

Recurrent Neural Networks for Prediction
Authored by Danilo P. Mandic, Jonathon A. Chambers
Copyright ©2001 John Wiley & Sons Ltd
ISBNs: 0-471-49517-4 (Hardback); 0-470-84535-X (Electronic)

RECURRENT
NEURAL
NETWORKS
FOR PREDICTION

WILEY SERIES IN ADAPTIVE AND LEARNING SYSTEMS FOR
SIGNAL PROCESSING, COMMUNICATIONS, AND CONTROL

Editor: Simon Haykin

Beckerman/ADAPTIVE COOPERATIVE SYSTEMS

Chen and Gu/CONTROL-ORIENTED SYSTEM IDENTIFICATION: An H
Approach

Cherkassky and Mulier/LEARNING FROM DATA: Concepts, Theory and Methods

Diamantaras and Kung/PRINCIPAL COMPONENT NEURAL NETWORKS:
Theory and Applications

Haykin and Puthusserypady/CHAOTIC DYNAMICS OF SEA CLUTTER

Haykin/NONLINEAR DYNAMICAL SYSTEMS: Feedforward Neural Network
Perspectives

Haykin/UNSUPERVISED ADAPTIVE FILTERING, VOLUME I: Blind Source
Separation

Haykin/UNSUPERVISED ADAPTIVE FILTERING, VOLUME II: Blind
Deconvolution

Hines/FUZZY AND NEURAL APPROACHES IN ENGINEERING

Hrycej/NEUROCONTROL: Towards an Industrial Control Methodology

Krstic, Kanellakopoulos, and Kokotovic/NONLINEAR AND ADAPTIVE
CONTROL DESIGN

Mann/INTELLIGENT IMAGE PROCESSING

Nikias and Shao/SIGNAL PROCESSING WITH ALPHA-STABLE
DISTRIBUTIONS AND APPLICATIONS

Passino and Burgess/STABILITY ANALYSIS OF DISCRETE EVENT SYSTEMS

Sanchez-Peña and Sznajder/ROBUST SYSTEMS THEORY AND APPLICATIONS

Tao and Kokotovic/ADAPTIVE CONTROL OF SYSTEMS WITH ACTUATOR
AND SENSOR NONLINEARITIES

Van Hulle/FAITHFUL REPRESENTATIONS AND TOPOGRAPHIC MAPS:
From Distortion- to Information-Based Self-Organization

Vapnik/STATISTICAL LEARNING THEORY

Werbos/THE ROOTS OF BACKPROPAGATION: From Ordered Derivatives to
Neural Networks and Political Forecasting

Yee and Haykin/REGULARIZED RADIAL-BASIS FUNCTION NETWORKS:
Theory and Applications

RECURRENT
NEURAL
NETWORKS
FOR PREDICTION

LEARNING ALGORITHMS,
ARCHITECTURES AND STABILITY

Danilo P. Mandic

*School of Information Systems,
University of East Anglia, UK*

Jonathon A. Chambers

*Department of Electronic and Electrical Engineering,
University of Bath, UK*

JOHN WILEY & SONS, LTD

Chichester • New York • Weinheim • Brisbane • Singapore • Toronto

Copyright ©2001 John Wiley & Sons, Ltd
Baffins Lane, Chichester,
West Sussex, PO19 1UD, England
National 01243 779777
International (+44) 1243 779777

e-mail (for orders and customer service enquiries): cs-books@wiley.co.uk

Visit our Home Page on <http://www.wiley.co.uk> or <http://www.wiley.com>

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London, W1P 0LP, UK, without the permission in writing of the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the publication.

Neither the author(s) nor John Wiley & Sons Ltd accept any responsibility or liability for loss or damage occasioned to any person or property through using the material, instructions, methods or ideas contained herein, or acting or refraining from acting as a result of such use. The author(s) and Publisher expressly disclaim all implied warranties, including merchantability of fitness for any particular purpose.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons is aware of a claim, the product names appear in initial capital or capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Other Wiley Editorial Offices

John Wiley & Sons, Inc., New York, USA

WILEY-VCH Verlag GmbH, Weinheim, Germany

John Wiley & Sons Australia, Ltd, Queensland

John Wiley & Sons (Canada) Ltd, Ontario

John Wiley & Sons (Asia) Pte Ltd, Singapore

Library of Congress Cataloging-in-Publication Data

Mandic, Danilo P.

Recurrent neural networks for prediction: learning algorithms, architectures, and stability / Danilo P. Mandic, Jonathon A. Chambers.

p. cm -- (Wiley series in adaptive and learning systems for signal processing, communications, and control)

Includes bibliographical references and index.

ISBN 0-471-49517-4 (alk. paper)

1. Machine learning. 2. Neural networks (Computer science) I. Chambers, Jonathon A. II. Title. III. Adaptive and learning systems for signal processing, communications, and control.

Q325.5 .M36 2001

006.3'2--dc21

2001033418

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0-471-49517-4

Produced from L^AT_EX files supplied by the author, typeset by T&T Productions Ltd, London. Printed and bound in Great Britain by Antony Rowe, Chippenham, Wiltshire.

This book is printed on acid-free paper responsibly manufactured from sustainable forestry, in which at least two trees are planted for each one used for paper production.

To our students and families

Contents

Preface	xiv
1 Introduction	1
1.1 Some Important Dates in the History of Connectionism	2
1.2 The Structure of Neural Networks	2
1.3 Perspective	4
1.4 Neural Networks for Prediction: Perspective	5
1.5 Structure of the Book	6
1.6 Readership	8
2 Fundamentals	9
2.1 Perspective	9
2.1.1 Chapter Summary	9
2.2 Adaptive Systems	9
2.2.1 Configurations of Adaptive Systems Used in Signal Processing	10
2.2.2 Blind Adaptive Techniques	12
2.3 Gradient-Based Learning Algorithms	12
2.4 A General Class of Learning Algorithms	14
2.4.1 Quasi-Newton Learning Algorithm	15
2.5 A Step-by-Step Derivation of the Least Mean Square (LMS) Algorithm	15
2.5.1 The Wiener Filter	16
2.5.2 Further Perspective on the Least Mean Square (LMS) Algorithm	17
2.6 On Gradient Descent for Nonlinear Structures	18
2.6.1 Extension to a General Neural Network	19
2.7 On Some Important Notions From Learning Theory	19
2.7.1 Relationship Between the Error and the Error Function	19
2.7.2 The Objective Function	20
2.7.3 Types of Learning with Respect to the Training Set and Objective Function	20
2.7.4 Deterministic, Stochastic and Adaptive Learning	21
2.7.5 Constructive Learning	21

2.7.6	Transformation of Input Data, Learning and Dimensionality	22
2.8	Learning Strategies	24
2.9	General Framework for the Training of Recurrent Networks by Gradient-Descent-Based Algorithms	24
2.9.1	Adaptive Versus Nonadaptive Training	24
2.9.2	Performance Criterion, Cost Function, Training Function	25
2.9.3	Recursive Versus Nonrecursive Algorithms	25
2.9.4	Iterative Versus Noniterative Algorithms	25
2.9.5	Supervised Versus Unsupervised Algorithms	25
2.9.6	Pattern Versus Batch Learning	26
2.10	Modularity Within Neural Networks	26
2.11	Summary	29
3	Network Architectures for Prediction	31
3.1	Perspective	31
3.2	Introduction	31
3.3	Overview	32
3.4	Prediction	33
3.5	Building Blocks	35
3.6	Linear Filters	37
3.7	Nonlinear Predictors	39
3.8	Feedforward Neural Networks: Memory Aspects	41
3.9	Recurrent Neural Networks: Local and Global Feedback	43
3.10	State-Space Representation and Canonical Form	44
3.11	Summary	45
4	Activation Functions Used in Neural Networks	47
4.1	Perspective	47
4.2	Introduction	47
4.3	Overview	51
4.4	Neural Networks and Universal Approximation	51
4.5	Other Activation Functions	54
4.6	Implementation Driven Choice of Activation Functions	57
4.7	MLP versus RBF Networks	60
4.8	Complex Activation Functions	60
4.9	Complex Valued Neural Networks as Modular Groups of Compositions of Möbius Transformations	65
4.9.1	Möbius Transformation	65
4.9.2	Activation Functions and Möbius Transformations	65
4.9.3	Existence and Uniqueness of Fixed Points in a Complex Neural Network via Theory of Modular Groups	67
4.10	Summary	68

5	Recurrent Neural Networks Architectures	69
5.1	Perspective	69
5.2	Introduction	69
5.3	Overview	72
5.4	Basic Modes of Modelling	72
5.4.1	Parametric versus Nonparametric Modelling	72
5.4.2	White, Grey and Black Box Modelling	73
5.5	NARMAX Models and Embedding Dimension	74
5.6	How Dynamically Rich are Nonlinear Neural Models?	75
5.6.1	Feedforward versus Recurrent Networks for Nonlinear Modelling	76
5.7	Wiener and Hammerstein Models and Dynamical Neural Networks	77
5.7.1	Overview of Block-Stochastic Models	77
5.7.2	Connection Between Block-Stochastic Models and Neural Networks	78
5.8	Recurrent Neural Network Architectures	81
5.9	Hybrid Neural Network Architectures	84
5.10	Nonlinear ARMA Models and Recurrent Networks	86
5.11	Summary	89
6	Neural Networks as Nonlinear Adaptive Filters	91
6.1	Perspective	91
6.2	Introduction	91
6.3	Overview	92
6.4	Neural Networks and Polynomial Filters	92
6.5	Neural Networks and Nonlinear Adaptive Filters	95
6.6	Training Algorithms for Recurrent Neural Networks	101
6.7	Learning Strategies for a Neural Predictor/Identifier	101
6.7.1	Learning Strategies for a Neural Adaptive Recursive Filter	103
6.7.2	Equation Error Formulation	104
6.7.3	Output Error Formulation	104
6.8	Filter Coefficient Adaptation for IIR Filters	105
6.8.1	Equation Error Coefficient Adaptation	107
6.9	Weight Adaptation for Recurrent Neural Networks	107
6.9.1	Teacher Forcing Learning for a Recurrent Perceptron	108
6.9.2	Training Process for a NARMA Neural Predictor	109
6.10	The Problem of Vanishing Gradients in Training of Recurrent Neural Networks	109
6.11	Learning Strategies in Different Engineering Communities	111
6.12	Learning Algorithms and the Bias/Variance Dilemma	111
6.13	Recursive and Iterative Gradient Estimation Techniques	113
6.14	Exploiting Redundancy in Neural Network Design	113
6.15	Summary	114

7	Stability Issues in RNN Architectures	115
7.1	Perspective	115
7.2	Introduction	115
7.3	Overview	118
7.4	A Fixed Point Interpretation of Convergence in Networks with a Sigmoid Nonlinearity	118
7.4.1	Some Properties of the Logistic Function	118
7.4.2	Logistic Function, Rate of Convergence and Fixed Point Theory	121
7.5	Convergence of Nonlinear Relaxation Equations Realised Through a Recurrent Perceptron	124
7.6	Relaxation in Nonlinear Systems Realised by an RNN	127
7.7	The Iterative Approach and Nesting	130
7.8	Upper Bounds for GAS Relaxation within FCRNNs	133
7.9	Summary	133
8	Data-Reusing Adaptive Learning Algorithms	135
8.1	Perspective	135
8.2	Introduction	135
8.2.1	Towards an <i>A Posteriori</i> Nonlinear Predictor	136
8.2.2	Note on the Computational Complexity	137
8.2.3	Chapter Summary	138
8.3	A Class of Simple <i>A Posteriori</i> Algorithms	138
8.3.1	The Case of a Recurrent Neural Filter	140
8.3.2	The Case of a General Recurrent Neural Network	141
8.3.3	Example for the Logistic Activation Function	141
8.4	An Iterated Data-Reusing Learning Algorithm	142
8.4.1	The Case of a Recurrent Predictor	143
8.5	Convergence of the <i>A Posteriori</i> Approach	143
8.6	<i>A Posteriori</i> Error Gradient Descent Algorithm	144
8.6.1	<i>A Posteriori</i> Error Gradient Algorithm for Recurrent Neural Networks	146
8.7	Experimental Results	146
8.8	Summary	147
9	A Class of Normalised Algorithms for Online Training of Recurrent Neural Networks	149
9.1	Perspective	149
9.2	Introduction	149
9.3	Overview	150
9.4	Derivation of the Normalised Adaptive Learning Rate for a Simple Feedforward Nonlinear Filter	151
9.5	A Normalised Algorithm for Online Adaptation of Recurrent Neural Networks	156
9.6	Summary	160

10	Convergence of Online Learning Algorithms in Neural Networks	161
10.1	Perspective	161
10.2	Introduction	161
10.3	Overview	164
10.4	Convergence Analysis of Online Gradient Descent Algorithms for Recurrent Neural Adaptive Filters	164
10.5	Mean-Squared and Steady-State Mean-Squared Error Convergence	167
10.5.1	Convergence in the Mean Square	168
10.5.2	Steady-State Mean-Squared Error	169
10.6	Summary	169
11	Some Practical Considerations of Predictability and Learning Algorithms for Various Signals	171
11.1	Perspective	171
11.2	Introduction	171
11.2.1	Detecting Nonlinearity in Signals	173
11.3	Overview	174
11.4	Measuring the Quality of Prediction and Detecting Nonlinearity within a Signal	174
11.4.1	Deterministic Versus Stochastic Plots	175
11.4.2	Variance Analysis of Delay Vectors	175
11.4.3	Dynamical Properties of NO ₂ Air Pollutant Time Series	176
11.5	Experiments on Heart Rate Variability	181
11.5.1	Experimental Results	181
11.6	Prediction of the Lorenz Chaotic Series	195
11.7	Bifurcations in Recurrent Neural Networks	197
11.8	Summary	198
12	Exploiting Inherent Relationships Between Parameters in Recurrent Neural Networks	199
12.1	Perspective	199
12.2	Introduction	199
12.3	Overview	204
12.4	Static and Dynamic Equivalence of Two Topologically Identical RNNs	205
12.4.1	Static Equivalence of Two Isomorphic RNNs	205
12.4.2	Dynamic Equivalence of Two Isomorphic RNNs	206
12.5	Extension to a General RTRL Trained RNN	208
12.6	Extension to Other Commonly Used Activation Functions	209
12.7	Extension to Other Commonly Used Learning Algorithms for Recurrent Neural Networks	209
12.7.1	Relationships Between β and η for the Backpropagation Through Time Algorithm	210
12.7.2	Results for the Recurrent Backpropagation Algorithm	211

12.7.3	Results for Algorithms with a Momentum Term	211
12.8	Simulation Results	212
12.9	Summary of Relationships Between β and η for General Recurrent Neural Networks	213
12.10	Relationship Between η and β for Modular Neural Networks: Perspective	214
12.11	Static Equivalence Between an Arbitrary and a Referent Modular Neural Network	214
12.12	Dynamic Equivalence Between an Arbitrary and a Referent Modular Network	215
12.12.1	Dynamic Equivalence for a GD Learning Algorithm	216
12.12.2	Dynamic Equivalence Between Modular Recurrent Neural Networks for the ERLS Learning Algorithm	217
12.12.3	Equivalence Between an Arbitrary and the Referent PRNN	218
12.13	Note on the β - η - \mathbf{W} Relationships and Contractivity	218
12.14	Summary	219
Appendix A The \mathcal{O} Notation and Vector and Matrix Differentiation		221
A.1	The \mathcal{O} Notation	221
A.2	Vector and Matrix Differentiation	221
Appendix B Concepts from the Approximation Theory		223
Appendix C Complex Sigmoid Activation Functions, Holomorphic Mappings and Modular Groups		227
C.1	Complex Sigmoid Activation Functions	227
C.1.1	Modular Groups	228
Appendix D Learning Algorithms for RNNs		231
D.1	The RTRL Algorithm	231
D.1.1	Teacher Forcing Modification of the RTRL Algorithm	234
D.2	Gradient Descent Learning Algorithm for the PRNN	234
D.3	The ERLS Algorithm	236
Appendix E Terminology Used in the Field of Neural Networks		239
Appendix F On the <i>A Posteriori</i> Approach in Science and Engineering		241
F.1	History of <i>A Posteriori</i> Techniques	241
F.2	The Usage of <i>A Posteriori</i>	242
F.2.1	<i>A Posteriori</i> Techniques in the RNN Framework	242

CONTENTS	xiii
F.2.2 The Geometric Interpretation of <i>A Posteriori</i> Error Learning	243
Appendix G Contraction Mapping Theorems	245
G.1 Fixed Points and Contraction Mapping Theorems	245
G.1.1 Contraction Mapping Theorem in \mathbb{R}	245
G.1.2 Contraction Mapping Theorem in \mathbb{R}^N	246
G.2 Lipschitz Continuity and Contraction Mapping	246
G.3 Historical Perspective	247
Appendix H Linear GAS Relaxation	251
H.1 Relaxation in Linear Systems	251
H.1.1 Stability Result for $\sum_{i=1}^m a_i = 1$	253
H.2 Examples	253
Appendix I The Main Notions in Stability Theory	263
Appendix J Deseasonalising Time Series	265
References	267
Index	281

Preface

New technologies in engineering, physics and biomedicine are creating problems in which nonstationarity, nonlinearity, uncertainty and complexity play a major role. Solutions to many of these problems require the use of nonlinear processors, among which neural networks are one of the most powerful. Neural networks are appealing because they learn by example and are strongly supported by statistical and optimisation theories. They not only complement conventional signal processing techniques, but also emerge as a convenient alternative to expand signal processing horizons.

The use of recurrent neural networks as identifiers and predictors in nonlinear dynamical systems has increased significantly. They can exhibit a wide range of dynamics, due to feedback, and are also tractable nonlinear maps.

In our work, neural network models are considered as massively interconnected nonlinear adaptive filters. The emphasis is on dynamics, stability and spatio-temporal behaviour of recurrent architectures and algorithms for prediction. However, wherever possible the material has been presented starting from feedforward networks and building up to the recurrent case.

Our objective is to offer an accessible self-contained research monograph which can also be used as a graduate text. The material presented in the book is of interest to a wide population of researchers working in engineering, computing, science, finance and biosciences. So that the topics are self-contained, we assume familiarity with the basic concepts of analysis and linear algebra. The material presented in Chapters 1–6 can serve as an advanced text for courses on neural adaptive systems. The book encompasses traditional and advanced learning algorithms and architectures for recurrent neural networks. Although we emphasise the problem of time series prediction, the results are applicable to a wide range of problems, including other signal processing configurations such as system identification, noise cancellation and inverse system modelling. We harmonise the concepts of learning algorithms, embedded systems, representation of memory, neural network architectures and causal–noncausal dealing with time. A special emphasis is given to stability of algorithms – a key issue in real-time applications of adaptive systems.

This book has emerged from the research that D. Mandic has undertaken while at Imperial College of Science, Technology and Medicine, London, UK. The work was continued within the vibrant culture of the University of East Anglia, Norwich, UK.

Acknowledgements

Danilo Mandic acknowledges Dr M. Razaz for providing a home from home in the Bioinformatics Laboratory, School of Information Systems, University of East Anglia. Many thanks to the people from the lab for creating a congenial atmosphere at work. The Dean of the School of Information Systems, Professor V. Rayward-Smith and his predecessor Dr J. Glauert, deserve thanks for their encouragement and support. Dr M. Bozic has done a tremendous job on proofreading the mathematics. Dr W. Sherliker has contributed greatly to Chapter 10. Dr D. I. Kim has proofread the mathematically involved chapters. I thank Dr G. Cawley, Dr M. Dzamonja, Dr A. James and Dr G. Smith for proofreading the manuscript in its various phases. Dr R. Harvey has been of great help throughout. Special thanks to my research associates I. Krcmar and Dr R. Foxall for their help with some of the experimental results. H. Graham has always been at hand with regard to computing problems. Many of the results presented here have been achieved while I was at Imperial College, where I greatly benefited from the unique research atmosphere in the Signal Processing Section of the Department of Electrical and Electronic Engineering.

Jonathon Chambers acknowledges the outstanding PhD researchers with whom he has had the opportunity to interact, they have helped so much towards his orientation in adaptive signal processing. He also acknowledges Professor P. Watson, Head of the Department of Electronic and Electrical Engineering, University of Bath, who has provided the opportunity to work on the book during its later stages. Finally, he thanks Mr D. M. Brookes and Dr P. A. Naylor, his former colleagues, for their collaboration in research projects.

