & C.A. Kamm (eds.) *Neural Networks for Signal Processing 2: Proceedings of the 1992 IEEE-SP Workshop*, Piscataway, New Jersey: IEEE, 1992, pp. 511–521.
- L.O. CHUA & R.L.P. YING: "Canonical Piecewise-Linear Analysis," *IEEE Transaction on Circuits and Systems*, vol. 30, no. 3, pp. 125–140, March 1983.
- N.E. COTTER: "The Stone-Weierstrass Theorem and Its Application to Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 4, pp. 290–295, Dec. 1990.
- G. CYBENKO: "Approximation by Superposition of a Sigmoidal Function," *Math. Control Signal Systems*, no. 2, pp. 303–314, 1989.
- E.D. DAHL: "Accelerated Learning using the Generalized Delta Rule," in *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, California, vol. 2, June 21–24, 1987, pp. 523–530.
- DEFENSE ADVANCED RESEARCH PROJECTS AGENCY: *DARPA Neural Network Study*, AFCEA International Press, 1988.
- N.R. DRAPER & H. SMITH: *Applied Regression Analysis*, New York, New York: John Wiley & Sons, 1981.
- H. DRUCKER & Y. LE CUN: "Improving Generalization Performance in Character Recognition," in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.) *Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop*, Piscataway, New Jersey: IEEE, 1991, pp. 198–207.
- R.O. DUDA & P.E. HART: *Pattern Classification and Scene Analysis*, New York, New York: John Wiley & Sons, 1973.
- S.E. FAHLMAN & C. LEBIERE: "The Cascade-Correlation Learning Architecture," in D.S. Touretzky (ed.) *Advances in Neural Information Processing Systems 2, Proceed-*

- ings of the 1989 Conference, San Mateo, California: Morgan Kaufmann Publishers, 1990, pp. 524–532.
- D.D. FALCONER: “Adaptive Equalization of Channel Nonlinearities in QAM Data Transmission Systems,” *The Bell Technical Journal*, vol. 57, no. 7, pp. 2589–2611, Sept. 1978.
- J.D. FARMER & J.J. SIDOROWICH: “Exploiting Chaos to Predict the Future and Reduce Noise,” *Technical Report LA-UR-88*, Los Alamos National Laboratory, 1988.
- D.B. FOGEL: “An Information Criterion for Optimal Neural Network Selection,” *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 490–498, Sept. 1991.
- J.H. FRIEDMAN: “Multivariate Adaptive Regression Splines,” *The Annals of Statistics*, vol. 19, no. 1, pp. 1–141, 1991.
- J.H. FRIEDMAN & W. STUETZLE: “Projection Pursuit Regression,” *Journal of the American Statistical Association*, vol. 76, no. 376, pp. 817–823, Dec. 1981.
- A.R. GALLANT: “Testing a Nonlinear Regression Specification: A Nonregular Case,” *Journal of the American Statistical Association*, vol. 72, no. 359, pp. 523–530, Sept. 1977.
- S.B. GELFAND, C.S. RAVISHANKAR & E.J. DELP: “A Tree-structured Piecewise Linear Adaptive Filter,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1991*, Toronto, Canada, May 14–17, 1991, pp. 2141–2144.
- S. GEMAN, E. BIENENSTOCK & R. DOURSAT: “Neural Networks and the Bias/Variance Dilemma,” *Neural Computation*, vol. 4, pp. 1–58, 1992.
- G.B. GIANNAKIS & A.V. DANDAWATE: “Linear and Non-Linear Adaptive Noise Cancelers,” in *Proceeding of the IEEE International Conference on Acoustics, Speech and Signal Processing 1990*, Albuquerque, New Mexico, April 3–6, 1990, pp. 1373–1376.
- A.H. HADDAD (ed.): *Nonlinear Systems: Processing of Random Signals - Classical Analysis*, Stroudsburg, PA: Dowden, Hutchinson & Ross, Inc., 1975.
- L.K. HANSEN: “Stochastic Linear Learning: Exact Test and Training Error Averages,” *Neural Networks*, vol. 6, pp. 393–396, 1993.
- L.K. HANSEN & P. SALAMON: “Neural Network Ensembles,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.
- E. HARTMANN & J.D. KEELER: “Predicting the Future: Advantages of Semilocal Units,” *Neural Computation*, vol. 3, pp. 566–578, 1991.
- E. HARTMANN, J.D. KEELER & J.M. KOWALSKI: “Layered Neural Networks with Gaussian Hidden Units as Universal Approximators,” *Neural Computation*, vol. 2, pp. 210–215, 1990.
- B. HASSIBI & D.G. STORK: “Second Order Derivatives for Network Pruning: Optimal Brain Surgeon,” in S.J. Hanson, J. Cowan & C. Giles (eds.) *Advances in Neural Information Processing Systems 5, Proceedings of the 1992 Conference*, San Mateo, California: Morgan Kaufmann Publishers, 1993, pp. 164–171.
- S. HAYKIN: *Adaptive Filter Theory*, Englewood Cliffs, New Jersey: Prentice-Hall, 1991.
- S. HAYKIN: *Neural Networks: A Comprehensive Foundation*, New York, New York: Macmillan College Publishing Company, 1994.

- R. HECT-NIELSEN: "Theory of the Backpropagation Neural Network," in *Proceedings of the International Joint Conference on Neural Networks*, Washington D.C., June 18–22, 1989, vol. 1, pp. 593–605.
- J. HERTZ, A. KROGH & R.G. PALMER: *Introduction to the Theory of Neural Computation*, Redwood City, California: Addison-Wesley Publishing Company, 1991.
- N. HOFFMANN: *Filtering with Nonlinear Adaptive Filters*, Ph.D. Thesis, Electronics Institute, The Technical University of Denmark, May 1992a.
- N. HOFFMANN: "A Neural Feed-forward Network with a Polynomial Nonlinearity," in S.Y. Kung, F. Fallside, J. Aa. Sørensen & C.A. Kamm (eds.) *Neural Networks for Signal Processing 2: Proceedings of the 1992 IEEE-SP Workshop*, Piscataway, New Jersey: IEEE, 1992b, pp. 49–58.
- N. HOFFMANN & J. LARSEN: "A Neural Architecture for Nonlinear Adaptive Filtering of Time Series," in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.) *Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop*, Piscataway, New Jersey: IEEE, 1991, pp. 533–542.
- K. HORNIK: "Approximation Capabilities of Multilayer Feedforward Networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.
- K. HORNIK, M. STINCHCOMBE & H. WHITE: "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, 1990.
- P.J. HUBER: "Projection Pursuit," *The Annals of Statistics*, vol. 13, no. 2, pp. 435–525, 1985.
- D.R. HUSH & B.G. HORNE: "Progress in Supervised Neural Networks," *IEEE Signal Processing Magazine*, pp. 8–39, Jan. 1993.
- R.A. JACOBS, M.I. JORDAN, S.J. NOWLAND & G.E. HINTON: "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, pp. 79–87, 1991.
- C. KAHLERT & L.O. CHUA: "A Generalized Piecewise-Linear Representation," *IEEE Transaction on Circuits and Systems*, vol. 37, no. 3, pp. 373–383, March 1990.
- S.M. KANG & L.O. CHUA: "A Global Representation of Multidimensional Piecewise-Linear Functions with Linear Partitions," *IEEE Transaction on Circuits and Systems*, vol. 25, no. 11, pp. 938–940, Nov. 1978.
- R. KANNURPATTI & G.W. HART: "Memoryless Nonlinear System Identification With Unknown Model Order," *IEEE Transaction on Information Theory*, vol. 37, no. 5, pp. 1441–1450, Sept. 1991.
- R.L. KASHYAP: "Inconsistency of the AIC Rule for Estimating the Order of Autoregressive Models," *IEEE Transactions on Automatic Control*, vol. 25, no. 5, pp. 996–998, Oct. 1980.
- K.I. KIM & E.J. POWERS: "A Digital Method of Modelling Quadratically Nonlinear Systems with a General Random Input," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 11, pp. 1758–1769, Nov. 1988.
- T. KOH & E.J. POWERS: "Second-Order Volterra Filtering and Its Application to Nonlinear System Identification," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, no. 6, Dec. 1985.