*Danilo Mandic
Jonathon Chambers*

List of Abbreviations

ACF	Autocorrelation function
AIC	Akaike Information Criterion
ANN	Artificial Neural Network
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
ART	Adaptive Resonance Theory
AS	Asymptotic Stability
ATM	Asynchronous Transfer Mode
BIC	Bayesian Information Criterion
BC	Before Christ
BIBO	Bounded Input Bounded Output
BP	Backpropagation
BPTT	Backpropagation Through Time
CM	Contraction Mapping
CMT	Contraction Mapping Theorem
CNN	Cellular Neural Network
DC	Direct Current
DR	Data Reusing
DSP	Digital Signal Processing
DVS	Deterministic Versus Stochastic
ECG	Electrocardiogram
EKF	Extended Kalman Filter
ERLS	Extended Recursive Least Squares
ES	Exponential Stability
FCRNN	Fully Connected Recurrent Neural Network
FFNN	Feedforward Neural Network
FIR	Finite Impulse Response
FPI	Fixed Point Iteration
GAS	Global Asymptotic Stability
GD	Gradient Descent
HOS	Higher-Order Statistics
HRV	Heart Rate Variability
i.i.d.	Independent Identically Distributed
IIR	Infinite Impulse Response
IVT	Intermediate Value Theorem
KF	Kalman Filter

LMS	Least Mean Square
LPC	Linear Predictive Coding
LRGF	Locally Recurrent Globally Feedforward
LUT	Look-Up Table
MA	Moving Average
MLP	Multi-Layer Perceptron
MMSE	Minimum Mean Square Error
MPEG	Moving Pictures Experts Group
MSLC	Multiple Sidelobe Cancellation
MSE	Mean Squared Error
MVT	Mean Value Theorem
NARMA	Nonlinear Autoregressive Moving Average
NARMAX	Nonlinear Autoregressive Moving Average with eXogenous input
NARX	Nonlinear Autoregressive with eXogenous input
NGD	Nonlinear Gradient Descent
NNGD	Normalised Nonlinear Gradient Descent
NLMS	Normalised Least Mean Square
NMA	Nonlinear Moving Average
NN	Neural Network
NO ₂	Nitrogen dioxide
NRTRL	Normalised RTRL algorithm
pdf	probability density function
PG	Prediction Gain
PRNN	Pipelined Recurrent Neural Network
PSD	Power Spectral Density
RAM	Random Access Memory
RBF	Radial Basis Function
RBP	Recurrent Backpropagation
RLS	Recursive Least Squares
RNN	Recurrent Neural Network
ROM	Read-Only Memory
R-R	Distance between two consecutive R waves in ECG
RTRL	Real-Time Recurrent Learning
SG	Stochastic Gradient
SP	Signal Processing
VLSI	Very Large Scale Integration
WGN	White Gaussian Noise
WWW	World Wide Web

Mathematical Notation

z'	First derivative of variable z
$\{\cdot\}$	Set of elements
α	Momentum constant
β	Slope of the nonlinear activation function Φ
γ	Contraction constant, gain of the activation function
Γ	Modular group of compositions of Möbius transformations
$\Gamma(k)$	Adaptation gain vector
δ_i	Gradient at i th neuron
δ_{ij}	Kronecker delta function
$\epsilon(k)$	Additive noise
η	Learning rate
$\nabla_x Y$	Gradient of Y with respect to x
$ \cdot $	Modulus operator
$\ \cdot\ _p$	Vector or matrix p -norm operator
$*$	Convolution operator
\propto	Proportional to
$\mathbf{0}$	Null vector
λ	Forgetting factor
λ_i	Eigenvalues of a matrix
μ	Step size in the LMS algorithm
μ	Mean of a random variable
$\nu(k)$	Additive noise
Ω_k	Set of nearest neighbours vectors
$ \Omega_k $	Number of elements in set Ω_k
$\pi_{n,l}^j$	Sensitivity of the j th neuron to the change in $w_{n,l}$
Φ	Nonlinear activation function of a neuron
Φ'_x	First derivative of Φ with respect to x
$\mathbf{\Pi}$	Matrix of gradients at the output neuron of an RNN
∂	Partial derivative operator
\sum	Summation operator
σ	Variance
$\sigma(x)$	General sigmoid function
$\Sigma(k)$	Sample input matrix
τ	Delay operator
Θ	General nonlinear function
$\Theta(k)$	Parameter vector

$(\cdot)^T$	Vector or matrix transpose
$\mathcal{C}_L(\cdot)$	Computational load
B	Backshift operator
\mathbb{C}	Set of complex numbers
$C^n(a, b)$	The class of n -times continuously differentiable functions on an open interval (a, b)
$d(k)$	Desired response
$\deg(\cdot)$	Degree of a polynomial
$\text{diag}[\cdot]$	Diagonal elements of a matrix
$E(k)$	Cost function
$E[\cdot]$	Expectation operator
$E[y x]$	Conditional expectation operator
$e(k)$	Instantaneous prediction error
$\bar{e}(k)$	<i>A posteriori</i> prediction error
$F(\cdot, \cdot)$	Nonlinear approximation function
$G(\cdot)$	Basis function
\mathbf{I}	Identity matrix
\inf	Infimum
$J(k)$	Cost function
\mathcal{J}	Non-negative error measure
\mathbf{J}	Jacobian
k	Discrete time index
\mathbf{H}	Hessian matrix
$H(z)$	Transfer function in z -domain
H^∞	The infinity norm quadratic optimisation
L	Lipschitz constant
\mathcal{L}_p	\mathcal{L}_p norm
\lim	Limit
\max	Maximum
\min	Minimum
\mathbb{N}	Set of natural numbers
$\mathcal{N}(\mu, \sigma^2)$	Normal random process with mean μ and variance σ^2
$\mathcal{O}(\cdot)$	Order of computational complexity
$P(f)$	Power spectral density
$q(k)$	Measurement noise
\mathbb{R}	Set of real numbers
\mathbb{R}^+	$\{x \in \mathbb{R} \mid x > 0\}$
\mathbb{R}^n	The n -dimensional Euclidean space
R_p	Prediction gain
$\mathbf{R}_{x,y}, \mathbf{R}_{x,y}$	Correlation matrix between vectors \mathbf{x} and \mathbf{y}
S	Class of sigmoidal functions
$\text{sgn}(\cdot)$	Signum function
$\text{span}(\cdot)$	Span of a vector
\sup	Supremum
\mathbf{T}	Contraction operator
$\text{tr}\{\cdot\}$	Trace operator (sum of diagonal elements of a matrix)

$\mathbf{u}(k)$	Input vector to an RNN
v	Internal activation potential of a neuron
$v(k)$	System noise
$\mathbf{w}(k)$	Weight vector
$\Delta\mathbf{w}(k)$	Correction to the weight vector
$\tilde{\mathbf{w}}(k)$	Optimal weight vector
$\check{\mathbf{w}}(k)$	Weight error vector
$\hat{\mathbf{w}}(k)$	Weight vector estimate
$w_{i,j}$	Weight connecting neuron j to neuron i
$\mathbf{W}(k)$	Weight matrix of an NN
$\Delta\mathbf{W}(k)$	Correction to the weight matrix
$\mathbf{x}(k), \mathbf{X}(k)$	External input vector to an NN
$x_l(k) \in \mathbf{X}(k)$	The l th element of vector $\mathbf{X}(k)$
\hat{x}	Estimated value of x
x^*	Fixed point of the sequence $\{x\}$
y	Output of an NN
$y_{i,j}$	Output of the j th neuron of the i th module of the PRNN
z^{-k}	The k th-order time delay
\mathcal{Z}	The \mathcal{Z} transform
\mathcal{Z}^{-1}	The inverse \mathcal{Z} transform

1

Introduction

Artificial neural network (ANN) models have been extensively studied with the aim of achieving human-like performance, especially in the field of pattern recognition. These networks are composed of a number of nonlinear computational elements which operate in parallel and are arranged in a manner reminiscent of biological neural interconnections. ANNs are known by many names such as connectionist models, parallel distributed processing models and neuromorphic systems (Lippmann 1987). The origin of connectionist ideas can be traced back to the Greek philosopher, Aristotle, and his ideas of mental associations. He proposed some of the basic concepts such as that memory is composed of simple elements connected to each other via a number of different mechanisms (Medler 1998).

While early work in ANNs used anthropomorphic arguments to introduce the methods and models used, today neural networks used in engineering are related to algorithms and computation and do not question how brains might work (Hunt *et al.* 1992). For instance, recurrent neural networks have been attractive to physicists due to their isomorphism to spin glass systems (Ermentrout 1998). The following properties of neural networks make them important in signal processing (Hunt *et al.* 1992): they are nonlinear systems; they enable parallel distributed processing; they can be implemented in VLSI technology; they provide learning, adaptation and data fusion of both qualitative (symbolic data from artificial intelligence) and quantitative (from engineering) data; they realise multivariable systems.

The area of neural networks is nowadays considered from two main perspectives. The first perspective is cognitive science, which is an interdisciplinary study of the mind. The second perspective is connectionism, which is a theory of information processing (Medler 1998). The neural networks in this work are approached from an engineering perspective, i.e. to make networks efficient in terms of topology, learning algorithms, ability to approximate functions and capture dynamics of time-varying systems. From the perspective of connection patterns, neural networks can be grouped into two categories: feedforward networks, in which graphs have no loops, and recurrent networks, where loops occur because of feedback connections. Feedforward networks are static, that is, a given input can produce only one set of outputs, and hence carry no memory. In contrast, recurrent network architectures enable the information to be temporally memorised in the networks (Kung and Hwang 1998). Based on training by example, with strong support of statistical and optimisation theories

(Cichocki and Unbehauen 1993; Zhang and Constantinides 1992), neural networks are becoming one of the most powerful and appealing nonlinear signal processors for a variety of signal processing applications. As such, neural networks expand signal processing horizons (Chen 1997; Haykin 1996b), and can be considered as massively interconnected nonlinear adaptive filters. Our emphasis will be on dynamics of recurrent architectures and algorithms for prediction.

1.1 Some Important Dates in the History of Connectionism

In the early 1940s the pioneers of the field, McCulloch and Pitts, studied the potential of the interconnection of a model of a neuron. They proposed a computational model based on a simple neuron-like element (McCulloch and Pitts 1943). Others, like Hebb were concerned with the adaptation laws involved in neural systems. In 1949 Donald Hebb devised a learning rule for adapting the connections within artificial neurons (Hebb 1949). A period of early activity extends up to the 1960s with the work of Rosenblatt (1962) and Widrow and Hoff (1960). In 1958, Rosenblatt coined the name ‘perceptron’. Based upon the perceptron (Rosenblatt 1958), he developed the theory of statistical separability. The next major development is the new formulation of learning rules by Widrow and Hoff in their Adaline (Widrow and Hoff 1960). In 1969, Minsky and Papert (1969) provided a rigorous analysis of the perceptron. The work of Grossberg in 1976 was based on biological and psychological evidence. He proposed several new architectures of nonlinear dynamical systems (Grossberg 1974) and introduced adaptive resonance theory (ART), which is a real-time ANN that performs supervised and unsupervised learning of categories, pattern classification and prediction. In 1982 Hopfield pointed out that neural networks with certain symmetries are analogues to spin glasses.

A seminal book on ANNs is by Rumelhart *et al.* (1986). Fukushima explored competitive learning in his biologically inspired Cognitron and Neocognitron (Fukushima 1975; Widrow and Lehr 1990). In 1971 Werbos developed a backpropagation learning algorithm which he published in his doctoral thesis (Werbos 1974). Rumelhart *et al.* rediscovered this technique in 1986 (Rumelhart *et al.* 1986). Kohonen (1982), introduced self-organised maps for pattern recognition (Burr 1993).

1.2 The Structure of Neural Networks

In neural networks, computational models or nodes are connected through weights that are adapted during use to improve performance. The main idea is to achieve good performance via dense interconnection of simple computational elements. The simplest node provides a linear combination of N weights w_1, \dots, w_N and N inputs x_1, \dots, x_N , and passes the result through a nonlinearity Φ , as shown in Figure 1.1.

Models of neural networks are specified by the net topology, node characteristics and training or learning rules. From the perspective of connection patterns, neural networks can be grouped into two categories: feedforward networks, in which graphs have no loops, and recurrent networks, where loops occur because of feedback connections. Neural networks are specified by (Tsoi and Back 1997)

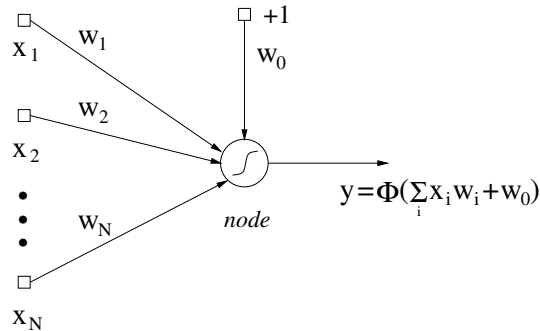


Figure 1.1 Connections within a node

- Node: typically a sigmoid function;
- Layer: a set of nodes at the same hierarchical level;
- Connection: constant weights or weights as a linear dynamical system, feedforward or recurrent;
- Architecture: an arrangement of interconnected neurons;
- Mode of operation: analogue or digital.

Massively interconnected neural nets provide a greater degree of robustness or fault tolerance than sequential machines. By robustness we mean that small perturbations in parameters will also result in small deviations of the values of the signals from their nominal values.

In our work, hence, the term *neuron* will refer to an operator which performs the mapping

$$\text{Neuron: } \mathbb{R}^{N+1} \rightarrow \mathbb{R} \quad (1.1)$$

as shown in Figure 1.1. The equation

$$y = \Phi \left(\sum_{i=1}^N w_i x_i + w_0 \right) \quad (1.2)$$

represents a mathematical description of a neuron. The input vector is given by $\mathbf{x} = [x_1, \dots, x_N, 1]^T$, whereas $\mathbf{w} = [w_1, \dots, w_N, w_0]^T$ is referred to as the weight vector of a neuron. The weight w_0 is the weight which corresponds to the bias input, which is typically set to unity. The function $\Phi : \mathbb{R} \rightarrow (0, 1)$ is monotone and continuous, most commonly of a sigmoid shape. A set of interconnected neurons is a neural network (NN). If there are N input elements to an NN and M output elements of an NN, then an NN defines a continuous mapping

$$\text{NN: } \mathbb{R}^N \rightarrow \mathbb{R}^M. \quad (1.3)$$

1.3 Perspective

Before the 1920s, prediction was undertaken by simply extrapolating the time series through a global fit procedure. The beginning of modern time series prediction was in 1927 when Yule introduced the autoregressive model in order to predict the annual number of sunspots. For the next half century the models considered were linear, typically driven by white noise. In the 1980s, the state-space representation and machine learning, typically by neural networks, emerged as new potential models for prediction of highly complex, nonlinear and nonstationary phenomena. This was the shift from rule-based models to data-driven methods (Gershenfeld and Weigend 1993).

Time series prediction has traditionally been performed by the use of linear parametric autoregressive (AR), moving-average (MA) or autoregressive moving-average (ARMA) models (Box and Jenkins 1976; Ljung and Soderstrom 1983; Makhoul 1975), the parameters of which are estimated either in a block or a sequential manner with the least mean square (LMS) or recursive least-squares (RLS) algorithms (Haykin 1994). An obvious problem is that these processors are linear and are not able to cope with certain nonstationary signals, and signals whose mathematical model is not linear. On the other hand, neural networks are powerful when applied to problems whose solutions require knowledge which is difficult to specify, but for which there is an abundance of examples (Dillon and Manikopoulos 1991; Gent and Shepard 1992; Townshend 1991). As time series prediction is conventionally performed entirely by inference of future behaviour from examples of past behaviour, it is a suitable application for a neural network predictor. The neural network approach to time series prediction is non-parametric in the sense that it does not need to know any information regarding the process that generates the signal. For instance, the order and parameters of an AR or ARMA process are not needed in order to carry out the prediction. This task is carried out by a process of learning from examples presented to the network and changing network weights in response to the output error.

Li (1992) has shown that the recurrent neural network (RNN) with a sufficiently large number of neurons is a realisation of the nonlinear ARMA (NARMA) process. RNNs performing NARMA prediction have traditionally been trained by the real-time recurrent learning (RTRL) algorithm (Williams and Zipser 1989a) which provides the training process of the RNN ‘on the run’. However, for a complex physical process, some difficulties encountered by RNNs such as the high degree of approximation involved in the RTRL algorithm for a high-order MA part of the underlying NARMA process, high computational complexity of $\mathcal{O}(N^4)$, with N being the number of neurons in the RNN, insufficient degree of nonlinearity involved, and relatively low robustness, induced a search for some other, more suitable schemes for RNN-based predictors.

In addition, in time series prediction of nonlinear and nonstationary signals, there is a need to learn long-time temporal dependencies. This is rather difficult with conventional RNNs because of the problem of vanishing gradient (Bengio *et al.* 1994). A solution to that problem might be NARMA models and nonlinear autoregressive moving average models with exogenous inputs (NARMAX) (Siegelmann *et al.* 1997) realised by recurrent neural networks. However, the quality of performance is highly dependent on the order of the AR and MA parts in the NARMAX model.

The main reasons for using neural networks for prediction rather than classical time series analysis are (Wu 1995)

- they are computationally at least as fast, if not faster, than most available statistical techniques;
- they are self-monitoring (i.e. they learn how to make accurate predictions);
- they are as accurate if not more accurate than most of the available statistical techniques;
- they provide iterative forecasts;
- they are able to cope with nonlinearity and nonstationarity of input processes;
- they offer both parametric and nonparametric prediction.

1.4 Neural Networks for Prediction: Perspective

Many signals are generated from an inherently nonlinear physical mechanism and have statistically non-stationary properties, a classic example of which is speech. Linear structure adaptive filters are suitable for the nonstationary characteristics of such signals, but they do not account for nonlinearity and associated higher-order statistics (Shynk 1989). Adaptive techniques which recognise the nonlinear nature of the signal should therefore outperform traditional linear adaptive filtering techniques (Haykin 1996a; Kay 1993). The classic approach to time series prediction is to undertake an analysis of the time series data, which includes modelling, identification of the model and model parameter estimation phases (Makhoul 1975). The design may be iterated by measuring the closeness of the model to the real data. This can be a long process, often involving the derivation, implementation and refinement of a number of models before one with appropriate characteristics is found.

In particular, the most difficult systems to predict are

- those with non-stationary dynamics, where the underlying behaviour varies with time, a typical example of which is speech production;
- those which deal with physical data which are subject to noise and experimentation error, such as biomedical signals;
- those which deal with short time series, providing few data points on which to conduct the analysis, such as heart rate signals, chaotic signals and meteorological signals.

In all these situations, traditional techniques are severely limited and alternative techniques must be found (Bengio 1995; Haykin and Li 1995; Li and Haykin 1993; Niranjana and Kadiramanathan 1991).

On the other hand, neural networks are powerful when applied to problems whose solutions require knowledge which is difficult to specify, but for which there is an abundance of examples (Dillon and Manikopoulos 1991; Gent and Sheppard 1992; Townshend 1991). From a system theoretic point of view, neural networks can be considered as a conveniently parametrised class of nonlinear maps (Narendra 1996).

There has been a recent resurgence in the field of ANNs caused by new net topologies, VLSI computational algorithms and the introduction of massive parallelism into neural networks. As such, they are both universal function approximators (Cybenko 1989; Hornik *et al.* 1989) and arbitrary pattern classifiers. From the Weierstrass Theorem, it is known that polynomials, and many other approximation schemes, can approximate arbitrarily well a continuous function. Kolmogorov's theorem (a negative solution of Hilbert's 13th problem (Lorentz 1976)) states that any continuous function can be approximated using only linear summations and nonlinear but continuously increasing functions of only one variable. This makes neural networks suitable for universal approximation, and hence prediction. Although sometimes computationally demanding (Williams and Zipser 1995), neural networks have found their place in the area of nonlinear autoregressive moving average (NARMA) (Bailer-Jones *et al.* 1998; Connor *et al.* 1992; Lin *et al.* 1996) prediction applications. Comprehensive survey papers on the use and role of ANNs can be found in Widrow and Lehr (1990), Lippmann (1987), Medler (1998), Ermentrout (1998), Hunt *et al.* (1992) and Billings (1980).

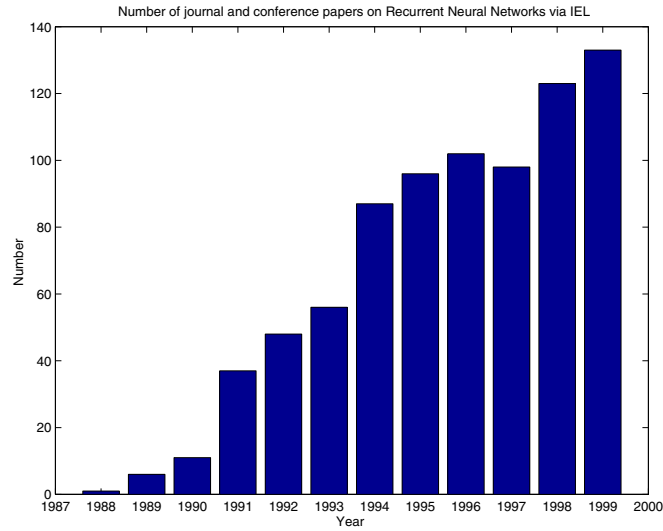
Only recently, neural networks have been considered for prediction. A recent competition by the Santa Fe Institute for Studies in the Science of Complexity (1991–1993) (Weigend and Gershenfeld 1994) showed that neural networks can outperform conventional linear predictors in a number of applications (Waibel *et al.* 1989). In journals, there has been an ever increasing interest in applying neural networks. A most comprehensive issue on recurrent neural networks is the issue of the *IEEE Transactions of Neural Networks*, vol. 5, no. 2, March 1994. In the signal processing community, there has been a recent special issue 'Neural Networks for Signal Processing' of the *IEEE Transactions on Signal Processing*, vol. 45, no. 11, November 1997, and also the issue 'Intelligent Signal Processing' of the *Proceedings of IEEE*, vol. 86, no. 11, November 1998, both dedicated to the use of neural networks in signal processing applications.

Figure 1.2 shows the frequency of the appearance of articles on recurrent neural networks in common citation index databases. Figure 1.2(a) shows number of journal and conference articles on recurrent neural networks in IEE/IEEE publications between 1988 and 1999. The data were gathered using the IEL Online service, and these publications are mainly periodicals and conferences in electronics engineering. Figure 1.2(b) shows the frequency of appearance for BIDS/ATHENS database, between 1988 and 2000,¹ which also includes non-engineering publications. From Figure 1.2, there is a clear growing trend in the frequency of appearance of articles on recurrent neural networks. Therefore, we felt that there was a need for a research monograph that would cover a part of the area with up to date ideas and results.

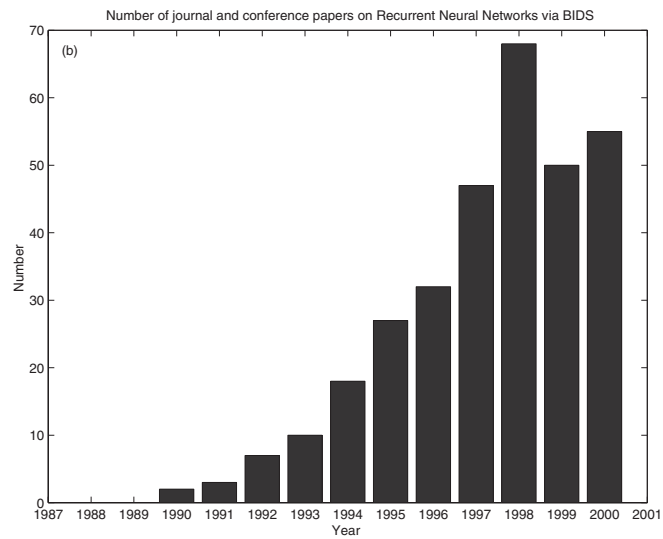
1.5 Structure of the Book

The book is divided into 12 chapters and 10 appendices. An introduction to connectionism and the notion of neural networks for prediction is included in Chapter 1. The fundamentals of adaptive signal processing and learning theory are detailed in Chapter 2. An initial overview of network architectures for prediction is given in Chapter 3.

¹ At the time of writing, only the months up to September 2000 were covered.



(a) Appearance of articles on Recurrent Neural Networks in IEE/IEEE publications in period 1988–1999



(b) Appearance of articles on Recurrent Neural Networks in BIDS database in period 1988–2000

Figure 1.2 Appearance of articles on RNNs in major citation databases. (a) Appearance of articles on recurrent neural networks in IEE/IEEE publications in period 1988–1999. (b) Appearance of articles on recurrent neural networks in BIDS database in period 1988–2000.

Chapter 4 contains a detailed discussion of activation functions and new insights are provided by the consideration of neural networks within the framework of modular groups from number theory. The material in Chapter 5 builds upon that within Chapter 3 and provides more comprehensive coverage of recurrent neural network architectures together with concepts from nonlinear system modelling. In Chapter 6, neural networks are considered as nonlinear adaptive filters whereby the necessary learning strategies for recurrent neural networks are developed. The stability issues for certain recurrent neural network architectures are considered in Chapter 7 through the exploitation of fixed point theory and bounds for global asymptotic stability are derived. *A posteriori* adaptive learning algorithms are introduced in Chapter 8 and the synergy with data-reusing algorithms is highlighted. In Chapter 9, a new class of normalised algorithms for online training of recurrent neural networks is derived. The convergence of online learning algorithms for neural networks is addressed in Chapter 10. Experimental results for the prediction of nonlinear and nonstationary signals with recurrent neural networks are presented in Chapter 11. In Chapter 12, the exploitation of inherent relationships between parameters within recurrent neural networks is described. Appendices A to J provide background to the main chapters and cover key concepts from linear algebra, approximation theory, complex sigmoid activation functions, a precedent learning algorithm for recurrent neural networks, terminology in neural networks, *a posteriori* techniques in science and engineering, contraction mapping theory, linear relaxation and stability, stability of general nonlinear systems and deseasonalising of time series. The book concludes with a comprehensive bibliography.

1.6 Readership

This book is targeted at graduate students and research engineers active in the areas of communications, neural networks, nonlinear control, signal processing and time series analysis. It will also be useful for engineers and scientists working in diverse application areas, such as artificial intelligence, biomedicine, earth sciences, finance and physics.

2

Fundamentals

2.1 Perspective

Adaptive systems are at the very core of modern digital signal processing. There are many reasons for this, foremost amongst these is that adaptive filtering, prediction or identification do not require explicit *a priori* statistical knowledge of the input data. Adaptive systems are employed in numerous areas such as biomedicine, communications, control, radar, sonar and video processing (Haykin 1996a).

2.1.1 Chapter Summary

In this chapter the fundamentals of adaptive systems are introduced. Emphasis is first placed upon the various structures available for adaptive signal processing, and includes the predictor structure which is the focus of this book. Basic learning algorithms and concepts are next detailed in the context of linear and nonlinear structure filters and networks. Finally, the issue of modularity is discussed.

2.2 Adaptive Systems

Adaptability, in essence, is the ability to react in sympathy with disturbances to the environment. A system that exhibits adaptability is said to be adaptive. Biological systems are adaptive systems; animals, for example, can adapt to changes in their environment through a learning process (Haykin 1999a).

A generic adaptive system employed in engineering is shown in Figure 2.1. It consists of

- a set of adjustable parameters (weights) within some filter structure;
- an error calculation block (the difference between the desired response and the output of the filter structure);
- a control (learning) algorithm for the adaptation of the weights.

The type of learning represented in Figure 2.1 is so-called supervised learning, since the learning is directed by the desired response of the system. Here, the goal

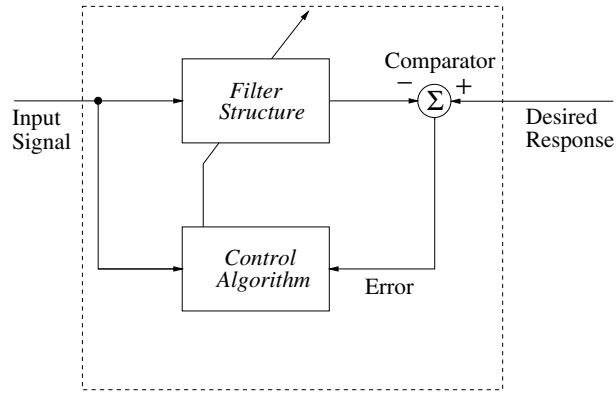


Figure 2.1 Block diagram of an adaptive system

is to adjust iteratively the free parameters (weights) of the adaptive system so as to minimise a prescribed cost function in some predetermined sense.¹ The filter structure within the adaptive system may be linear, such as a finite impulse response (FIR) or infinite impulse response (IIR) filter, or nonlinear, such as a Volterra filter or a neural network.

2.2.1 Configurations of Adaptive Systems Used in Signal Processing

Four typical configurations of adaptive systems used in engineering are shown in Figure 2.2 (Jenkins *et al.* 1996). The notions of an adaptive filter and adaptive system are used here interchangeably.

For the system identification configuration shown in Figure 2.2(a), both the adaptive filter and the unknown system are fed with the same input signal $x(k)$. The error signal $e(k)$ is formed at the output as $e(k) = d(k) - y(k)$, and the parameters of the adaptive system are adjusted using this error information. An attractive point of this configuration is that the desired response signal $d(k)$, also known as a teaching or training signal, is readily available from the unknown system (plant). Applications of this scheme are in acoustic and electrical echo cancellation, control and regulation of real-time industrial and other processes (plants). The knowledge about the system is stored in the set of converged weights of the adaptive system. If the dynamics of the plant are not time-varying, it is possible to identify the parameters (weights) of the plant to an arbitrary accuracy.

If we desire to form a system which inter-relates noise components in the input and desired response signals, the noise cancelling configuration can be implemented (Figure 2.2(b)). The only requirement is that the noise in the primary input and the reference noise are correlated. This configuration subtracts an estimate of the noise from the received signal. Applications of this configuration include noise cancellation

¹ The aim is to minimise some function of the error e . If $E[e^2]$ is minimised, we consider minimum mean squared error (MSE) adaptation, the statistical expectation operator, $E[\cdot]$, is due to the random nature of the inputs to the adaptive system.

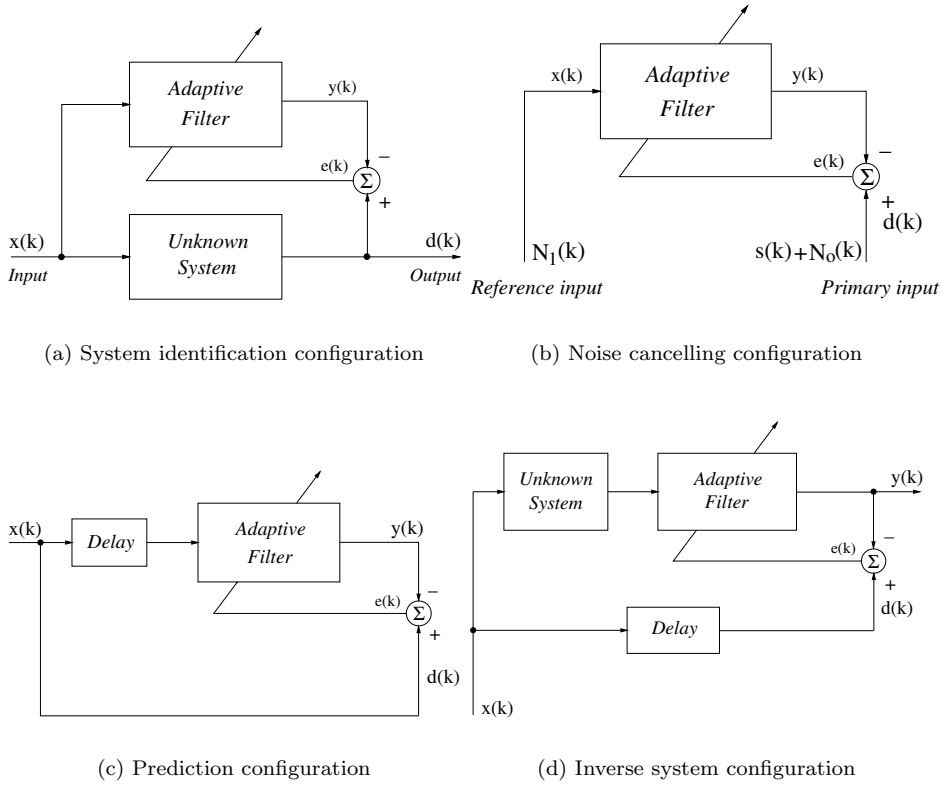


Figure 2.2 Configurations for applications of adaptive systems

in acoustic environments and estimation of the foetal ECG from the mixture of the maternal and foetal ECG (Widrow and Stearns 1985).

In the adaptive prediction configuration, the desired signal is the input signal advanced relative to the input of the adaptive filter, as shown in Figure 2.2(c). This configuration has numerous applications in various areas of engineering, science and technology and most of the material in this book is dedicated to prediction. In fact, prediction may be considered as a basis for any adaptation process, since the adaptive filter is trying to predict the desired response.

The inverse system configuration, shown in Figure 2.2(d), has an adaptive system cascaded with the unknown system. A typical application is adaptive channel equalisation in telecommunications, whereby an adaptive system tries to compensate for the possibly time-varying communication channel, so that the transfer function from the input to the output of Figure 2.2(d) approximates a pure delay.

In most adaptive signal processing applications, parametric methods are applied which require *a priori* knowledge (or postulation) of a specific model in the form of differential or difference equations. Thus, it is necessary to determine the appropriate model order for successful operation, which will underpin data length requirements. On the other hand, nonparametric methods employ general model forms of integral

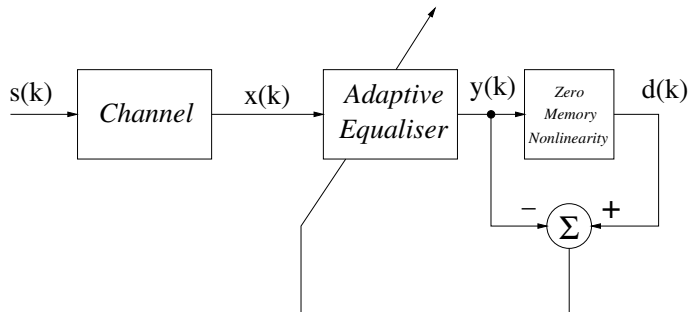


Figure 2.3 Block diagram of a blind equalisation structure

equations or functional expansions valid for a broad class of dynamic nonlinearities. The most widely used nonparametric methods are referred to as the Volterra–Wiener approach and are based on functional expansions.

2.2.2 Blind Adaptive Techniques

The presence of an explicit desired response signal, $d(k)$, in all the structures shown in Figure 2.2 implies that conventional, supervised, adaptive signal processing techniques may be applied for the purpose of learning. When no such signal is available, it may still be possible to perform learning by exploiting so-called blind, or unsupervised, methods. These methods exploit certain *a priori* statistical knowledge of the input data. For a single signal, this knowledge may be in the form of its constant modulus property, or, for multiple signals, their mutual statistical independence (Haykin 2000). In Figure 2.3 the structure of a blind equaliser is shown, notice the desired response is generated from the output of a zero-memory nonlinearity. This nonlinearity is implicitly being used to test the higher-order (i.e. greater than second-order) statistical properties of the output of the adaptive equaliser. When ideal convergence of the adaptive filter is achieved, the zero-memory nonlinearity has no effect upon the signal $y(k)$ and therefore $y(k)$ has identical statistical properties to that of the channel input $s(k)$.

2.3 Gradient-Based Learning Algorithms

We provide a brief introduction to the notion of gradient-based learning. The aim is to update iteratively the weight vector \mathbf{w} of an adaptive system so that a nonnegative error measure $\mathcal{J}(\cdot)$ is reduced at each time step k ,

$$\mathcal{J}(\mathbf{w} + \Delta\mathbf{w}) < \mathcal{J}(\mathbf{w}), \quad (2.1)$$

where $\Delta\mathbf{w}$ represents the change in \mathbf{w} from one iteration to the next. This will generally ensure that after training, an adaptive system has captured the relevant properties of the unknown system that we are trying to model. Using a Taylor series expansion

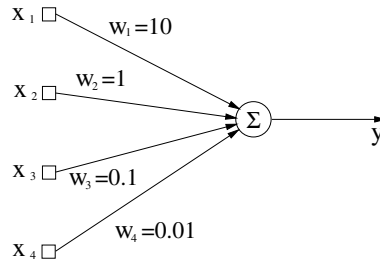


Figure 2.4 Example of a filter with widely differing weights

to approximate the error measure, we obtain²

$$\mathcal{J}(\mathbf{w}) + \Delta \mathbf{w} \frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} + \mathcal{O}(\mathbf{w}^2) < \mathcal{J}(\mathbf{w}). \quad (2.2)$$

This way, with the assumption that the higher-order terms in the left-hand side of (2.2) can be neglected, (2.1) can be rewritten as

$$\Delta \mathbf{w} \frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} < 0. \quad (2.3)$$

From (2.3), an algorithm that would continuously reduce the error measure on the run, should change the weights in the opposite direction of the gradient $\partial \mathcal{J}(\mathbf{w})/\partial \mathbf{w}$, i.e.

$$\Delta \mathbf{w} = -\eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}}, \quad (2.4)$$

where η is a small positive scalar called the learning rate, step size or adaptation parameter.

Examining (2.4), if the gradient of the error measure $\mathcal{J}(\mathbf{w})$ is steep, large changes will be made to the weights, and conversely, if the gradient of the error measure $\mathcal{J}(\mathbf{w})$ is small, namely a flat error surface, a larger step size η may be used. Gradient descent algorithms cannot, however, provide a sense of importance or hierarchy to the weights (Agarwal and Mammone 1994). For example, the value of weight w_1 in Figure 2.4 is 10 times greater than w_2 and 1000 times greater than w_4 . Hence, the component of the output of the filter within the adaptive system due to w_1 will, on the average, be larger than that due to the other weights. For a conventional gradient algorithm, however, the change in w_1 will not depend upon the relative sizes of the coefficients, but the relative sizes of the input data. This deficiency provides the motivation for certain partial update gradient-based algorithms (Douglas 1997).

It is important to notice that *gradient-descent-based algorithms inherently forget old data*, which leads to a problem called vanishing gradient and has particular importance for learning in filters with recursive structures. This issue is considered in more detail in Chapter 6.

² The explanation of the \mathcal{O} notation can be found in Appendix A.

2.4 A General Class of Learning Algorithms

To introduce a general class of learning algorithms and explain in very crude terms relationships between them, we follow the approach from Guo and Ljung (1995). Let us start from the linear regression equation,

$$y(k) = \mathbf{x}^T(k)\mathbf{w}(k) + \nu(k), \quad (2.5)$$

where $y(k)$ is the output signal, $\mathbf{x}(k)$ is a vector comprising the input signals, $\nu(k)$ is a disturbance or noise sequence, and $\mathbf{w}(k)$ is an unknown time-varying vector of weights (parameters) of the adaptive system. Variation of the weights at time k is denoted by $\mathbf{n}(k)$, and the weight change equation becomes

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{n}(k). \quad (2.6)$$

Adaptive algorithms can track the weights only approximately, hence for the following analysis we use the symbol $\hat{\mathbf{w}}$. A general expression for weight update in an adaptive algorithm is

$$\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \eta \mathbf{\Gamma}(k)(y(k) - \mathbf{x}^T(k)\hat{\mathbf{w}}(k)), \quad (2.7)$$

where $\mathbf{\Gamma}(k)$ is the adaptation gain vector, and η is the step size. To assess how far an adaptive algorithm is from the optimal solution we introduce the weight error vector, $\check{\mathbf{w}}(k)$, and a sample input matrix $\mathbf{\Sigma}(k)$ as

$$\check{\mathbf{w}}(k) = \mathbf{w}(k) - \hat{\mathbf{w}}(k), \quad \mathbf{\Sigma}(k) = \mathbf{\Gamma}(k)\mathbf{x}^T(k). \quad (2.8)$$

Equations (2.5)–(2.8) yield the following weight error equation:

$$\check{\mathbf{w}}(k+1) = (\mathbf{I} - \eta\mathbf{\Sigma}(k))\check{\mathbf{w}}(k) - \eta\mathbf{\Gamma}(k)\nu(k) + \mathbf{n}(k+1). \quad (2.9)$$

For different gains $\mathbf{\Gamma}(k)$, the following three well-known algorithms can be obtained from (2.7).³

1. The least mean square (LMS) algorithm:

$$\mathbf{\Gamma}(k) = \mathbf{x}(k). \quad (2.10)$$

2. Recursive least-squares (RLS) algorithm:

$$\mathbf{\Gamma}(k) = P(k)\mathbf{x}(k), \quad (2.11)$$

$$P(k) = \frac{1}{1-\eta} \left[P(k-1) - \eta \frac{P(k-1)\mathbf{x}(k)\mathbf{x}^T(k)P(k-1)}{1-\eta + \eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)} \right]. \quad (2.12)$$

3. Kalman filter (KF) algorithm (Guo and Ljung 1995; Kay 1993):

$$\mathbf{\Gamma}(k) = \frac{P(k-1)\mathbf{x}(k)}{R + \eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)}, \quad (2.13)$$

$$P(k) = P(k-1) - \frac{\eta P(k-1)\mathbf{x}(k)\mathbf{x}^T(k)P(k-1)}{R + \eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)} + \eta Q. \quad (2.14)$$

³ Notice that the role of η in the RLS and KF algorithm is different to that in the LMS algorithm. For RLS and KF we may put $\eta = 1$ and introduce a forgetting factor instead.

The KF algorithm is the optimal algorithm in this setting if the elements of $\mathbf{n}(k)$ and $\nu(k)$ in (2.5) and (2.6) are Gaussian noises with a covariance matrix $Q > 0$ and a scalar value $R > 0$, respectively (Kay 1993). All of these adaptive algorithms can be referred to as sequential estimators, since they refine their estimate as each new sample arrives. On the other hand, block-based estimators require all the measurements to be acquired before the estimate is formed.

Although the most important measure of quality of an adaptive algorithm is generally the covariance matrix of the weight tracking error $E[\check{\mathbf{w}}(k)\check{\mathbf{w}}^T(k)]$, due to the statistical dependence between $\mathbf{x}(k)$, $\nu(k)$ and $\mathbf{n}(k)$, precise expressions for this covariance matrix are extremely difficult to obtain.

To undertake statistical analysis of an adaptive learning algorithm, the classical approach is to assume that $\mathbf{x}(k)$, $\nu(k)$ and $\mathbf{n}(k)$ are statistically independent. Another assumption is that the homogeneous part of (2.9)

$$\check{\mathbf{w}}(k+1) = (\mathbf{I} - \eta \boldsymbol{\Sigma}(k))\check{\mathbf{w}}(k) \tag{2.15}$$

and its averaged version

$$E[\check{\mathbf{w}}(k+1)] = (\mathbf{I} - \eta E[\boldsymbol{\Sigma}(k)])E[\check{\mathbf{w}}(k)] \tag{2.16}$$

are exponentially stable in stochastic and deterministic senses (Guo and Ljung 1995).