- S. KOLLIAS & D. ANASTASSIOU: "An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 8, pp. 1092–1101, Aug. 1989.
- M. KORENBERG & L.D. PAARMANN: "Orthogonal Approaches to Time-Series Analysis and System Identification," *IEEE Signal Processing Magazine*, vol. 8, no. 3, pp. 29–43, July 1991.
- A. KROGH & J.A. HERTZ: "A Simple Weight Decay Can Improve Generalization," in J.E. Moody, S.J. Hanson, R.P. Lippmann (eds.) *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 Conference*, San Mateo, California: Morgan Kaufmann Publishers, 1992, pp. 950–957.
- P.R. KRISHNAIAH (ed.): *Multivariate Analysis 2*, New York, New York: Academic Press, 1969.
- S. KULLBACK: *Information Theory and Statistic*, New York, New York: John Wiley & Sons, 1959.
- A.S. LAPEDES & R. FARBER: "Nonlinear Signal Processing Using Neural Networks, Prediction and System Modeling," *Technical Report LA-UR-87*, Los Alamos National Laboratory, 1987.
- J. LARSEN: "A Generalization Error Estimate for Nonlinear Systems," in S.Y. Kung, F. Fallside, J. Aa. Sørensen & C.A. Kamm (eds.) *Neural Networks for Signal Processing 2: Proceedings of the 1992 IEEE-SP Workshop*, Piscataway, New Jersey: IEEE, 1992, pp. 29–38.
- Y. LE CUN, J.S. DENKER & S.A. SOLLA: "Optimal Brain Damage," in D.S. Touretzky (ed.) *Advances in Neural Information Processing Systems 2, Proceedings of the 1989 Conference*, San Mateo, California: Morgan Kaufmann Publishers, 1990, pp. 598–605.
- E.L. LEHMANN: *Testing Statistical Hypotheses*, New York, New York: John Wiley & Sons, 1986.
- I.J. LEONTARITIS & S.A. BILLINGS: "Input-Output Parametric Models for Non-linear Systems, Part 1: Deterministic Non-linear Systems, Part 2: Stochastic Non-linear Systems," *International Journal of Control*, vol. 41, pp. 303–344, 1985.
- I.J. LEONTARITIS & S.A. BILLINGS: "Model Selection and Validation Methods for Non-Linear Systems," *International Journal of Control*, vol. 45, no. 1, pp. 311–341, 1987.
- E. LEVIN, R. GEWIRTZMAN & G.F. INBAR: "Neural Network Architecture for Adaptive System Modeling and Control," *Neural Networks*, vol. 4, pp. 185–191, 1991.
- J. LIN & R. UNBEHAUEN: "Adaptive Nonlinear Digital Filter with Canonical Piecewise-Linear Structure," *IEEE Transactions on Circuits and Systems* vol. 37, no. 3, pp. 347–353, March 1990.
- L. LJUNG: *System Identification: Theory for the User*, Englewood Cliffs, New Jersey: Prentice-Hall, 1987.
- L. LJUNG & T. SÖDERSTRÖM: *Theory and Practice of Recursive Identification*, Cambridge, Massachusetts: MIT-Press, 1983.

- D.J.C. MACKAY: "Bayesian Model Comparison and Backprop Nets," in J.E. Moody, S.J. Hanson, R.P. Lippmann (eds.) *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 Conference*, San Mateo, California: Morgan Kaufmann Publishers, 1992a, pp. 839–846.
- M.C. MACKAY & L. GLASS: "Oscillation and Chaos in Physiological Control Systems," *Science*, vol. 197, pp. 287–289, July 1977.
- J.L. MCCLELLAND & D.E. RUMELHART (eds.): *Parallel Distributed Processing, Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, Cambridge, Massachusetts: MIT Press, 1986.
- W.S. MCCULLOCH & W. PITTS: "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
- K. MADSEN & H.B. NIELSEN: *Solving Systems of Linear Equations*, Numerical Institute, Technical University of Denmark, Lyngby, 1972 (In Danish).
- K. MADSEN & H.B. NIELSEN: *Eigenvalues for Matrices*, Numerical Institute, Technical University of Denmark, Lyngby, 1975 (In Danish).
- M.A. MASNADI-SHIRAZI & N. AHMED: "Optimum Laguerre Networks for a Class of Discrete-Time Systems," *IEEE Transactions on Signal Processing*, vol. 39, no. 9, pp. 2104–2108, Sept. 1991.
- D.F. MARSHALL, W.K. JENKINS & J.J. MURPHY: "The Use of Orthogonal Transforms for Improving Performance of Adaptive Filters," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 4, pp. 474–483, April 1989.
- V.J. MATHEWS: "Adaptive Polynomial Filters," *IEEE Signal Processing Magazine*, vol. 8, no. 3, pp. 10–26, July 1991.
- MATLABTM Reference Guide*, 24 Prime Park Way, Natick, Massachusetts: The MathWorks, Inc., 1984–1994.
- M.L. MINSKY & S.A. PAPERT: *Perceptrons, Introduction to Computational Geometry*, Cambridge, Massachusetts: MIT Press, 1988.
- J. MOODY: "Note on Generalization, Regularization, and Architecture Selection in Nonlinear Learning Systems," in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.) *Proceedings of the first IEEE Workshop on Neural Networks for Signal Processing*, Piscataway, New Jersey: IEEE, 1991, pp. 1–10.
- J. MOODY: "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," in J.E. Moody, S.J. Hanson, R.P. Lippmann (eds.) *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 Conference*, San Mateo, California: Morgan Kaufmann Publishers, 1992, pp. 847–854.
- J. MOODY: "Prediction Risk and Architecture Selection for Neural Networks" in V. Cherkassky, J. H. Friedman & H. Wechsler (eds.) *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, Series F, vol. 136, Berlin, Germany: Springer-Verlag, 1994.
- J. MOODY & C.J. DARKEN: "Learning with Localized Receptive Fields," in D.S. Touretzky, G.E. Hinton & T.J. Sejnowski (eds.) *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo, California: Morgan-Kaufmann, 1988.

- J. MOODY & C.J. DARKEN: “Fast Learning in Networks of Locally-Tuned Processing Units,” *Neural Computation*, vol. 1, pp. 281–294, 1989.
- N. MURATA, S. YOSHIKAWA AND S. AMARI: “Network Information Criterion — Determining the Number of Hidden Units for an Artificial Neural Network Model,” *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 865–872, 1994.
- K.S. NARENDRA & K. PARTHASARATHY: “Identification and Control of Dynamical Systems Using Neural Networks,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, March 1990.
- M. NIRANJAN & V. KARDIRKAMANATHAN: “A Nonlinear Model for Time Series Prediction and Signal Interpolation,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1991*, Toronto, Canada, May 14–17, 1991, pp. 1713–1716.
- D. NGUYEN & B. WIDROW: “The Truck Backer-Upper: An Example of Self-Learning in Neural Networks,” in *Proceedings of the International Joint Conference on Neural Networks*, Washington D.C., June 18–22, 1989, pp. 357–363.
- S.J. NOWLAND & G.E. HINTON: “Simplifying Neural Networks by Soft Weight-Sharing,” *Neural Computation*, vol. 4, pp. 473–493, 1992.
- Z. OBRADOVIC & P. YAN: “Small Depth Polynomial Size Neural Networks,” *Neural Computation*, vol. 2, pp. 402–404, 1990.
- A.V. OPPENHEIM & R.W. SCHAFER: *Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1975.
- A.V. OPPENHEIM & R.W. SCHAFER: *Discrete-Time Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1989.
- S.J. ORFANIDIS: “Gram-Schmidt Neural Nets,” *Neural Computation*, vol. 2, pp. 116–126, 1990.
- A. PAPOULIS: *Probability, Random Variables, and Stochastic Processes*, New York, New York: McGraw-Hill, 1984a.
- A. PAPOULIS: *Signal Analysis*, New York, New York: McGraw-Hill, 1984b.
- S.R. PARKER & F.A. PERRY: “A Discrete ARMA Model for Nonlinear System Identification,” *IEEE Transactions on Circuit and Systems*, vol. 28, no. 3, pp. 224–233, March 1981.