2.4.1 Quasi-Newton Learning Algorithm

The quasi-Newton learning algorithm utilises the second-order derivative of the objective function⁴ to adapt the weights. If the change in the objective function between iterations in a learning algorithm is modelled with a Taylor series expansion, we have

$$\Delta E(\mathbf{w}) = E(\mathbf{w} + \Delta \mathbf{w}) - E(\mathbf{w}) \approx (\nabla_{\mathbf{w}} E(\mathbf{w}))^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w}. \tag{2.17}$$

After setting the differential with respect to $\Delta \mathbf{w}$ to zero, the weight update equation becomes

$$\Delta \mathbf{w} = -\mathbf{H}^{-1} \nabla_{\mathbf{w}} E(\mathbf{w}). \tag{2.18}$$

The Hessian \mathbf{H} in this equation determines not only the direction but also the step size of the gradient descent.

To conclude: adaptive algorithms mainly differ in their form of adaptation gains. The gains can be roughly divided into two classes: gradient-based gains (e.g. LMS, quasi-Newton) and Riccati equation-based gains (e.g. KF and RLS).

2.5 A Step-by-Step Derivation of the Least Mean Square (LMS) Algorithm

Consider a set of input–output pairs of data described by a mapping function f :

$$d(k) = f(\mathbf{x}(k)), \quad k = 1, 2, \dots, N. \tag{2.19}$$

⁴ The term *objective function* will be discussed in more detail later in this chapter.

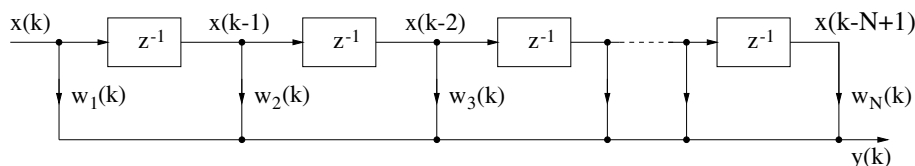


Figure 2.5 Structure of a finite impulse response filter

The function $f(\cdot)$ is assumed to be unknown. Using the concept of adaptive systems explained above, the aim is to approximate the unknown function $f(\cdot)$ by a function $F(\cdot, \mathbf{w})$ with adjustable parameters \mathbf{w} , in some prescribed sense. The function F is defined on a system with a known architecture or structure. It is convenient to define an instantaneous performance index,

$$J(\mathbf{w}(k)) = [d(k) - F(\mathbf{x}(k), \mathbf{w}(k))]^2, \quad (2.20)$$

which represents an energy measure. In that case, function F is most often just the inner product $F = \mathbf{x}^T(k)\mathbf{w}(k)$ and corresponds to the operation of a linear FIR filter structure. As before, the goal is to find an optimisation algorithm that minimises the cost function $J(\mathbf{w})$. The common choice of the algorithm is motivated by the method of steepest descent, and generates a sequence of weight vectors $\mathbf{w}(1), \mathbf{w}(2), \dots$, as

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta \mathbf{g}(k), \quad k = 0, 1, 2, \dots, \quad (2.21)$$

where $\mathbf{g}(k)$ is the gradient vector of the cost function $J(\mathbf{w})$ at the point $\mathbf{w}(k)$

$$\mathbf{g}(k) = \left. \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}(k)}. \quad (2.22)$$

The parameter η in (2.21) determines the behaviour of the algorithm:

- for η small, algorithm (2.21) converges towards the global minimum of the error performance surface;
- if the value of η approaches some critical value η_c , the trajectory of convergence on the error performance surface is either oscillatory or overdamped;
- if the value of η is greater than η_c , the system is unstable and does not converge.

These observations can only be visualised in two dimensions, i.e. for only two parameter values $w_1(k)$ and $w_2(k)$, and can be found in Widrow and Stearns (1985). If the approximation function F in the gradient descent algorithm (2.21) is linear we call such an adaptive system a linear adaptive system. Otherwise, we describe it as a nonlinear adaptive system. Neural networks belong to this latter class.

2.5.1 The Wiener Filter

Suppose the system shown in Figure 2.1 is modelled as a linear FIR filter (shown in Figure 2.5), we have $F(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T \mathbf{w}$, dropping the k index for convenience. Consequently, the instantaneous cost function $J(\mathbf{w}(k))$ is a quadratic function of the

weight vector. The Wiener filter is based upon minimising the ensemble average of this instantaneous cost function, i.e.

$$J_{\text{Wiener}}(\mathbf{w}(k)) = E[[d(k) - \mathbf{x}^T(k)\mathbf{w}(k)]^2] \quad (2.23)$$

and assuming $d(k)$ and $x(k)$ are zero mean and jointly wide sense stationary. To find the minimum of the cost function, we differentiate with respect to \mathbf{w} and obtain

$$\frac{\partial J_{\text{Wiener}}}{\partial \mathbf{w}} = -2E[e(k)\mathbf{x}(k)], \quad (2.24)$$

where $e(k) = d(k) - \mathbf{x}^T(k)\mathbf{w}(k)$.

At the Wiener solution, this gradient equals the null vector $\mathbf{0}$. Solving (2.24) for this condition yields the Wiener solution,

$$\mathbf{w} = \mathbf{R}_{\mathbf{x},\mathbf{x}}^{-1} \mathbf{r}_{\mathbf{x},d}, \quad (2.25)$$

where $\mathbf{R}_{\mathbf{x},\mathbf{x}} = E[\mathbf{x}(k)\mathbf{x}^T(k)]$ is the autocorrelation matrix of the zero mean input data $\mathbf{x}(k)$ and $\mathbf{r}_{\mathbf{x},d} = E[\mathbf{x}(k)d(k)]$ is the crosscorrelation between the input vector and the desired signal $d(k)$. The Wiener formula has the same general form as the block least-squares (LS) solution, when the exact statistics are replaced by temporal averages.

The RLS algorithm, as in (2.12), with the assumption that the input and desired response signals are jointly ergodic, approximates the Wiener solution and asymptotically matches the Wiener solution.

More details about the derivation of the Wiener filter can be found in Haykin (1996a, 1999a).

2.5.2 Further Perspective on the Least Mean Square (LMS) Algorithm

To reduce the computational complexity of the Wiener solution, which is a block solution, we can use the method of steepest descent for a recursive, or sequential, computation of the weight vector \mathbf{w} . Let us derive the LMS algorithm for an adaptive FIR filter, the structure of which is shown in Figure 2.5. In view of a general adaptive system, this FIR filter becomes the filter structure within Figure 2.1. The output of this filter is

$$y(k) = \mathbf{x}^T(k)\mathbf{w}(k). \quad (2.26)$$

Widrow and Hoff (1960) utilised this structure for adaptive processing and proposed instantaneous values of the autocorrelation and crosscorrelation matrices to calculate the gradient term within the steepest descent algorithm. The cost function they proposed was

$$J(k) = \frac{1}{2}e^2(k), \quad (2.27)$$

which is again based upon the instantaneous output error $e(k) = d(k) - y(k)$. In order to derive the weight update equation we start from the instantaneous gradient

$$\frac{\partial J(k)}{\partial \mathbf{w}(k)} = e(k) \frac{\partial e(k)}{\partial \mathbf{w}(k)}. \quad (2.28)$$

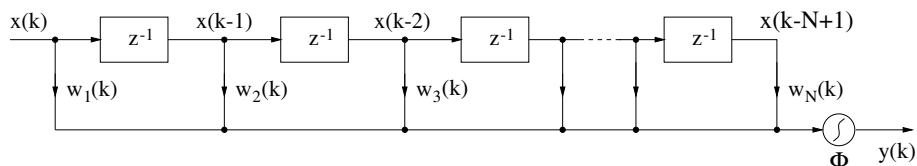


Figure 2.6 The structure of a nonlinear adaptive filter

Following the same procedure as for the general gradient descent algorithm, we obtain

$$\frac{\partial e(k)}{\partial \mathbf{w}(k)} = -\mathbf{x}(k) \quad (2.29)$$

and finally

$$\frac{\partial J(k)}{\partial \mathbf{w}(k)} = -e(k)\mathbf{x}(k). \quad (2.30)$$

The set of equations that describes the LMS algorithm is given by

$$\left. \begin{aligned} y(k) &= \sum_{i=1}^N x_i(k)w_i(k) = \mathbf{x}^T(k)\mathbf{w}(k), \\ e(k) &= d(k) - y(k), \\ \mathbf{w}(k+1) &= \mathbf{w}(k) + \eta e(k)\mathbf{x}(k). \end{aligned} \right\} \quad (2.31)$$

The LMS algorithm is a very simple yet extremely popular algorithm for adaptive filtering. It is also optimal in the H^∞ sense which justifies its practical utility (Hassibi *et al.* 1996).

2.6 On Gradient Descent for Nonlinear Structures

Adaptive filters and neural networks are formally equivalent, in fact, the structures of neural networks are generalisations of linear filters (Maass and Sontag 2000; Nerrand *et al.* 1991). Depending on the architecture of a neural network and whether it is used online or offline, two broad classes of learning algorithms are available:

- techniques that use a direct computation of the gradient, which is typical for linear and nonlinear adaptive filters;
- techniques that involve backpropagation, which is commonplace for most offline applications of neural networks.

Backpropagation is a computational procedure to obtain gradients necessary for adaptation of the weights of a neural network contained within its hidden layers and is not radically different from a general gradient algorithm.

As we are interested in neural networks for real-time signal processing, we will analyse online algorithms that involve direct gradient computation. In this section we introduce a learning algorithm for a nonlinear FIR filter, whereas learning algorithms for online training of recurrent neural networks will be introduced later. Let us start from a simple nonlinear FIR filter, which consists of the standard FIR filter cascaded

with a memoryless nonlinearity Φ as shown in Figure 2.6. This structure can be seen as a single neuron with a dynamical FIR synapse. This FIR synapse provides memory to the neuron. The output of this filter is given by

$$y(k) = \Phi(\mathbf{x}^T(k)\mathbf{w}(k)). \quad (2.32)$$

The nonlinearity $\Phi(\cdot)$ after the tap-delay line is typically a sigmoid. Using the ideas from the LMS algorithm, if the cost function is given by

$$J(k) = \frac{1}{2}e^2(k) \quad (2.33)$$

we have

$$e(k) = d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)), \quad (2.34)$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta \nabla_{\mathbf{w}} J(k), \quad (2.35)$$

where $e(k)$ is the instantaneous error at the output neuron, $d(k)$ is some teaching (desired) signal, $\mathbf{w}(k) = [w_1(k), \dots, w_N(k)]^T$ is the weight vector and $\mathbf{x}(k) = [x_1(k), \dots, x_N(k)]^T$ is the input vector.

The gradient $\nabla_{\mathbf{w}} J(k)$ can be calculated as

$$\frac{\partial J(k)}{\partial \mathbf{w}(k)} = e(k) \frac{\partial e(k)}{\partial \mathbf{w}(k)} = -e(k) \Phi'(\mathbf{x}^T(k)\mathbf{w}(k)) \mathbf{x}(k), \quad (2.36)$$

where $\Phi'(\cdot)$ represents the first derivative of the nonlinearity $\Phi(\cdot)$ and the weight update Equation (2.35) can be rewritten as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \Phi'(\mathbf{x}^T(k)\mathbf{w}(k)) e(k) \mathbf{x}(k). \quad (2.37)$$

This is the weight update equation for a direct gradient algorithm for a nonlinear FIR filter.

2.6.1 Extension to a General Neural Network

When deriving a direct gradient algorithm for a general neural network, the network architecture should be taken into account. For large networks for offline processing, classical backpropagation is the most convenient algorithm. However, for online learning, extensions of the previous algorithm should be considered.

2.7 On Some Important Notions From Learning Theory

In this section we discuss in more detail the inter-relations between the error, error function and objective function in learning theory.

2.7.1 Relationship Between the Error and the Error Function

The error at the output of an adaptive system is defined as the difference between the output value of the network and the target (desired output) value. For instance,

the instantaneous error $e(k)$ is defined as

$$e(k) = d(k) - y(k). \quad (2.38)$$

The instantaneous error can be positive, negative or zero, and is therefore not a good candidate for the criterion function for training adaptive systems. Hence we look for another function, called the *error* function, that is a function of the instantaneous error, but is suitable as a criterion function for learning. Error functions are also called *loss* functions. They are defined so that an increase in the error function corresponds to a reduction in the quality of learning, and they are nonnegative. An error function can be defined as

$$E(N) = \sum_{i=0}^N e^2(i) \quad (2.39)$$

or as an average value

$$\bar{E}(N) = \frac{1}{N+1} \sum_{i=0}^N e^2(i). \quad (2.40)$$

2.7.2 The Objective Function

The *objective* function is a function that we want to minimise during training. It can be equal to an error function, but often it may include other terms to introduce constraints. For instance in generalisation, too large a network might lead to overfitting. Hence the objective function can consist of two parts, one for the error minimisation and the other which is either a penalty for a large network or a penalty term for excessive increase in the weights of the adaptive system or some other chosen function (Tikhonov *et al.* 1998). An example of such an objective function for online learning is

$$J(k) = \frac{1}{N} \sum_{i=1}^N (e^2(k-i+1) + G(\|\mathbf{w}(k-i+1)\|_2^2)), \quad (2.41)$$

where G is some linear or nonlinear function. We often use symbols E and J interchangeably to denote the cost function.

2.7.3 Types of Learning with Respect to the Training Set and Objective Function

Batch learning

Batch learning is also known as epochwise, or offline learning, and is a common strategy for offline training. The idea is to adapt the weights once the whole training set has been presented to an adaptive system. It can be described by the following steps.

1. Initialise the weights
2. Repeat
 - Pass all the training data through the network

- Sum the errors after each particular pattern
- Update the weights based upon the total error
- Stop if some prescribed error performance is reached

The counterpart of batch learning is so-called incremental learning, online, or pattern learning. The procedure for this type of learning is as follows.

1. Initialise the weights
2. Repeat
 - Pass one pattern through the network
 - Update the weights based upon the instantaneous error
 - Stop if some prescribed error performance is reached

The choice of the type of learning is very much dependent upon application. Quite often, for networks that need initialisation, we perform one type of learning in the initialisation procedure, which is by its nature an offline procedure, and then use some other learning strategy while the network is running. Such is the case with recurrent neural networks for online signal processing (Mandic and Chambers 1999f).

2.7.4 *Deterministic, Stochastic and Adaptive Learning*

Deterministic learning is an optimisation technique based on an objective function which always produces the same result, no matter how many times we recompute it. Deterministic learning is always offline.

Stochastic learning is useful when the objective function is affected by noise and local minima. It can be employed within the context of a gradient descent learning algorithm. The idea is that the learning rate gradually decreases during training and hence the steps on the error performance surface in the beginning of training are large which speeds up training when far from the optimal solution. The learning rate is small when approaching the optimal solution, hence reducing misadjustment. This gradual reduction of the learning rate can be achieved by e.g. annealing (Kirkpatrick *et al.* 1983; Rose 1998; Szu and Hartley 1987).

The idea behind the concept of adaptive learning is to forget the past when it is no longer relevant and adapt to the changes in the environment. The terms ‘adaptive learning’ or ‘gear-shifting’ are sometimes used for gradient methods in which the learning rate is changed during training.

2.7.5 *Constructive Learning*

Constructive learning deals with the change of architecture or interconnections in the network during training. Neural networks for which topology can change over time are called *ontogenic* neural networks (Fiesler and Beale 1997). Two basic classes of constructive learning are network growing and network pruning. In the network growing approach, learning begins with a network with no hidden units, and if the

error is too big, new hidden units are added to the network, training resumes, and so on. The most used algorithm based upon network growing is the so-called cascade-correlation algorithm (Hoehfeld and Fahlman 1992). Network pruning starts from a large network and if the error in learning is smaller than allowed, the network size is reduced until the desired ratio between accuracy and network size is reached (Reed 1993; Sum *et al.* 1999).

2.7.6 Transformation of Input Data, Learning and Dimensionality

A natural question is whether to linearly/nonlinearly transform the data before feeding them to an adaptive processor. This is particularly important for neural networks, which are nonlinear processors. If we consider each neuron as a basic component of a neural network, then we can refer to a general neural network as a system with componentwise nonlinearities. To express this formally, consider a scalar function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ and systems of the form,

$$\mathbf{y}(k) = \boldsymbol{\sigma}(\mathbf{A}\mathbf{x}(k)), \quad (2.42)$$

where the matrix \mathbf{A} is an $N \times N$ matrix and $\boldsymbol{\sigma}$ is applied componentwise

$$\boldsymbol{\sigma}(x_1(k), \dots, x_N(k)) = (\sigma(x_1(k)), \dots, \sigma(x_N(k))). \quad (2.43)$$

Systems of this type arise in a wide variety of situations. For a linear σ , we have a linear system. If the range of σ is finite, the state vector of (2.42) takes values from a finite set, and dynamical properties can be analysed in time which is polynomial in the number of possible states. Throughout this book we are interested in functions, σ , and combination matrices, \mathbf{A} , which would guarantee a fixed point of this mapping. Neural networks are commonly of the form (2.42). In such a context we call σ the activation function. Results of Siegelmann and Sontag (1995) show that saturated linear systems (piecewise linear) can represent Turing machines, which is achieved by encoding the transition rules of the Turing machine in the matrix \mathbf{A} .

The curse of dimensionality

The curse of dimensionality (Bellman 1961) refers to the exponential growth of computation needed for a specific task as a function of the dimensionality of the input space. In neural networks, a network quite often has to deal with many irrelevant inputs which, in turn, increase the dimensionality of the input space. In such a case, the network uses much of its resources to represent and compute irrelevant information, which hampers processing of the desired information. A remedy for this problem is preprocessing of input data, such as feature extraction, and to introduce some importance function to the input samples. The curse of dimensionality is particularly prominent in unsupervised learning algorithms. Radial basis functions are also prone to this problem. Selection of a neural network model must therefore be suited for a particular task. Some *a priori* information about the data and scaling of the inputs can help to reduce the severity of the problem.

Transformations on the input data

Activation functions used in neural networks are centred around a certain value in their output space. For instance, the mean of the logistic function is 0.5, whereas the tanh function is centred around zero. Therefore, in order to perform efficient prediction, we should match the range of the input data, their mean and variance, with the range of the chosen activation function. There are several operations that we could perform on the input data, such as the following.

1. Normalisation, which in this context means dividing each element of the input vector $\mathbf{x}(k)$ by its squared norm, i.e. $x_i(k) \in \mathbf{x}(k) \rightarrow x_i(k)/\|\mathbf{x}(k)\|_2^2$.
2. Rescaling, which means transforming the input data in the manner that we multiply/divide them by a constant and also add/subtract a constant from the data.⁵
3. Standardisation, which is borrowed from statistics, where, for instance, a random Gaussian vector is standardised if its mean is subtracted from it, and the vector is then divided by its standard deviation. The resulting random variable is called a ‘standard normal’ random variable with zero mean and unity standard deviation. Some examples of data standardisation are

- Standardisation to zero mean and unity standard deviation can be performed as

$$\text{mean} = \frac{\sum_i X_i}{N}, \quad \text{std} = \sqrt{\frac{\sum_i (X_i - \text{mean})^2}{N - 1}}.$$

The standardised quantity becomes $S_i = (X_i - \text{mean})/\text{std}$.

- Standardise X to midrange 0 and range 2. This can be achieved by

$$\text{midrange} = \frac{1}{2}(\max_i X_i + \min_i X_i), \quad \text{range} = \max_i X_i - \min_i X_i,$$

$$S_i = \frac{X_i - \text{midrange}}{\text{range}/2}.$$

4. Principal component analysis (PCA) represents the data by a set of unit norm vectors called normalised eigenvectors. The eigenvectors are positioned along the directions of greatest data variance. The eigenvectors are found from the covariance matrix \mathbf{R} of the input dataset. An eigenvalue λ_i , $i = 1, \dots, N$, is associated with each eigenvector. Every input data vector is then represented by a linear combination of eigenvectors.

As pointed out earlier, standardising input variables has an effect on training, since steepest descent algorithms are sensitive to scaling due to the change in the weights being proportional to the value of the gradient and the input data.

⁵ In real life a typical rescaling is transforming the temperature from Celsius into Fahrenheit scale.

Nonlinear transformations of the data

This method to transform the data can help when the dynamic range of the data is too high. In that case, for instance, we typically apply the log function to the input data. The log function is often applied in the error and objective functions for the same purposes.

2.8 Learning Strategies

To construct an optimal neural approximating model we have to determine an appropriate training set containing all the relevant information of the process and define a suitable topology that matches the complexity and performance requirements. The training set construction issue requires four entities to be considered (Alippi and Piuri 1996; Bengio 1995; Haykin and Li 1995; Shadafan and Niranjana 1993):

- the number of training data samples N_D ;
- the number of patterns N_P constituting a batch;
- the number of batches N_B to be extracted from the training set;
- the number of times the generic batch is presented to the network during learning.

The assumption is that the training set is sufficiently rich so that it contains all the relevant information necessary for learning.

The requirement coincides with the hypothesis that the training data have been generated by a fully exciting input signal, such as white noise, which is able to excite all the process dynamics. White noise is a persistently exciting input signal and is used for the driving component of moving average (MA), autoregressive (AR) and autoregressive moving average (ARMA) models.