- J. PARK & I.W. SANDBERG: “Universal Approximation Using Radial-Basis-Function Networks,” *Neural Computation*, vol. 3, pp. 246–257, 1991.
- J. PLATT: “A Resource-Allocating Network for Function Interpolation,” *Neural Computation*, vol. 3, pp. 213–225, 1991.
- P. PODDAR & K.P. UNNIKRISHNAN: “Non-Linear Prediction of Speech Signals Using Memory Neuron Networks,” in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.) *Proceedings of the first IEEE Workshop on Neural Networks for Signal Processing*, Piscataway, New Jersey: IEEE, 1991, pp. 395–405.
- W.H. PRESS, B.P. FLANNERY, S.A. TEUKOLSKY & W.T. VETTERLING: *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge, Massachusetts: Cambridge University Press, 1988.

- M.B. PRIESTLEY: *Non-linear and Non-stationary Time Series Analysis*, London, Great Britain: Academic Press Ltd., 1988.
- J.C. PRINCIPE, B. DE VRIES, J. KUO & P.G. DE OLIVEIRA: "Modeling Applications with the Focused Gamma Net," in J.E. Moody, S.J. Hanson, R.P. Lippmann (eds.) *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 Conference*, San Mateo, California: Morgan Kaufmann Publishers, 1992, pp. 143–150.
- C.R. RAO: *Linear Statistical Inference and Its Applications*, New York, New York: John Wiley & Sons, 1965.
- J. RISSANEN: "Modelling by Shortest Data Description," *Automatica*, vol. 14, pp. 465–471, 1978.
- H. ROBBINS & J. MONRO: "A Stochastic Approximation Method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.
- F. ROSENBLATT: "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, pp. 336–408, 1958.
- F. ROSENBLATT: *Principles of Neurodynamics*, New York, New York: Spartan, 1962.
- M. ROSENBLATT: *Stationary Sequences and Random Fields*, Boston, Massachusetts: Birkhäuser, 1985.
- T.D. SANGER: "A Tree-Structured Algorithm for Reducing Computations in Networks with Separable Basis Functions," *Neural Computation*, vol. 3, pp. 67–78, 1991.
- M. SCHETZEN: *The Volterra and Wiener Theories of Nonlinear Systems*, Malabar, Florida: Robert E. Krieger Publishing Company, 1989.
- G.A.F. SEBER & C.J. WILD: *Nonlinear Regression*, New York, New York: John Wiley & Sons, 1989.
- J. SIETSMA & R.J.F. DOW: "Neural Net Pruning - Why and How," in *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, California, vol. 1, July 24–27, 1988, pp. 325–333.
- F.M. SILVA & L.B. ALMEIDA: "Acceleration Techniques of the Backpropagation Algorithm," in L.B. Almeida & C.J. Wellekens (eds.) *Lecture Notes in Computer Science: Neural Networks, EURASIP Workshop 1990*, vol. 412, Berlin, Germany: Springer-Verlag, pp. 110–119, 1990.
- D.F. SPECHT: "A General Regression Neural Network," *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568–576, Nov. 1991.
- M.R. SPIEGEL: *Schaum's Outline Series: Mathematical Handbook of Formulas and Tables*, New York, New York: McGraw-Hill Book Company, 1968.
- K. STOKBRO, D.K. UMBERGER & J.A. HERTZ: "Exploiting Neurons with Localized Receptive Fields to Learn Chaos," *Complex Systems*, vol. 4, pp. 603–622, 1990.
- M. STONE: "Cross-validatory Choice and Assessment of Statistical Predictors," *Journal of the Royal Statistical Society B*, vol. 36, no. 2, pp. 111–147, 1974.
- J.AA. SØRENSEN: "A Family of Quantization based Piecewise-Linear Filter Networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1992*, San Francisco, California, March 23–26, 1992, vol. 2, pp. 329–331.

- F. TAKENS: "Detecting Strange Attractors in Fluid Turbulence," in D. Rand & L.S. Yong (eds.) *Dynamical Systems and Turbulence*, Berlin, Germany: Springer-Verlag, 1981.
- S. TAMURA & M. NAKAMURA: "Improvements to the Noise Reduction Neural Network," in *Proceeding of the IEEE International Conference on Acoustics, Speech and Signal Processing 1990*, Albuquerque, New Mexico, April 3–6, 1990, pp. 825–828.