2.9 General Framework for the Training of Recurrent Networks by Gradient-Descent-Based Algorithms

In this section we summarise some of the important concepts mentioned earlier.

2.9.1 Adaptive Versus Nonadaptive Training

The training of a network makes use of two sequences, the sequence of inputs and the sequence of corresponding desired outputs. If the network is first trained (with a training sequence of finite length) and subsequently used (with the fixed weights obtained from training), this mode of operation is referred to as *non-adaptive* (Nerrand *et al.* 1994). Conversely, the term *adaptive* refers to the mode of operation whereby the network is trained permanently throughout its application (with a training sequence of infinite length). Therefore, the adaptive network is suitable for input processes which exhibit statistically non-stationary behaviour, a situation which is normal in the fields of adaptive control and signal processing (Bengio 1995; Haykin 1996a; Haykin and

Li 1995; Khotanzad and Lu 1990; Narendra and Parthasarathy 1990; Nerrand *et al.* 1994).

2.9.2 Performance Criterion, Cost Function, Training Function

The computation of the coefficients during training aims at finding a system whose operation is optimal with respect to some performance criterion which may be either qualitative, e.g. (subjective) quality of speech reconstruction, or quantitative, e.g. maximising signal to noise ratio for spatial filtering. The goal is to define a positive *training function* which is such that a decrease of this function through modifications of the coefficients of the network leads to an improvement of the performance of the system (Bengio 1995; Haykin and Li 1995; Nerrand *et al.* 1994; Qin *et al.* 1992). In the case of non-adaptive training, the training function is defined as a function of all the data of the training set (in such a case, it is usually termed as a *cost function*). The minimum of the cost function corresponds to the optimal performance of the system. Training is an optimisation procedure, conventionally using gradient-based methods.

In the case of adaptive training, it is impossible, in most instances, to define a time-independent cost function whose minimisation leads to a system that is optimal with respect to the performance criterion. Therefore, the training function is time dependent. The modification of the coefficients is computed continually from the gradient of the training function. The latter involves the data pertaining to a time window of finite length, which shifts in time (sliding window) and the coefficients are updated at each sampling time.

2.9.3 Recursive Versus Nonrecursive Algorithms

A nonrecursive algorithm employs a cost function (i.e. a training function defined on a fixed window), whereas a recursive algorithm makes use of a training function defined on a sliding window of data. An adaptive system must be trained by a recursive algorithm, whereas a non-adaptive system may be trained either by a nonrecursive or by a recursive algorithm (Nerrand *et al.* 1994).

2.9.4 Iterative Versus Noniterative Algorithms

An iterative algorithm performs coefficient modifications several times from a set of data pertaining to a given data window, a non-iterative algorithm makes only one (Nerrand *et al.* 1994). For instance, the conventional LMS algorithm (2.31) is thus a recursive, non-iterative algorithm operating on a sliding window.

2.9.5 Supervised Versus Unsupervised Algorithms

A supervised learning algorithm performs learning by using a *teaching signal*, i.e. the desired output signal, while an unsupervised learning algorithm, as in blind signal processing, has no reference signal as a teaching input signal. An example of a supervised learning algorithm is the *delta rule*, while unsupervised learning algorithms are,

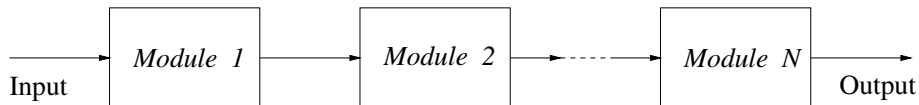


Figure 2.7 A cascaded realisation of a general system

for example, the *reinforcement learning algorithm* and the *competitive rule* (*‘winner takes all’*) *algorithm*, whereby there is some sense of concurrency between the elements of the network structure (Bengio 1995; Haykin and Li 1995).

2.9.6 Pattern Versus Batch Learning

Updating the network weights by *pattern learning* means that the weights of the network are updated immediately after each pattern is fed in. The other approach is to take all the data as a whole batch, and the network is not updated until the entire batch of data is processed. This approach is referred to as *batch learning* (Haykin and Li 1995; Qin *et al.* 1992).

It can be shown (Qin *et al.* 1992) that while considering feedforward networks (FFN), after one training sweep through all the data, the pattern learning is a first-order approximation of the batch learning with respect to the learning rate η . Therefore, the FFN pattern learning approximately implements the FFN batch learning after one batch interval. After multiple sweeps through the training data, the difference between the FFN pattern learning and FFN batch learning is of the order⁶ $\mathcal{O}(\eta^2)$. Therefore, for small training rates, the FFN pattern learning approximately implements FFN batch learning after multiple sweeps through the training data. For recurrent networks, the weight updating slopes for pattern learning and batch learning are different⁷ (Qin *et al.* 1992). However, the difference could also be controlled by the learning rate η . The difference will converge to zero as quickly as η goes to zero⁸ (Qin *et al.* 1992).

2.10 Modularity Within Neural Networks

The hierarchical levels in neural network architectures are synapses, neurons, layers and neural networks, and will be discussed in Chapter 5. The next step would be combinations of neural networks. In this case we consider modular neural networks. Modular neural networks are composed of a set of smaller subnetworks (modules), each performing a subtask of the complete problem. To depict this problem, let us recourse to the case of linear adaptive filters described by a transfer function in the

⁶ In fact, if the data being processed exhibit highly stationary behaviour, then the average error calculated after FFN batch learning is very close to the instantaneous error calculated after FFN pattern learning, e.g. the speech data can be considered as being stationary within an observed frame. That forms the basis for use of various real-time and recursive learning algorithms, e.g. RTRL.

⁷ It can be shown (Qin *et al.* 1992) that for feedforward networks, the updated weights for both pattern learning and batch learning adapt at the same slope (derivative $dw/d\eta$) with respect to the learning rate η . For recurrent networks, this is not the case.

⁸ In which case we have a very slow learning process.

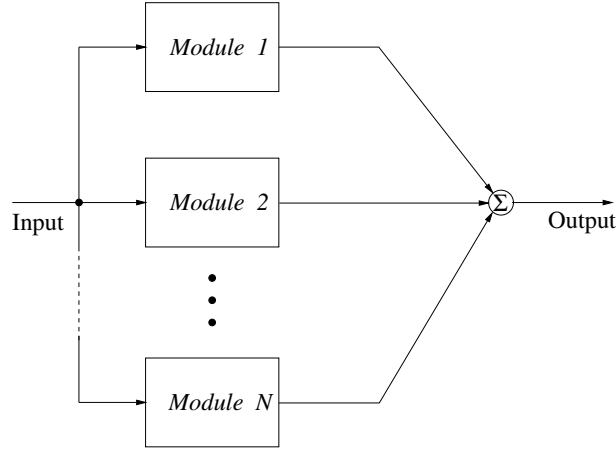


Figure 2.8 A parallel realisation of a general system

z -domain $H(z)$ as

$$H(z) = \frac{\sum_{k=0}^M b(k)z^{-k}}{1 + \sum_{k=1}^N a(k)z^{-k}}. \quad (2.44)$$

We can rearrange this function either in a cascaded manner as

$$H(z) = A \prod_{k=1}^{\max\{M,N\}} \frac{1 - \beta_k z^{-1}}{1 - \alpha_k z^{-1}}, \quad (2.45)$$

or in a parallel manner as

$$H(z) = \sum_{k=1}^N \frac{A_k}{1 - \alpha_k z^{-1}}, \quad (2.46)$$

where for simplicity we have assumed first-order poles and zeros of $H(z)$. A cascaded realisation of a general system is shown in Figure 2.7, whereas a parallel realisation of a general system is shown in Figure 2.8. We can also combine neural networks in these two configurations. An example of cascaded neural network is the so-called pipelined recurrent neural network, whereas an example of a parallel realisation of a neural network is the associative Gaussian mixture model, or winner takes all network. Taking into account that neural networks are nonlinear systems, we talk about nested modular architectures instead of cascaded architectures. The nested neural scheme can be written as

$$F(W, X) = \Phi \left(\sum_n w_n \Phi \left(\sum_i v_i \Phi \left(\cdots \Phi \left(\sum_j u_j X_j \right) \cdots \right) \right) \right), \quad (2.47)$$

where Φ is a sigmoidal function. It corresponds to a multilayer network of units that sum their inputs with ‘weights’ $W = \{w_n, v_i, u_j, \dots\}$ and then perform a sigmoidal

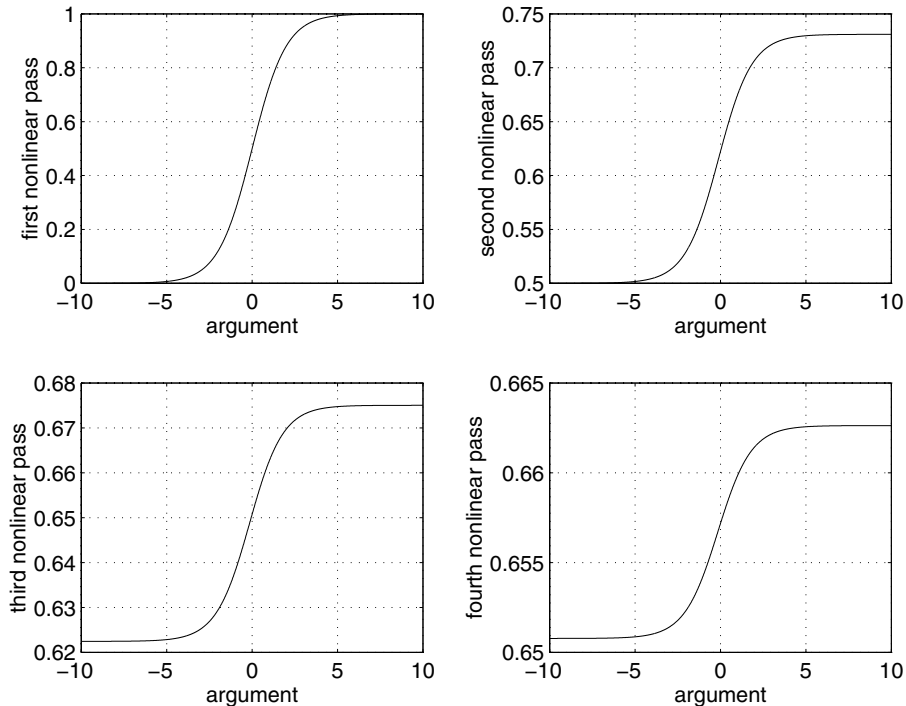


Figure 2.9 Effects of nesting sigmoid nonlinearities: first, second, third and fourth pass

transformation of this sum. Its motivation is that the function

$$F(W, X) = \Phi \left(\sum_n w_n \Phi \left(\sum_j u_j X_j \right) \right) \quad (2.48)$$

can approximate arbitrarily well any continuous multivariate function (Funahashi 1989; Poggio and Girosi 1990).

Since we use sigmoid ‘squashing’ activation functions, modular structures contribute to a general stability issue. The effects of a simple scheme of nested sigmoids are shown in Figure 2.9. From Figure 2.9 we see that pure nesting successively reduces the range of the output signal, bringing this composition of nonlinear functions to the fixed point of the employed nonlinearity for sufficiently many nested sigmoids.

Modular networks possess some advantages over classical networks, since the overall complex function is simplified and modules possibly do not have hidden units which speeds up training. Also, input data might be decomposable into subsets which can be fed to separate modules. Utilising modular neural networks has not only computational advantages but also development advantages, improved efficiency, improved interpretability and easier hardware implementation. Also, there are strong suggestions from biology that modular structures are exploited in cognitive mechanisms (Fiesler and Beale 1997).

2.11 Summary

Configurations of general adaptive systems have been provided, and the prediction configuration has been introduced within this framework. Gradient-descent-based learning algorithms have then been developed for these configurations, with an emphasis on the LMS algorithm. A thorough discussion of learning modes and learning parameters is given. Finally, modularity within neural networks has been addressed.

3

Network Architectures for Prediction

3.1 Perspective

The architecture, or structure, of a predictor underpins its capacity to represent the dynamic properties of a statistically nonstationary discrete time input signal and hence its ability to predict or forecast some future value. This chapter therefore provides an overview of available structures for the prediction of discrete time signals.

3.2 Introduction

The basic building blocks of all discrete time predictors are adders, delayers, multipliers and for the nonlinear case zero-memory nonlinearities. The manner in which these elements are interconnected describes the architecture of a predictor. The foundations of linear predictors for statistically stationary signals are found in the work of Yule (1927), Kolmogorov (1941) and Wiener (1949). The later studies of Box and Jenkins (1970) and Makhoul (1975) were built upon these fundamentals. Such linear structures are very well established in digital signal processing and are classified either as finite impulse response (FIR) or infinite impulse response (IIR) digital filters (Oppenheim *et al.* 1999). FIR filters are generally realised without feedback, whereas IIR filters¹ utilise feedback to limit the number of parameters necessary for their realisation. The presence of feedback implies that the consideration of stability underpins the design of IIR filters. In statistical signal modelling, FIR filters are better known as moving average (MA) structures and IIR filters are named autoregressive (AR) or autoregressive moving average (ARMA) structures. The most straightforward version of nonlinear filter structures can easily be formulated by including a nonlinear operation in the output stage of an FIR or an IIR filter. These represent simple examples of nonlinear autoregressive (NAR), nonlinear moving average (NMA) or nonlinear autoregressive moving average (NARMA) structures (Nerrand *et al.* 1993). Such filters have immediate application in the prediction of discrete time random signals that arise from some

¹ FIR filters can be represented by IIR filters, however, in practice it is not possible to represent an arbitrary IIR filter with an FIR filter of finite length.

nonlinear physical system, as for certain speech utterances. These filters, moreover, are strongly linked to single neuron neural networks.

The neuron, or node, is the basic processing element within a neural network. The structure of a neuron is composed of multipliers, termed synaptic weights, or simply weights, which scale the inputs, a linear combiner to form the activation potential, and a certain zero-memory nonlinearity to model the activation function. Different neural network architectures are formulated by the combination of multiple neurons with various interconnections, hence the term connectionist modelling (Rumelhart *et al.* 1986). Feedforward neural networks, as for FIR/MA/NMA filters, have no feedback within their structure. Recurrent neural networks, on the other hand, similarly to IIR/AR/NAR/NARMA filters, exploit feedback and hence have much more potential structural richness. Such feedback can either be local to the neurons or global to the network (Haykin 1999b; Tsoi and Back 1997). When the inputs to a neural network are delayed versions of a discrete time random input signal the correspondence between the architectures of nonlinear filters and neural networks is evident.

From a biological perspective (Marmarelis 1989), the *prototypical* neuron is composed of a cell body (soma), a tree-like element of fibres (dendrites) and a long fibre (axon) with sparse branches (collaterals). The axon is attached to the soma at the *axon hillock*, and, together with its collaterals, ends at synaptic terminals (boutons), which are employed to pass information onto their neurons through *synaptic junctions*. The soma contains the nucleus and is attached to the trunk of the dendritic tree from which it receives incoming information. The dendrites are conductors of input information to the soma, i.e. input ports, and usually exhibit a high degree of arborisation.

The possible architectures for nonlinear filters or neural networks are manifold. The state-space representation from system theory is established for linear systems (Kailath 1980; Kailath *et al.* 2000) and provides a mechanism for the representation of structural variants. An insightful canonical form for neural networks is provided by Nerrand *et al.* (1993), by the exploitation of state-space representation which facilitates a unified treatment of the architectures of neural networks.²

3.3 Overview

The chapter begins with an explanation of the concept of prediction of a statistically stationary discrete time random signal. The building blocks for the realisation of linear and nonlinear predictors are then discussed. These same building blocks are also shown to be the basic elements necessary for the realisation of a neuron. Emphasis is placed upon the particular zero-memory nonlinearities used in the output of nonlinear filters and activation functions of neurons.

An aim of this chapter is to highlight the correspondence between the structures in nonlinear filtering and neural networks, so as to remove the apparent boundaries between the work of practitioners in control, signal processing and neural engineering. Conventional linear filter models for discrete time random signals are introduced and,

² ARMA models also have a canonical (up to an invariant) representation.

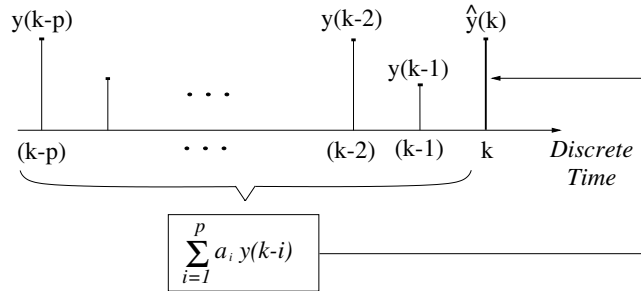


Figure 3.1 Basic concept of linear prediction

with the aid of statistical modelling, motivate the structures for linear predictors; their nonlinear counterparts are then developed.

A feedforward neural network is next introduced in which the nonlinear elements are distributed throughout the structure. To employ such a network as a predictor, it is shown that short-term memory is necessary, either at the input or integrated within the network. Recurrent networks follow naturally from feedforward neural networks by connecting the output of the network to its input. The implications of local and global feedback in neural networks are also discussed.

The role of state-space representation in architectures for neural networks is described and this leads to a canonical representation. The chapter concludes with some comments.

3.4 Prediction

A real discrete time random signal $\{y(k)\}$, where k is the discrete time index and $\{\cdot\}$ denotes the set of values, is most commonly obtained by sampling some analogue measurement. The voice of an individual, for example, is translated from pressure variation in air into a continuous time electrical signal by means of a microphone and then converted into a digital representation by an analogue-to-digital converter. Such discrete time random signals have statistics that are time-varying, but on a short-term basis, the statistics may be assumed to be time invariant.

The principle of the prediction of a discrete time signal is represented in Figure 3.1 and forms the basis of linear predictive coding (LPC) which underlies many compression techniques. The value of signal $y(k)$ is predicted on the basis of a sum of p past values, i.e. $y(k-1), y(k-2), \dots, y(k-p)$, weighted, by the coefficients a_i , $i = 1, 2, \dots, p$, to form a prediction, $\hat{y}(k)$. The prediction error, $e(k)$, thus becomes

$$e(k) = y(k) - \hat{y}(k) = y(k) - \sum_{i=1}^p a_i y(k-i). \tag{3.1}$$

The estimation of the parameters a_i is based upon minimising some function of the error, the most convenient form being the mean square error, $E[e^2(k)]$, where $E[\cdot]$ denotes the statistical expectation operator, and $\{y(k)\}$ is assumed to be statistically

wide sense stationary,³ with zero mean (Papoulis 1984). A fundamental advantage of the mean square error criterion is the so-called orthogonality condition, which implies that

$$E[e(k)y(k-j)] = 0, \quad j = 1, 2, \dots, p, \quad (3.2)$$

is satisfied only when a_i , $i = 1, 2, \dots, p$, take on their optimal values. As a consequence of (3.2) and the linear structure of the predictor, the optimal weight parameters may be found from a set of linear equations, named the Yule–Walker equations (Box and Jenkins 1970),

$$\begin{bmatrix} r_{yy}(0) & r_{yy}(1) & \cdots & r_{yy}(p-1) \\ r_{yy}(1) & r_{yy}(0) & \cdots & r_{yy}(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_{yy}(p-1) & r_{yy}(p-2) & \cdots & r_{yy}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} r_{yy}(1) \\ r_{yy}(2) \\ \vdots \\ r_{yy}(p) \end{bmatrix}, \quad (3.3)$$

where $r_{yy}(\tau) = E[y(k)y(k+\tau)]$ is the value of the autocorrelation function of $\{y(k)\}$ at lag τ . These equations may be equivalently written in matrix form as

$$\mathbf{R}_{yy}\mathbf{a} = \mathbf{r}_{yy}, \quad (3.4)$$

where $\mathbf{R}_{yy} \in \mathbb{R}^{p \times p}$ is the autocorrelation matrix and $\mathbf{a}, \mathbf{r}_{yy} \in \mathbb{R}^p$ are, respectively, the parameter vector of the predictor and the crosscorrelation vector. The Toeplitz symmetric structure of \mathbf{R}_{yy} is exploited in the Levinson–Durbin algorithm (Hayes 1997) to solve for the optimal parameters in $\mathcal{O}(p^2)$ operations. The quality of the prediction is judged by the minimum mean square error (MMSE), which is calculated from $E[e^2(k)]$ when the weight parameters of the predictor take on their optimal values. The MMSE is calculated from $r_{yy}(0) - \sum_{i=1}^p a_i r_{yy}(i)$.

Real measurements can only be assumed to be locally wide sense stationary and therefore, in practice, the autocorrelation function values must be estimated from some finite length measurement in order to employ (3.3). A commonly used, but statistically biased and low variance (Kay 1993), autocorrelation estimator for application to a finite length N measurement, $\{y(0), y(1), \dots, y(N-1)\}$, is given by

$$\hat{r}_{yy}(\tau) = \frac{1}{N} \sum_{k=0}^{N-\tau-1} y(k)y(k+\tau), \quad \tau = 0, 1, 2, \dots, p. \quad (3.5)$$

These estimates would then replace the exact values in (3.3) from which the weight parameters of the predictor are calculated. This procedure, however, needs to be repeated for each new length N measurement, and underlies the operation of a block-based predictor.