- J.M.T. THOMSON & H.B. STEWART: *Nonlinear Dynamics and Chaos*, New York, New York: John Wiley & Sons, 1986.
- B. TOWNSHEND: "Nonlinear Prediction of Speech," in *Proceeding of the IEEE International Conference on Acoustics, Speech and Signal Processing 1991*, Toronto, Canada, May 14–17, 1991, pp. 425–428.
- H. TONG & K.S. LIM: "Threshold Autoregression, Limit Cycles and Cyclical Data," *Journal of the Royal Statistical Society B*, vol. 42, no. 3, pp. 245–292, 1980.
- G.T. TOUSSAINT: "Bibliography on Estimation of Misclassification," *IEEE Transactions on Information Theory*, vol. 20, no. 4, pp. 472–479, July 1974.
- A. TÖRN & A. ŽILINSKAS: "Global Optimization," in G. Goos & J. Hartmanis (eds.) *Lecture Notes in Computer Science*, vol. 350, Berlin, Germany: Springer-Verlag, 1988.
- B. DE VRIES, J.C. PRINCIPE & P.G. DE OLIVEIRA: "Adaline With Adaptive Recursive Memory," in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.) *Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop*, Piscataway, New Jersey: IEEE, 1991, pp. 101–110.
- A.S. WEIGEND & N.A. GERSHENFELD (eds.): *Time Series Prediction: Forecasting the Future and Understanding the Past*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1994.
- A.S. WEIGEND, B.A. HUBERMANN & D.E. RUMELHART: "Predicting the Future: A Connectionist Approach," *International Journal of Neural Systems*, vol. 1, no. 3, pp. 193–209, 1990.
- H. WHITE: "Consequences and Detection of Misspecified Nonlinear Regression Models," *Journal of the American Statistical Association*, vol. 76, no. 374, pp. 419–433, June 1981.
- H. WHITE: "Some Asymptotic Results for Back-Propagation," in *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, California, June 21–24, 1987, vol. 2, pp. 261–266.
- H. WHITE: "Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns," in *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, California, July 24–27, 1988, vol. 2, pp. 451–458.
- H. WHITE: "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, vol. 1, pp. 425–464, 1989a.
- H. WHITE: "An Additional Hidden Unit Test for Neglected Nonlinearity in Multilayer Feedforward Networks," in *Proceedings of the International Joint Conference on Neural Networks*, Washington D.C., June 18–22, 1989b, vol. 2, pp. 451–455.

- B. WIDROW & M.A. LEHR: “30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1441, Sept. 1990.
- B. WIDROW & M.E. HOFF, JR.: “Adaptive Switching Circuits,” *IRE WESCON Convention Record*, part 4, pp. 96–104, 1960.
- B. WIDROW, J.M. MCCOOL, M.G. LARIMORE & C.R. JOHNSON, JR.: “Stationary and Nonstationary Learning Characteristic of the LMS Adaptive Filter,” *Proceedings of the IEEE*, vol. 64, no. 8, Aug. 1976.
- B. WIDROW & S.D. STEARNS: *Adaptive Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1985.
- B. WIDROW & R.G. WINTER: “Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition,” *IEEE Computer*, pp. 25–39, March 1988.
- B. WIDROW, R.G. WINTER & R.A. BAXTER: “Layered Neural Nets for Pattern Recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 7, pp. 1109–1117, July 1988.
- P.H. WINSTON: *Artificial Intelligence*, Redwood City, California: Addison-Wesley Publishing Company, 1984.
- N.H. WULFF & J.A. HERTZ: “Prediction with Recurrent Networks,” in S.Y. Kung, F. Fallside, J. Aa. Sørensen & C.A. Kamm (eds.) *Neural Networks for Signal Processing 2: Proceedings of the 1992 IEEE-SP Workshop*, Piscataway, New Jersey: IEEE, 1992, pp. 464–473.
- K. YU: “Recursive Updating the Eigenvalue Decomposition of a Covariance Matrix,” *IEEE Transactions on Signal Processing*, vol. 39, no. 5, pp. 1136–1145, 1991.