A second approach to the estimation of the weight parameters $\mathbf{a}(k)$ of a predictor is the sequential, adaptive or learning approach. The estimates of the weight parameters are refined at each sample number, k , on the basis of the new sample $y(k)$ and the prediction error $e(k)$. This yields an update equation of the form

$$\hat{\mathbf{a}}(k+1) = \hat{\mathbf{a}}(k) + \eta f(e(k), \mathbf{y}(k)), \quad k \geq 0, \quad (3.6)$$

³ Wide sense stationarity implies that the mean is constant, the autocorrelation function is only a function of the time lag and the variance is finite.

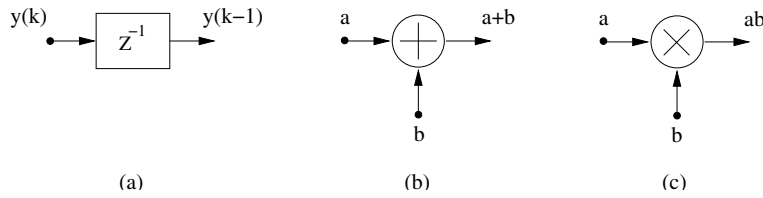


Figure 3.2 Building blocks of predictors: (a) delayer, (b) adder, (c) multiplier

where η is termed the adaptation gain, $f(\cdot)$ is some function dependent upon the particular learning algorithm, whereas $\hat{\mathbf{a}}(k)$ and $\mathbf{y}(k)$ are, respectively, the estimated weight vector and the predictor input vector. Without additional prior knowledge, zero or random values are chosen for the initial values of the weight parameters in (3.6), i.e. $\hat{a}_i(0) = 0$, or n_i , $i = 1, 2, \dots, p$, where n_i is a random variable drawn from a suitable distribution. The sequential approach to the estimation of the weight parameters is particularly suitable for operation of predictors in statistically nonstationary environments. Both the block and sequential approach to the estimation of the weight parameters of predictors can be applied to linear and nonlinear structure predictors.

3.5 Building Blocks

In Figure 3.2 the basic building blocks of discrete time predictors are shown. A simple delayer has input $y(k)$ and output $y(k-1)$, note that the sampling period is normalised to unity. From linear discrete time system theory, the delay operation can also be conveniently represented in \mathcal{Z} -domain notation as the z^{-1} operator⁴ (Oppenheim *et al.* 1999). An adder, or sumer, simply produces an output which is the sum of all the components at its input. A multiplier, or scaler, used in a predictor generally has two inputs and yields an output which is the product of the two inputs. The manner in which delayers, adders and multipliers are interconnected determines the architecture of linear predictors. These architectures, or structures, are shown in block diagram form in the ensuing sections.

To realise nonlinear filters and neural networks, zero-memory nonlinearities are required. Three zero-memory nonlinearities, as given in Haykin (1999b), with inputs $v(k)$ and outputs $\Phi(k)$ are described by the following operations:

$$\text{Threshold:} \quad \Phi(v(k)) = \begin{cases} 0, & v(k) < 0, \\ 1, & v(k) \geq 0, \end{cases} \quad (3.7)$$

$$\text{Piecewise-linear:} \quad \Phi(v(k)) = \begin{cases} 0, & v(k) \leq -\frac{1}{2}, \\ v(k), & -\frac{1}{2} < v(k) < +\frac{1}{2}, \\ 1, & v(k) \geq \frac{1}{2}, \end{cases} \quad (3.8)$$

$$\text{Logistic:} \quad \Phi(v(k)) = \frac{1}{1 + e^{-\beta v(k)}}, \quad \beta \geq 0. \quad (3.9)$$

⁴ The z^{-1} operator is a delay operator such that $\mathcal{Z}(y(k-1)) = z^{-1}\mathcal{Z}(y(k))$.

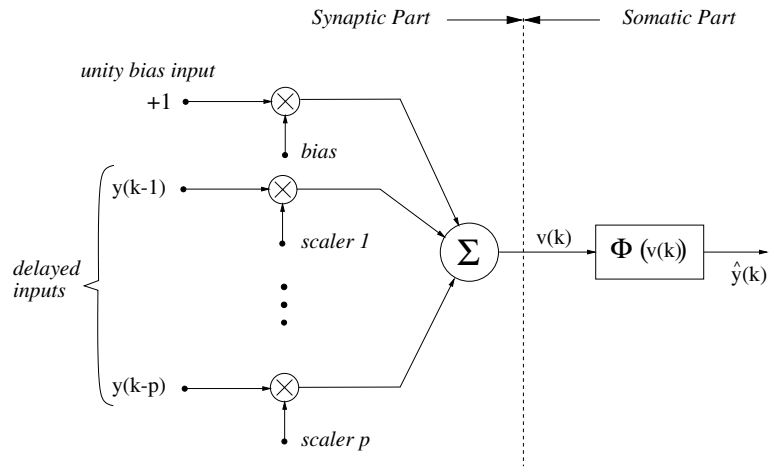


Figure 3.3 Structure of a neuron for prediction

The most commonly used nonlinearity is the logistic function since it is continuously differentiable and hence facilitates the analysis of the operation of neural networks. This property is crucial in the development of first- and second-order learning algorithms. When $\beta \rightarrow \infty$, moreover, the logistic function becomes the unipolar threshold function. The logistic function is a strictly nondecreasing function which provides for a gradual transition from linear to nonlinear operation. The inclusion of such a zero-memory nonlinearity in the output stage of the structure of a linear predictor facilitates the design of nonlinear predictors.

The threshold nonlinearity is well-established in the neural network community as it was proposed in the seminal work of McCulloch and Pitts (1943), however, it has a discontinuity at the origin. The piecewise-linear model, on the other hand, operates in a linear manner for $|v(k)| < \frac{1}{2}$ and otherwise saturates at zero or unity. Although easy to implement, neither of these zero-memory nonlinearities facilitates the analysis of the operation of nonlinear structures, because of badly behaved derivatives.

Neural networks are composed of basic processing units named neurons, or nodes, in analogy with the biological elements present within the human brain (Haykin 1999b). The basic building blocks of such artificial neurons are identical to those for nonlinear predictors. The block diagram of an artificial neuron⁵ is shown in Figure 3.3. In the context of prediction, the inputs are assumed to be delayed versions of $y(k)$, i.e. $y(k-i)$, $i = 1, 2, \dots, p$. There is also a constant bias input with unity value. These inputs are then passed through $(p+1)$ multipliers for scaling. In neural network parlance, this operation in scaling the inputs corresponds to the role of the synapses in physiological neurons. A sumer then linearly combines (in fact this is an affine transformation) these scaled inputs to form an output, $v(k)$, which is termed the induced local field or activation potential of the neuron. Save for the presence of the bias input, this output is identical to the output of a linear predictor. This component of the neuron, from a biological perspective, is termed the synaptic part (Rao and Gupta 1993). Finally,

⁵ The term ‘artificial neuron’ will be replaced by ‘neuron’ in the sequel.

$v(k)$ is passed through a zero-memory nonlinearity to form the output, $\hat{y}(k)$. This zero-memory nonlinearity is called the (nonlinear) activation function of a neuron and can be referred to as the somatic part (Rao and Gupta 1993). Such a neuron is a static mapping between its input and output (Hertz *et al.* 1991) and is very different from the dynamic form of a biological neuron. The synergy between nonlinear predictors and neurons is therefore evident. The structural power of neural networks in prediction results, however, from the interconnection of many such neurons to achieve the overall predictor structure in order to distribute the underlying nonlinearity.

3.6 Linear Filters

In digital signal processing and linear time series modelling, linear filters are well-established (Hayes 1997; Oppenheim *et al.* 1999) and have been exploited for the structures of predictors. Essentially, there are two families of filters: those without feedback, for which their output depends only upon current and past input values; and those with feedback, for which their output depends both upon input values and past outputs. Such filters are best described by a constant coefficient difference equation, the most general form of which is given by

$$y(k) = \sum_{i=1}^p a_i y(k-i) + \sum_{j=0}^q b_j e(k-j), \quad (3.10)$$

where $y(k)$ is the output, $e(k)$ is the input,⁶ a_i , $i = 1, 2, \dots, p$, are the (AR) feedback coefficients and b_j , $j = 0, 1, \dots, q$, are the (MA) feedforward coefficients. In causal systems, (3.10) is satisfied for $k \geq 0$ and the initial conditions, $y(i)$, $i = -1, -2, \dots, -p$, are generally assumed to be zero. The block diagram for the filter represented by (3.10) is shown in Figure 3.4. Such a filter is termed an autoregressive moving average (ARMA(p, q)) filter, where p is the order of the autoregressive, or feedback, part of the structure, and q is the order of the moving average, or feedforward, element of the structure. Due to the feedback present within this filter, the impulse response, namely the values of $y(k)$, $k \geq 0$, when $e(k)$ is a discrete time impulse, is infinite in duration and therefore such a filter is termed an infinite impulse response (IIR) filter within the field of digital signal processing.

The general form of (3.10) is simplified by removing the feedback terms to yield

$$y(k) = \sum_{j=0}^q b_j e(k-j). \quad (3.11)$$

Such a filter is termed moving average (MA(q)) and has a finite impulse response, which is identical to the parameters b_j , $j = 0, 1, \dots, q$. In digital signal processing, therefore, such a filter is named a finite impulse response (FIR) filter. Similarly, (3.10)

⁶ Notice $e(k)$ is used as the filter input, rather than $x(k)$, for consistency with later sections on prediction error filtering.

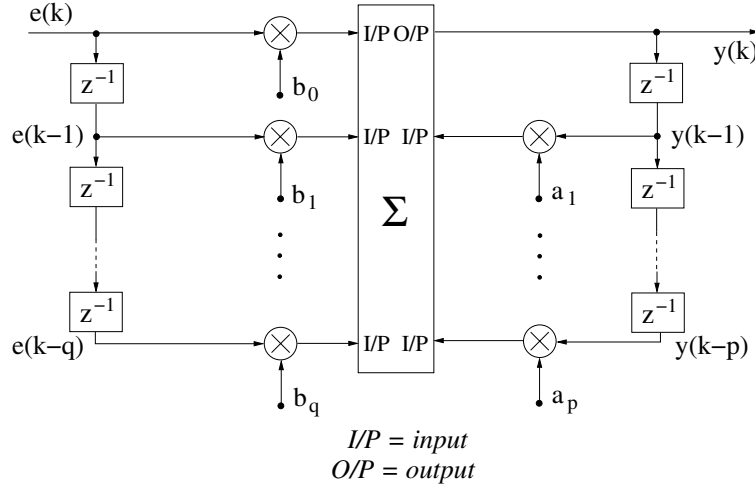


Figure 3.4 Structure of an autoregressive moving average filter (ARMA(p, q))

is simplified to yield an autoregressive (AR(p)) filter

$$y(k) = \sum_{i=1}^p a_i y(k-i) + e(k), \quad (3.12)$$

which is also termed an IIR filter. The filter described by (3.12) is the basis for modelling the speech production process (Makhoul 1975). The presence of feedback within the AR(p) and ARMA(p, q) filters implies that selection of the $a_i, i = 1, 2, \dots, p$, coefficients must be such that the filters are BIBO stable, i.e. a bounded output will result from a bounded input (Oppenheim *et al.* 1999).⁷ The most straightforward way to test stability is to exploit the \mathcal{Z} -domain representation of the transfer function of the filter represented by (3.10):

$$H(z) = \frac{Y(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + \dots + b_q z^{-q}}{1 - a_1 z^{-1} - \dots - a_p z^{-p}} = \frac{N(z)}{D(z)}. \quad (3.13)$$

To guarantee stability, the p roots of the denominator polynomial of $H(z)$, i.e. the values of z for which $D(z) = 0$, the poles of the transfer function, must lie within the unit circle in the z -plane, $|z| < 1$. In digital signal processing, cascade, lattice, parallel and wave filters have been proposed for the realisation of the transfer function described by (3.13) (Oppenheim *et al.* 1999). For prediction applications, however, the direct form, as in Figure 3.4, and lattice structures are most commonly employed.

In signal modelling, rather than being deterministic, the input $e(k)$ to the filter in (3.10) is assumed to be an independent identically distributed (i.i.d.) discrete time random signal. This input is an integral part of a rational transfer function discrete time signal model. The filtering operations described by Equations (3.10)–(3.12),

⁷ This type of stability is commonly denoted as BIBO stability in contrast to other types of stability, such as global asymptotic stability (GAS).

together with such an i.i.d. input with prescribed finite variance σ_e^2 , represent respectively, ARMA(p, q), MA(q) and AR(p) signal models. The autocorrelation function of the input $e(k)$ is given by $\sigma_e^2\delta(k)$ and therefore its power spectral density (PSD) is $P_e(f) = \sigma_e^2$, for all f . The PSD of an ARMA model is therefore

$$P_y(f) = |H(f)|^2 P_e(f) = \sigma_e^2 |H(f)|^2, \quad f \in (-\frac{1}{2}, \frac{1}{2}], \quad (3.14)$$

where f is the normalised frequency. The quantity $|H(f)|^2$ is the magnitude squared frequency domain transfer function found from (3.13) by replacing $z = e^{j2\pi f}$. The role of the filter is therefore to shape the PSD of the driving noise to match the PSD of the physical system. Such an ARMA model is well motivated by the Wold decomposition, which states that any stationary discrete time random signal can be split into the sum of uncorrelated deterministic and random components. In fact, an ARMA(∞, ∞) model is sufficient to model any stationary discrete time random signal (Theiler *et al.* 1993).

3.7 Nonlinear Predictors

If a measurement is assumed to be generated by an ARMA(p, q) model, the optimal conditional mean predictor of the discrete time random signal $\{y(k)\}$

$$\hat{y}(k) = E[y(k) | y(k-1), y(k-2), \dots, y(0)] \quad (3.15)$$

is given by

$$\hat{y}(k) = \sum_{i=1}^p a_i y(k-i) + \sum_{j=1}^q b_j \hat{e}(k-j), \quad (3.16)$$

where the residuals $\hat{e}(k-j) = y(k-j) - \hat{y}(k-j)$, $j = 1, 2, \dots, q$. Notice the predictor described by (3.16) utilises the past values of the actual measurement, $y(k-i)$, $i = 1, 2, \dots, p$; whereas the estimates of the unobservable input signal, $e(k-j)$, $j = 1, 2, \dots, q$, are formed as the difference between the actual measurements and the past predictions. The feedback present within (3.16), which is due to the residuals $\hat{e}(k-j)$, results from the presence of the MA(q) part of the model for $y(k)$ in (3.10). No information is available about $e(k)$ and therefore it cannot form part of the prediction. On this basis, the simplest form of nonlinear autoregressive moving average NARMA(p, q) model takes the form,

$$y(k) = \Theta \left(\sum_{i=1}^p a_i y(k-i) + \sum_{j=1}^q b_j e(k-j) \right) + e(k), \quad (3.17)$$

where $\Theta(\cdot)$ is an unknown differentiable zero memory nonlinear function. Notice $e(k)$ is not included within $\Theta(\cdot)$ as it is unobservable. The term NARMA(p, q) is adopted to define (3.17), since save for the $e(k)$, the output of an ARMA(p, q) model is simply passed through the zero-memory nonlinearity $\Theta(\cdot)$.

The corresponding NARMA(p, q) predictor is given by

$$\hat{y}(k) = \Theta \left(\sum_{i=1}^p a_i y(k-i) + \sum_{j=1}^q b_j \hat{e}(k-j) \right), \quad (3.18)$$

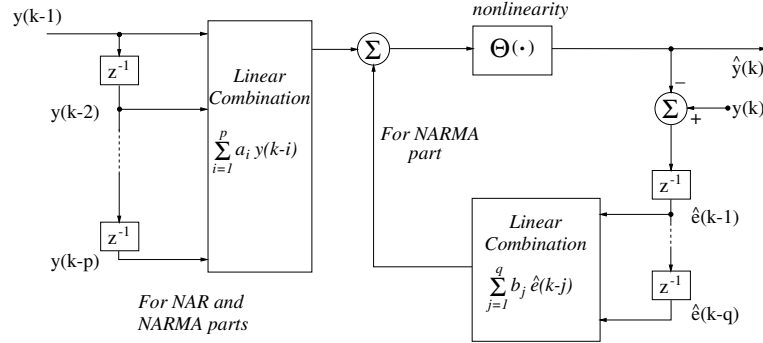


Figure 3.5 Structure of NARMA(p, q) and NAR(p) predictors

where the residuals $\hat{e}(k-j) = y(k-j) - \hat{y}(k-j)$, $j = 1, 2, \dots, q$. Equivalently, the simplest form of nonlinear autoregressive (NAR(p)) model is described by

$$y(k) = \Theta\left(\sum_{i=1}^p a_i y(k-i)\right) + e(k) \quad (3.19)$$

and its associated predictor is

$$\hat{y}(k) = \Theta\left(\sum_{i=1}^p a_i y(k-i)\right). \quad (3.20)$$

The associated structures for the predictors described by (3.18) and (3.20) are shown in Figure 3.5. Feedback is present within the NARMA(p, q) predictor, whereas the NAR(p) predictor is an entirely feedforward structure. The structures are simply those of linear filters described in Section 3.6 with the incorporation of a zero-memory nonlinearity.

In control applications, most generally, NARMA(p, q) models also include so-called exogenous inputs, $u(k-s)$, $s = 1, 2, \dots, r$, and following the approach of (3.17) and (3.19) the simplest example takes the form

$$y(k) = \Theta\left(\sum_{i=1}^p a_i y(k-i) + \sum_{j=1}^q b_j e(k-j) + \sum_{s=1}^r c_s u(k-s)\right) + e(k) \quad (3.21)$$

and is termed a nonlinear autoregressive moving average with exogeneous inputs model, NARMAX(p, q, r), with associated predictor

$$\hat{y}(k) = \Theta\left(\sum_{i=1}^p a_i y(k-i) + \sum_{j=1}^q b_j \hat{e}(k-j) + \sum_{s=1}^r c_s u(k-s)\right), \quad (3.22)$$

which again exploits feedback (Chen and Billings 1989; Siegelmann *et al.* 1997). This is the most straightforward form of nonlinear predictor structure derived from linear filters.

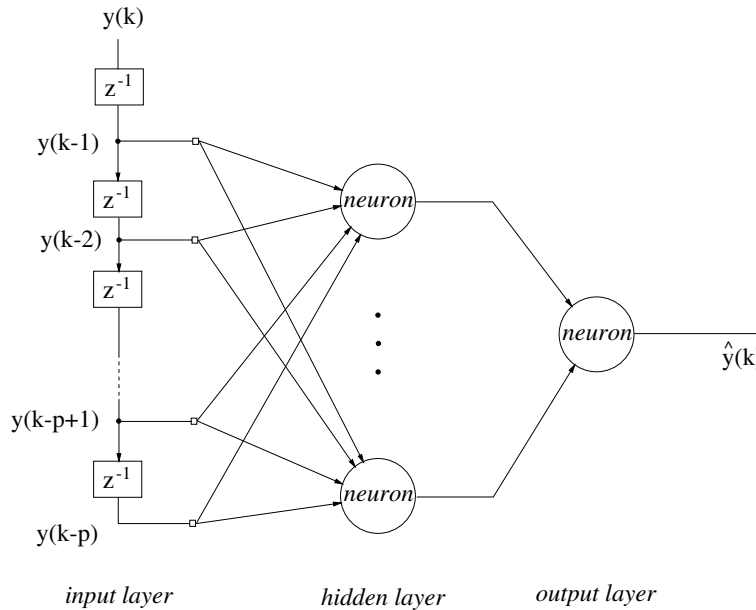


Figure 3.6 Multilayer feedforward neural network

3.8 Feedforward Neural Networks: Memory Aspects

The nonlinearity present in the predictors described by (3.18), (3.20) and (3.22) only appears at the overall output, in the same manner as in the simple neuron depicted in Figure 3.3. These predictors could therefore be referred to as single neuron structures. More generally, however, in neural networks, the nonlinearity is distributed through certain layers, or stages, of processing.

In Figure 3.6 a multilayer feedforward neural network is shown. The measurement samples appear at the input layer, and the output prediction is given from the output layer. To be consistent with the problem of prediction of a single discrete time random signal, only a single output is assumed. In between, there exist so-called hidden layers. Notice the outputs of each layer are only connected to the inputs of the adjacent layer. The nonlinearity inherent in the network is due to the overall action of all the activation functions of the neurons within the structure.

In the problem of prediction, the nature of the inputs to the multilayer feedforward neural network must capture something about the time evolution of the underlying discrete time random signal. The simplest situation is for the inputs to be time-delayed versions of the signal, i.e. $y(k-i)$, $i = 1, 2, \dots, p$, and is commonly termed a tapped delay line or delay space embedding (Mozer 1993). Such a block of inputs provides the network with a short-term memory of the signal. At each time sample, k , the inputs of the network only see the effect of one sample of $y(k)$, and Mozer (1994) terms this a high-resolution memory. The overall predictor can then be represented as

$$\hat{y}(k) = \Phi(y(k-1), y(k-2), \dots, y(k-p)), \quad (3.23)$$

where Φ represents the nonlinear mapping of the neural network.

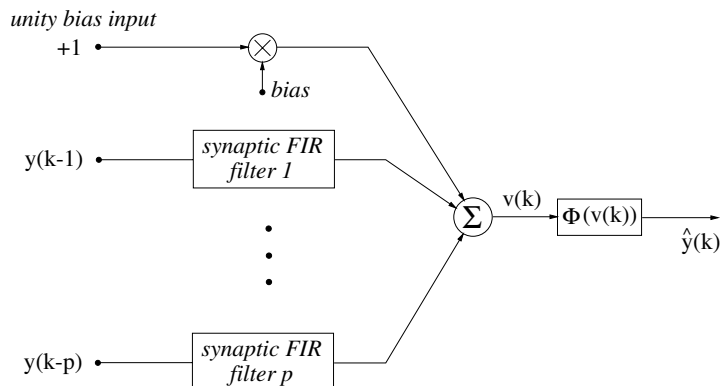


Figure 3.7 Structure of the neuron of a time delay neural network

Other forms of memory for the network include: samples with nonuniform delays, i.e. $y(k-i)$, $i = \tau_1, \tau_2, \dots, \tau_p$; exponential, where each input to the network, denoted $\tilde{y}_i(k)$, $i = 1, 2, \dots, p$, is calculated recursively from $\tilde{y}_i(k) = \mu_i \tilde{y}_i(k-1) + (1 - \mu_i) y_i(k)$, where $\mu_i \in [-1, 1]$ is the exponential factor which controls the depth (Mozier 1993) or time spread of the memory and $y_i(k) = y(k-i)$, $i = 1, 2, \dots, p$. A delay line memory is therefore termed high-resolution low-depth, while an exponential memory is low-resolution but high-depth. In continuous time, Principe *et al.* (1993) proposed the Gamma memory, which provided a method to trade resolution for depth. A discrete time version of this memory is described by

$$\tilde{y}_{\mu,j}(k) = \mu \tilde{y}_{\mu,j}(k-1) + (1 - \mu) \tilde{y}_{\mu,j-1}(k-1), \quad (3.24)$$

where the index j is included because it is necessary to evaluate (3.24) for $j = 0, 1, \dots, i$, where i is the delay of the particular input to the network and $\tilde{y}_{\mu,-1}(k) = y(k+1)$, for all $k \geq 0$, and $\tilde{y}_{\mu,j}(0) = 0$, for all $j \geq 0$. The form of the equation is, moreover, a convex mixture. The choice of μ controls the trade-off between depth and resolution; small μ provides low-depth and high-resolution memory, whereas high μ yields high-depth and low-resolution memory.

Restricting the memory in a multilayer feedforward neural network to the input layer may, however, lead to structures with an excessively large number of parameters. Wan (1993) therefore utilises a time-delay network where the memory is integrated within each layer of the network. Figure 3.7 shows the form of a neuron within a time-delay network, in which the multipliers of the basic neuron of Figure 3.3 are replaced by FIR filters to capture the dynamics of the input signals. Networks formed from such neurons are functionally equivalent to networks with only the memory at their input but generally have many fewer parameters, which is beneficial for learning algorithms.

The integration of memory into a multilayer feedforward network yields the structure for nonlinear prediction. It is clear, therefore, that such networks belong to the class of nonlinear filters.

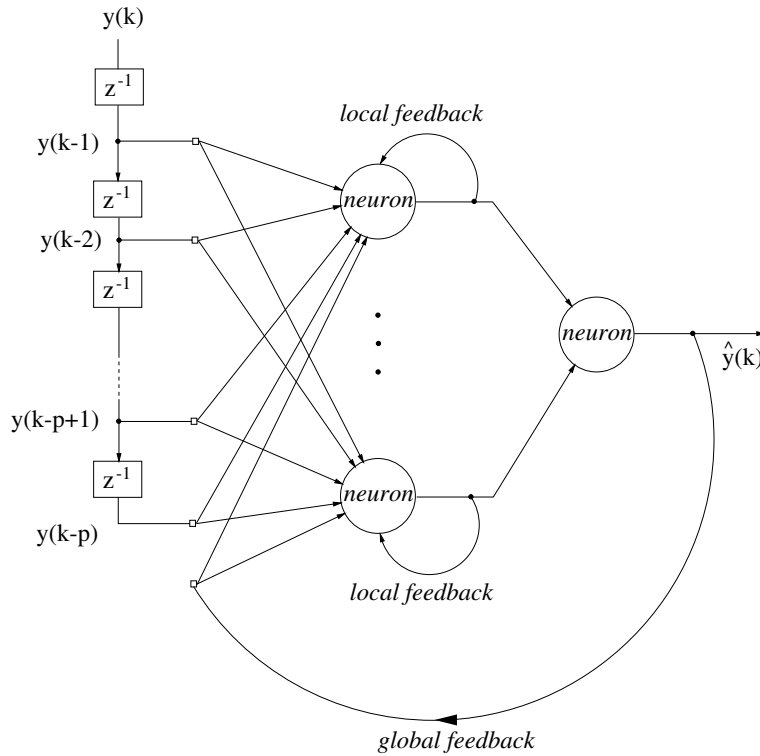


Figure 3.8 Structure of a recurrent neural network with local and global feedback

3.9 Recurrent Neural Networks: Local and Global Feedback

In Figure 3.6, the inputs to the network are drawn from the discrete time signal $y(k)$. Conceptually, it is straightforward to consider connecting the delayed versions of the output, $\hat{y}(k)$, of the network to its input. Such connections, however, introduce feedback into the network and therefore the stability of such networks must be considered, this is a particular focus of later parts of this book. The provision of feedback, with delay, introduces memory to the network and so is appropriate for prediction.

The feedback within recurrent neural networks can be achieved in either a local or global manner. An example of a recurrent neural network is shown in Figure 3.8 with connections for both local and global feedback. The local feedback is achieved by the introduction of feedback within the hidden layer, whereas the global feedback is produced by the connection of the network output to the network input. Inter-neuron connections can also exist in the hidden layer, but they are not shown in Figure 3.8. Although explicit delays are not shown in the feedback connections, they are assumed to be present within the neurons in order that the network is realisable. The operation of a recurrent neural network predictor that employs global feedback can now be represented by

$$\hat{y}(k) = \Phi(y(k-1), y(k-2), \dots, y(k-p), \hat{e}(k-1), \dots, \hat{e}(k-q)), \quad (3.25)$$

where again $\Phi(\cdot)$ represents the nonlinear mapping of the neural network and

$$\hat{e}(k-j) = y(k-j) - \hat{y}(k-j), \quad j = 1, \dots, q.$$

A taxonomy of recurrent neural networks architectures is presented by Tsoi and Back (1997). The choice of structure depends upon the dynamics of the signal, learning algorithm and ultimately the prediction performance. There is, unfortunately, no hard and fast rule as to the best structure to use for a particular problem (Personnaz and Dreyfus 1998).

3.10 State-Space Representation and Canonical Form

The structures in this chapter have been developed on the basis of difference equation representations. Simple nonlinear predictors can be formed by placing a zero-memory nonlinearity within the output stage of a classical linear predictor. In this case, the nonlinearity is restricted to the output stage, as in a single layer neural network realisation. On the other hand, if the nonlinearity is distributed through many layers of weighted interconnections, the concept of neural networks is fully exploited and more powerful nonlinear predictors may ensue. For the purpose of prediction, memory stages may be introduced at the input or within the network. The most powerful approach is to introduce feedback and to unify feedback networks. Nerrand *et al.* (1994) proposed an insightful canonical state-space representation:

Any feedback network can be cast into a canonical form that consists of a feedforward (static) network:

whose outputs are the outputs of the neurons that have desired values, and the values of the state variables,

whose inputs are the inputs of the network and the values of the state variables, the latter being delayed by one time unit.

Note that in the prediction of a single discrete-time random signal, the network will have only one output neuron with a predicted value. For a dynamic system, such as a recurrent neural network for prediction, the state represents *a set of quantities that summarizes all the information about the past behaviour of the system that is needed to uniquely describe its future behaviour, except for the purely external effects arising from the applied input (excitation)* (Haykin 1999b).

It should be noted that, whereas it is always possible to rewrite a nonlinear input-output model in a state-space representation, an input-output model equivalent to a given state-space model might not exist and, if it does, it is surely of higher order. Under fairly general conditions of observability of a system, however, an equivalent input-output model does exist but it may be of high order. A state-space model is likely to have lower order and require a smaller number of past inputs and, hopefully, a smaller number of parameters. This has fundamental importance when only a limited number of data samples is available. Takens' theorem (Wan 1993) implies that for a wide class of deterministic systems, there exists a diffeomorphism (one-to-one differential mapping) between a finite window of the time series and the underlying

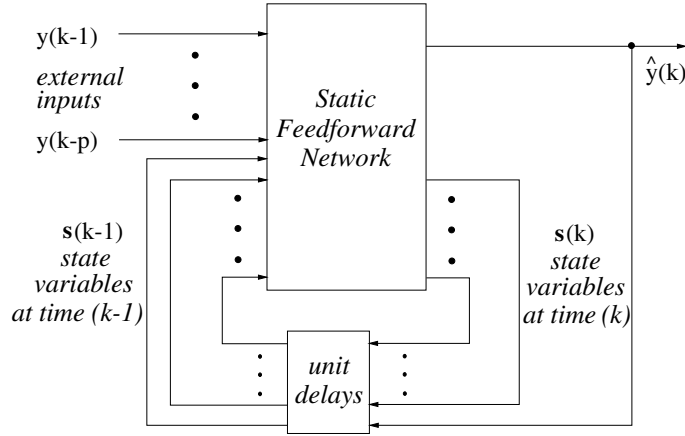


Figure 3.9 Canonical form of a recurrent neural network for prediction

state of the dynamic system which gives rise to the time series. A neural network can therefore approximate this mapping to realise a predictor.

In Figure 3.9, the general canonical form of a recurrent neural network is represented. If the state is assumed to contain N variables, then a state vector is defined as $\mathbf{s}(k) = [s_1(k), s_2(k), \dots, s_N(k)]^T$, and a vector of p external inputs is given by $\mathbf{y}(k-1) = [y(k-1), y(k-2), \dots, y(k-p)]^T$. The state evolution and output equations of the recurrent network for prediction are given, respectively, by

$$\mathbf{s}(k) = \varphi(\mathbf{s}(k-1), \mathbf{y}(k-1), \hat{y}(k-1)), \quad (3.26)$$

$$\hat{y}(k) = \psi(\mathbf{s}(k-1), \mathbf{y}(k-1), \hat{y}(k-1)), \quad (3.27)$$

where φ and ψ represent general classes of nonlinearities. The particular choice of N minimal state variables is not unique, therefore several canonical forms⁸ exist. A procedure for the determination of N for an arbitrary recurrent neural network is described by Nerrand *et al.* (1994). The NARMA and NAR predictors described by (3.18) and (3.20), however, follow naturally from the canonical state-space representation because the elements of the state vector are calculated from the inputs and outputs of the network. Moreover, even if the recurrent neural network contains local feedback and memory, it is still possible to convert the network into the above canonical form (Personnaz and Dreyfus 1998).

3.11 Summary

The aim of this chapter has been to show the commonality between the structures of nonlinear filters and neural networks. To this end, the basic building blocks for both structures have been shown to be adders, delayers, multipliers and zero-memory nonlinearities, and the manner in which these elements are interconnected defines

⁸ These canonical forms stem from Jordan canonical forms of matrices and companion matrices. Notice that in fact $\hat{y}(k)$ is a state variable but shown separately to emphasise its role as the predicted output.

the particular structure. The theory of linear predictors, for stationary discrete time random signals, which are optimal in the minimum mean square prediction error sense, has been shown to be well established. The structures of linear predictors have also been demonstrated to be established in signal processing and statistical modelling. Nonlinear predictors have then been developed on the basis of defining the dynamics of a discrete time random signal by a nonlinear model. In essence, in their simplest form these predictors have two stages: a weighted linear combination of inputs and/or past outputs, as for linear predictors, and a second stage defined by a zero-memory nonlinearity.

The neuron, the fundamental processing element in neural networks, has been introduced. Multilayer feedforward neural networks have been introduced in which the nonlinearity is distributed throughout the structure. To operate in a prediction mode, some local memory is required either at the input or integral to the network structure. Recurrent neural networks have then been formulated by connecting delayed versions of the global output to the input of a multilayer feedforward structure; or by the introduction of local feedback within the network. A canonical state-space form has been used to represent an arbitrary neural network.

4

Activation Functions Used in Neural Networks

4.1 Perspective

The choice of nonlinear activation function has a key influence on the complexity and performance of artificial neural networks, note the term *neural network* will be used interchangeably with the term *artificial neural network*. The brief introduction to activation functions given in Chapter 3 is therefore extended. Although sigmoidal nonlinear activation functions are the most common choice, there is no strong *a priori* justification why models based on such functions should be preferred to others.

We therefore introduce neural networks as universal approximators of functions and trajectories, based upon the Kolmogorov universal approximation theorem, which is valid for both feedforward and recurrent neural networks. From these universal approximation properties, we then demonstrate the need for a sigmoidal activation function within a neuron. To reduce computational complexity, approximations to sigmoid functions are further discussed. The use of nonlinear activation functions suitable for hardware realisation of neural networks is also considered.

For rigour, we extend the analysis to complex activation functions and recognise that a suitable complex activation function is a Möbius transformation. In that context, a framework for rigorous analysis of some inherent properties of neural networks, such as fixed points, nesting and invertibility based upon the theory of modular groups of Möbius transformations is provided.

All the relevant definitions, theorems and other mathematical terms are given in Appendix B and Appendix C.

4.2 Introduction

A century ago, a set of 23 (originally) unsolved problems in mathematics was proposed by David Hilbert (Hilbert 1901–1902). In his lecture ‘Mathematische Probleme’ at the second International Congress of Mathematics held in Paris in 1900, he presented 10 of them. These problems were designed to serve as examples for the kinds of problems whose solutions would lead to further development of disciplines in mathematics.

His 13th problem concerned solutions of polynomial equations. Although his original formulation dealt with properties of the solution of the seventh degree algebraic equation,¹ this problem can be restated as: *Prove that there are continuous functions of n variables, not representable by a superposition of continuous functions of $(n - 1)$ variables.* In other words, could a general algebraic equation of a high degree be expressed by sums and compositions of single-variable functions?² In 1957, Kolmogorov showed that the conjecture of Hilbert was not correct (Kolmogorov 1957).

Kolmogorov's theorem is a general representation theorem stating that any real-valued continuous function f defined on an n -dimensional cube I^n ($n \geq 2$) can be represented as

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \psi_{pq}(x_p) \right), \quad (4.1)$$

where $\Phi_q(\cdot)$, $q = 1, \dots, 2n + 1$, and $\psi_{pq}(\cdot)$, $p = 1, \dots, n$, $q = 1, \dots, 2n + 1$, are typically nonlinear continuous functions of one variable.

For a neural network representation, this means that an activation function of a neuron has to be nonlinear to form a universal approximator. This also means that every continuous function of many variables can be represented by a four-layered neural network with two hidden layers and an input and output layer, whose hidden units represent mappings Φ and ψ . However, this does not mean that a network with two hidden layers necessarily provides an accurate representation of function f . In fact, functions ψ_{pq} of Kolmogorov's theorem are quite often highly nonsmooth, whereas for a neural network we want smooth nonlinear activation functions, as is required by gradient-descent learning algorithms (Poggio and Girosi 1990). Vitushkin (1954) showed that there are functions of more than one variable which do not have a representation by superpositions of differentiable functions (Beiu 1998). Important questions about Kolmogorov's representation are therefore existence, constructive proofs and bounds on the size of a network needed for approximation.

Kolmogorov's representation has been improved by several authors. Sprecher (1965) replaced functions ψ_{pq} in the Kolmogorov representation by $\lambda^{pq}\psi_q$, where λ is a constant and ψ_q are monotonic increasing functions which belong to the class of Lipschitz functions. Lorentz (1976) showed that the functions Φ_q can be replaced by only one function Φ . Hecht-Nielsen reformulated this result for MLPs so that they are able to approximate any function. In this case, functions ψ are nonlinear activation functions in hidden layers, whereas functions Φ are nonlinear activation functions in the output layer. The functions Φ and ψ are found, however, to be generally highly nonsmooth. Further, in Katsuura and Sprecher (1994), the function ψ is obtained through a graph that is the limit point of an iterated composition of contraction mappings on their domain.

In applications of neural networks for universal approximation, the existence proof for approximation by neural networks is provided by Kolmogorov's theorem, which

¹ Hilbert conjectured that the roots of the equation $x^7 + ax^3 + bx^2 + cx + 1 = 0$ as functions of coefficients a, b, c are not representable by sums and superpositions of functions of two coefficients, or 'Show the impossibility of solving the general seventh degree equation by functions of two variables.'

² For example, function xy is a composition of functions $g(\cdot) = \exp(\cdot)$ and $h(\cdot) = \log(\cdot)$, therefore $xy = e^{\log(x) + \log(y)} = g(h(x) + h(y))$ (Gorban and Wunsch 1998).

in the neural network community was first recognised by Hecht-Nielsen (1987) and Lippmann (1987). The first constructive proof of neural networks as universal approximators was given by Cybenko (1989). Most of the analyses rest on the denseness property of nonlinear functions that approximate the desired function in the space in which the desired function is defined. In Cybenko's results, for instance, if σ is a continuous discriminatory function,³ then finite sums of the form,

$$g(\mathbf{x}) = \sum_{i=1}^N w_i \sigma(\mathbf{a}_i^T \mathbf{x} + b_i), \quad (4.2)$$

where $w_i, b_i, i = 1, \dots, N$, are coefficients, are dense in the space of continuous functions defined on $[0, 1]^n$. Following the classical approach to approximation, this means that given any continuous function f defined on $[0, 1]^N$ and any $\varepsilon > 0$, there is a $g(\mathbf{x})$ given by (4.2) for which $|g(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$ for all $\mathbf{x} \in [0, 1]^N$. Cybenko then concludes that any bounded and measurable sigmoidal function is discriminatory (Cybenko 1989), and that a three-layer neural network with a sufficient number of neurons in its hidden layer can represent an arbitrary function (Beiu 1998; Cybenko 1989).

Funahashi (1989) extended this to include sigmoidal functions so that any continuous function is approximately realisable by three-layer networks with bounded and monotonically increasing activation functions within hidden units. Hornik *et al.* (1989) showed that the output function does not have to be continuous, and they also proved that a neural network can approximate simultaneously both a function and its derivative (Hornik *et al.* 1990). Hornik (1990) further showed that the activation function has to be bounded and nonconstant (but not necessarily continuous), Kurkova (1992) revealed the existence of an approximate representation of functions by superposition of nonlinear functions within the constraints of neural networks. Leshno *et al.* (1993) relaxed the condition for the activation function to be 'locally bounded piecewise continuous' (i.e. if and only if the activation function is not a polynomial). This result encompasses most of the activation functions commonly used.

Funahashi and Nakamura (1993), in their article 'Approximation of dynamical systems by continuous time recurrent neural networks', proved that the universal approximation theorem also holds for trajectories and patterns and for recurrent neural networks. Li (1992) also showed that recurrent neural networks are universal approximators. Some recent results, moreover, suggest that 'smaller nets perform better' (Elsken 1999), which recommends recurrent neural networks, since a small-scale RNN has dynamics that can be achieved only by a large scale feedforward neural network.

³ $\sigma(\cdot)$ is discriminatory if for a Borel measure μ on $[0, 1]^N$,

$$\int_{[0,1]^N} \sigma(\mathbf{a}^T \mathbf{x} + b) d\mu(\mathbf{x}) = 0, \quad \forall \mathbf{a} \in \mathbb{R}^N, \quad \forall b \in \mathbb{R},$$

implies that $\mu = 0$. The sigmoids Cybenko considered had limits

$$\sigma(t) = \begin{cases} 0, & t \rightarrow -\infty, \\ 1, & t \rightarrow \infty. \end{cases}$$

This justifies the use of the logistic function $\sigma(x) = 1/(1 + e^{-\beta x})$ in neural network applications.

Sprecher (1993) considered the problem of dimensionality of neural networks and demonstrated that the number of hidden layers is independent of the number of input variables N . Barron (1993) described spaces of functions that can be approximated by the relaxed algorithm of Jones using functions computed by single-hidden-layer networks or perceptrons. Attali and Pages (1997) provided an approach based upon the Taylor series expansion. Maiorov and Pinkus have given lower bounds for neural network based approximation (Maiorov and Pinkus 1999). Approximation ability of neural networks has also been rigorously studied in Williamson and Helmke (1995).

Sigmoid neural units usually use a ‘bias’ or ‘threshold’ term in computing the activation potential (combination function, net input $\text{net}(k) = \mathbf{x}^T(k)\mathbf{w}(k)$) of the neural unit. The bias term is a connection weight from a unit with a constant value as shown in Figure 3.3. The bias unit is connected to every neuron in a neural network, the weight of which can be trained just like any other weight in a neural network.

From the geometric point of view, for an MLP with N output units, the operation of the network can be seen as defining an N -dimensional hypersurface in the space spanned by the inputs to the network. The weights define the position of this surface. Without a bias term, all the hypersurfaces would pass through the origin (Mandic and Chambers 2000c), which in turn means that the universal approximation property of neural networks would not hold if the bias was omitted.

A result by Hornik (1993) shows that a sufficient condition for the universal approximation property without biases is that no derivative of the activation function vanishes at the origin, which implies that a fixed nonzero bias can be used instead of a trainable bias.

Why use activation functions?

To introduce nonlinearity into a neural network, we employ nonlinear activation (output) functions. Without nonlinearity, since a composition of linear functions is again a linear function, an MLP would not be functionally different from a linear filter and would not be able to perform nonlinear separation and trajectory learning for nonlinear and nonstationary signals.

Due to the Kolmogorov theorem, almost any nonlinear function is a suitable candidate for an activation function of a neuron. However, for gradient-descent learning algorithms, this function ought to be differentiable. It also helps if the function is bounded.⁴ For the output neuron, one should either use an activation function suited to the distribution of desired (target) values, or preprocess the inputs to achieve this goal. If, for instance, the desired values are positive but have no known upper bound, an exponential nonlinear activation function can be used.

It is important to identify classes of functions and processes that can be approximated by artificial neural networks. Similar problems occur in nonlinear circuit theory, where analogue nonlinear devices are used to synthesise desired transfer functions (gyrators, impedance converters), and in digital signal processing where digital filters

⁴ The function $f(x) = e^x$ is a suitable candidate for an activation function and is suitable for unbounded signals. It is also continuously differentiable. However, to control the dynamics, fixed points and invertibility of a neural network, it is desirable to have bounded, ‘squashing’ activation functions for neurons.

are designed to approximate arbitrarily well any transfer function. Fuzzy sets are also universal approximators of functions and their derivatives (Kreinovich *et al.* 2000; Mitaim and Kosko 1996, 1997).

4.3 Overview

We first explain the requirements of an activation function mathematically. We will then introduce different types of nonlinear activation functions and discuss their properties and realisability. Finally, a complex form of activation functions within the framework of Möbius transformations will be introduced.

4.4 Neural Networks and Universal Approximation

Learning an input–output relationship from examples using a neural network can be considered as the problem of approximating an unknown function $f(x)$ from a set of data points (Girosi and Poggio 1989a). This is why the analysis of neural networks for approximation is important for neural networks for prediction, and also system identification and trajectory tracking. The property of uniform approximation is also found in algebraic and trigonometric polynomials, such as in the case of Weierstrass and Fourier representation, respectively.

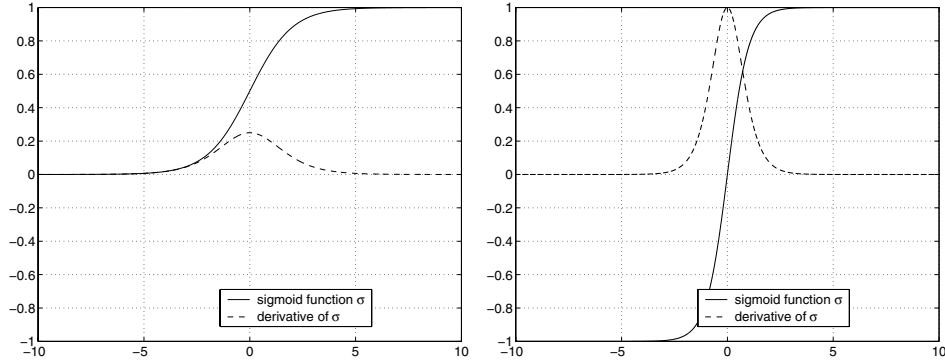
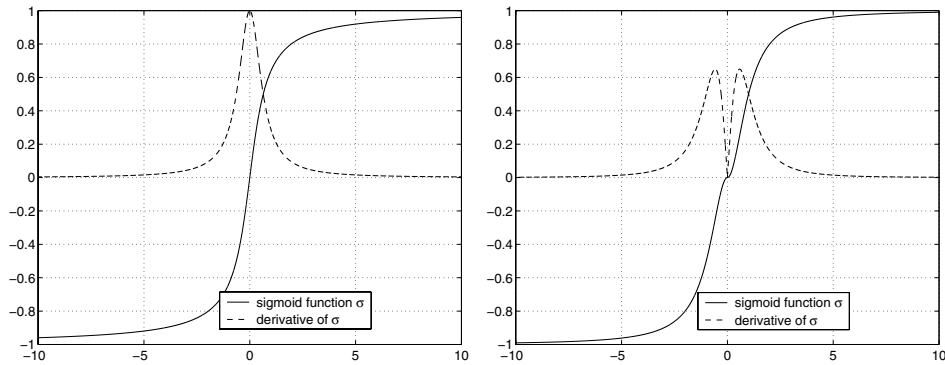
A neural activation function $\sigma(\cdot)$ is typically chosen to be a continuous and differentiable nonlinear function that belongs to the class $S = \{\sigma_i \mid i = 1, 2, \dots, n\}$ of sigmoid⁵ functions having the following desirable properties⁶

- (i) $\sigma_i \in S$ for $i = 1, \dots, n$;
- (ii) $\sigma_i(x_i)$ is a continuously differentiable function;
- (iii) $\sigma'_i(x_i) = \frac{d\sigma_i(x_i)}{dx_i} > 0$ for all $x_i \in \mathbb{R}$;
- (iv) $\sigma_i(\mathbb{R}) = (a_i, b_i)$, $a_i, b_i \in \mathbb{R}$, $a_i \neq b_i$;
- (v) $\sigma'_i(x) \rightarrow 0$ as $x \rightarrow \pm\infty$;
- (vi) $\sigma'_i(x)$ takes a global maximal value $\max_{x \in \mathbb{R}} \sigma'_i(x)$ at a unique point $x = 0$;
- (vii) a sigmoidal function has only one inflection point, preferably at $x = 0$;
- (viii) from (iii), function σ_i is monotonically nondecreasing, i.e. if $x_1 < x_2$ for each $x_1, x_2 \in \mathbb{R} \Rightarrow \sigma_i(x_1) \leq \sigma_i(x_2)$;
- (ix) σ_i is uniformly Lipschitz, i.e. there exists a constant $L > 0$ such that $\|\sigma_i(x_1) - \sigma_i(x_2)\| \leq L\|x_1 - x_2\|$, $\forall x_1, x_2 \in \mathbb{R}$, or in other words

$$\frac{\sigma_i(x_1) - \sigma_i(x_2)}{x_1 - x_2} \leq L, \quad \forall x_1, x_2 \in \mathbb{R}, \quad x_1 \neq x_2.$$

⁵ Sigmoid means *S*-shaped.

⁶ The constraints we impose on sigmoidal functions are stricter than the ones commonly employed.

(a) Sigmoid function σ_1 and its derivative(b) Sigmoid function σ_2 and its derivative(c) Sigmoid function σ_3 and its derivative(d) Sigmoid function σ_4 and its derivative**Figure 4.1** Sigmoid functions and their derivatives

We will briefly discuss some of the above requirements. Property (ii) represents continuous differentiability of a sigmoid function, which is important for higher order learning algorithms, which require not only existence of the Jacobian matrix, but also the existence of a Hessian and matrices containing higher-order derivatives. This is also necessary if the behaviour of a neural network is to be described via Taylor series expansion about the current point in the state space of the network. Property (iii) states that a sigmoid should have a positive first derivative, which in turn means that a gradient descent algorithm which is employed for training of a neural network should have gradient vectors pointing towards the bottom of the bowl shaped error performance surface, which is the global minimum of the surface. Property (vi) means that the point around which the first derivative is centred is the origin. This is connected with property (vii) which means that the second derivative of the activation function should change its sign at the origin. Going back to the error performance surface, this

means that irrespective of whether the current prediction error is positive or negative, the gradient vector of the network at that point should point downwards. Monotonicity, required by (viii) is useful for uniform convergence of algorithms and in search for fixed points of neural networks. Finally, the Lipschitz condition is connected with the boundedness of an activation function and degenerates into requirements of uniform convergence given by the contraction mapping theorem for $L < 1$.

Surveys of neural transfer functions can be found in Duch and Jankowski (1999) and Cichocki and Unbehauen (1993). Examples of sigmoidal functions are

$$\left. \begin{aligned} \sigma_1(x) &= \frac{1}{1 + e^{-\beta x}}, & \beta \in \mathbb{R}, \\ \sigma_2(x) &= \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}, & \beta \in \mathbb{R}, \\ \sigma_3(x) &= \frac{2}{\pi} \arctan(\frac{1}{2}\pi\beta x), & \beta \in \mathbb{R}, \\ \sigma_4(x) &= \frac{x^2}{1 + x^2} \operatorname{sgn}(x), \end{aligned} \right\} \quad (4.3)$$

where $\sigma(x) = \Phi(x)$ as in Chapter 3. For $\beta = 1$, these functions and their derivatives are given in Figure 4.1. The function σ_1 , also known as the logistic function,⁷ is unipolar, whereas the other three activation functions are bipolar. Two frequently used sigmoid functions in neural networks are σ_1 and σ_2 . Their derivatives are also simple to calculate, and are

$$\left. \begin{aligned} \sigma_1'(x) &= \beta\sigma_1(x)(1 - \sigma_1(x)), \\ \sigma_2'(x) &= \beta \operatorname{sech}^2(x) = \beta(1 - \sigma_2^2(x)). \end{aligned} \right\} \quad (4.4)$$

We can easily modify activation functions to have different saturation values. For the logistic function $\sigma_1(x)$, whose saturation values are $(0, 1)$, to obtain saturation values $(-1, 1)$, we perform

$$\sigma_s(x) = \frac{2}{1 + e^{-\beta x}} - 1. \quad (4.5)$$

To modify the input data to fall within the range of an activation function, we can normalise, standardise or rescale the input data, using mean μ , standard deviation std and the minimum and maximum range R_{\min} and R_{\max} .⁸ Cybenko (1989) has shown that neural networks with a single hidden layer of neurons with sigmoidal functions are

⁷ The logistic map $\dot{f} = rf(1 - f/K)$ (Strogatz 1994) is used to describe population dynamics, where f is the growth of a population of organisms, r denotes the growth rate and K is the so-called carrying capacity (population cannot grow unbounded). Fixed points of this map in the phase space are 0 and K , hence the population always approaches the carrying capacity. Under these conditions, the graph of $f(t)$ belongs to the class of sigmoid functions.

⁸ To normalise the input data to $\mu = 0$ and $\operatorname{std} = 1$, we calculate

$$\mu = \frac{\sum_{i=1}^N x_i}{N}, \quad \operatorname{std} = \sqrt{\frac{\sum_{i=1}^N (x_i - \mu)^2}{N}},$$

and perform the standardisation of the input data as $\bar{x}_i = (x_i - \mu)/\operatorname{std}$. To translate data to midrange

universal approximators and provided they have enough neurons, can approximate an arbitrary continuous function on a compact set with arbitrary precision. These results do not mean that sigmoidal functions always provide an optimal choice.⁹

Two functions determine the way signals are processed by neurons.

Combination functions. Each processing unit in a neural network performs some mathematical operation on values that are fed into it via synaptic connections (weights) from other units. The resulting value is called the activation potential or ‘net input’. This operation is known as a ‘combination function’, ‘activation function’ or ‘net input’. Any combination function is a net: $\mathbb{R}^N \rightarrow \mathbb{R}$ function, and its output is a scalar. Most frequently used combination functions are inner product (linear) combination functions (as in MLPs and RNNs) and Euclidean or Mahalanobis distance combination functions (as in RBF networks).

Activation functions. Neural networks for nonlinear processing of signals map their net input provided by a combination function onto the output of a neuron using a scalar function called a ‘nonlinear activation function’, ‘output function’ or sometimes even ‘activation function’. The entire functional mapping performed by a neuron (composition of a combination function and a nonlinear activation function) is sometimes called a ‘transfer’ function of a neuron $\sigma : \mathbb{R}^N \rightarrow \mathbb{R}$. Nonlinear activation functions with a bounded range are often called ‘squashing’ functions, such as the commonly used tanh and logistic functions. If a unit does not transform its net input, it is said to have an ‘identity’ or ‘linear’ activation function.¹⁰

Distance based combination functions (proximity functions) $D(\mathbf{x}; \mathbf{t}) \propto \|\mathbf{x} - \mathbf{t}\|$, are used to calculate how close \mathbf{x} is to a prototype vector \mathbf{t} . It is also possible to use some combination of the inner product and distance activation functions, for instance in the form $\alpha \mathbf{w}^T \mathbf{x} + \beta \|\mathbf{x} - \mathbf{t}\|$ (Duch and Jankowski 1999). Many other functions can be used to calculate the net input, as for instance

$$A(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^N w_i x_i + w_{N+1} \sum_{i=1}^N x_i^2$$

(Ridella *et al.* 1997).

4.5 Other Activation Functions

By the universal approximation theorems, there are many choices of the nonlinear activation function. Therefore, in this section we describe some commonly used application-motivated activation functions of a neuron.

0 and standardise to range R , we perform

$$Z = \frac{\max_i \{x_i\} + \min_i \{x_i\}}{R}, \quad S_x = \max_i \{x_i\} - \min_i \{x_i\}, \quad x_i^n = \frac{x_i - Z}{S_x/R}.$$

⁹ Rational transfer functions (Leung and Haykin 1993) and Gaussian transfer functions also allow NNs to implement universal approximators.

¹⁰ <http://www.informatik.uni-freiburg.de/~heinz/faq.html>

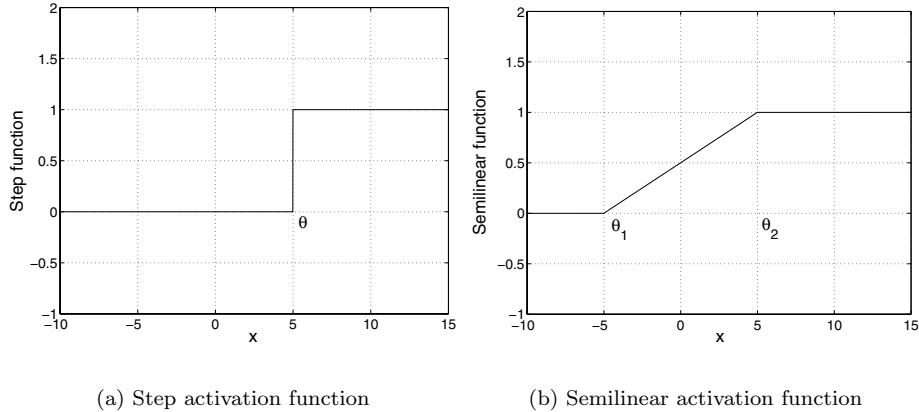


Figure 4.2 Step and semilinear activation function

The hard-limiter Heaviside (step) function was frequently used in the first implementations of neural networks, due to its simplicity. It is given by

$$H(x) = \begin{cases} 0, & x \leq \theta, \\ 1, & x > \theta, \end{cases} \quad (4.6)$$

where θ is some threshold. A natural extension of the step function is the multistep function $H_{MS}(x; \theta) = y_i$, $\theta_i \leq x \leq \theta_{i+1}$. A variant of this function resembles a staircase $\theta_1 < \theta_2 < \dots < \theta_N \Leftrightarrow y_1 < y_2 < \dots < y_N$, and is often called the staircase function. The semilinear function is defined as

$$H_{SL}(x; \theta_1, \theta_2) = \begin{cases} 0, & x \leq \theta_1, \\ (x - \theta_1)/(\theta_2 - \theta_1), & \theta_1 < x \leq \theta_2, \\ 1, & x > \theta_2. \end{cases} \quad (4.7)$$

The functions (4.6) and (4.7) are depicted in Figure 4.2. Both the above mentioned functions have discontinuous derivatives, preventing the use of gradient-based training procedures. Although they are, strictly speaking, *S*-shaped, we do not use them for neural networks for real-time processing, and this is why we restricted ourselves to differentiable functions in our nine requirements that a suitable activation function should satisfy. With the development of neural network theory, these discontinuous functions were later generalised to logistic functions, leading to the *graded response neurons*, which are suitable for gradient-based training. Indeed, the logistic function

$$\sigma(x) = \frac{1}{1 + e^{-\beta x}} \quad (4.8)$$

degenerates into the step function (4.6), as $\beta \rightarrow \infty$.

Many other activation functions have been designed for special purposes. For instance, a modified activation function which enables single layer perceptrons to solve

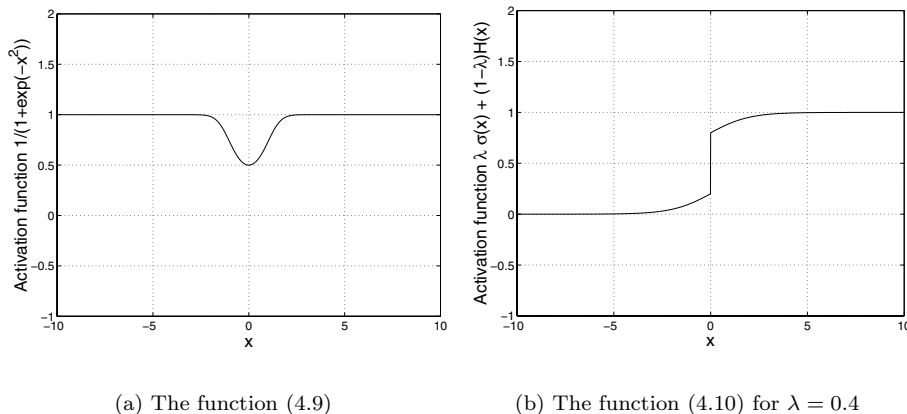


Figure 4.3 Other activation functions

some linearly inseparable problems has been proposed in Zhang and Sarhadi (1993) and takes the form,

$$f(x) = \frac{1}{1 + e^{-(x^2 + \text{bias})}}. \quad (4.9)$$

The function (4.9) is differentiable and therefore a network based upon this function can be trained using gradient descent methods. The square operation in the exponential term of the function enables individual neurons to perform limited nonlinear classification. This activation function has been employed for image segmentation (Zhang and Sarhadi 1993). There have been efforts to combine two or more forms of commonly used functions to obtain an improved activation function. For instance, a function defined by

$$f(x) = \lambda\sigma(x) + (1 - \lambda)H(x), \quad (4.10)$$

where $\sigma(x)$ is a sigmoid function, $H(x)$ is a hard-limiting function and $0 \leq \lambda \leq 1$, has been used in Jones (1990). The function (4.10) is a weighted sum of functions σ and H . The functions (4.9) and (4.10) are depicted in Figure 4.3.

Another possibility is to use a linear combination of sigmoid functions instead of a single sigmoid function as an activation function of a neuron. A sigmoid packet f is therefore defined as a linear combination of a set of sigmoid functions with different amplitudes h , slopes β and biases b (Peng *et al.* 1998). This function is defined as

$$f(x) = \sum_{n=1}^N h_n \sigma_n = \sum_{n=1}^N \frac{h_n}{1 + e^{-\beta_n x + b_n}}. \quad (4.11)$$

During the learning phase, all parameters (h, β, b) can be adjusted for adaptive shape-refining. Intuitively, a Gaussian-shaped activation function can be, for instance, approximated by a difference of two sigmoids, as shown in Figure 4.4. Other options include spline neural networks¹¹ (Guarnieri *et al.* 1999; Vecchi *et al.* 1997) and wavelet

¹¹ Splines are piecewise polynomials (often cubic) that are smooth and can retain the ‘squashing property’.

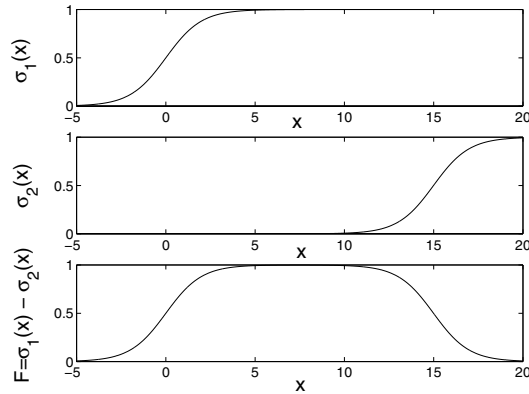


Figure 4.4 Approximation capabilities of sigmoid functions

based neural networks (Zhang *et al.* 1995), where the structure of the network is similar to the RBF, except that the RBFs are replaced by orthonormal scaling functions that are not necessarily radial-symmetric.

For neural systems operating on chaotic input signals, the most commonly used activation function is a sinusoidal function. Another activation function that is often used in order to detect chaos in the input signal is the so-called saturated-modulus function given by (Dogaru *et al.* 1996; Nakagawa 1996)

$$\varphi(x) = \begin{cases} |x|, & |x| \leq 1, \\ 1, & |x| > 1. \end{cases} \quad (4.12)$$

This activation function ensures chaotic behaviour even for a very small number of neurons within the network. This function corresponds to the rectifying operation used in electronic instrumentation and is therefore called a saturated modulus or saturated rectifier function.

4.6 Implementation Driven Choice of Activation Functions

When neurons of a neural network are realised in hardware, due to the limitation of processing power and available precision, activation functions can be significantly different from their ideal forms (Al-Ruwaihi 1997; Yang *et al.* 1998). Implementations of nonlinear activation functions of neurons proposed by various authors are based on a look-up table, McLaurin polynomial approximation, piecewise linear approximation or stepwise linear approximation (Basaglia *et al.* 1995; Murtagh and Tsoi 1992). These approximations require more iterations of the learning algorithm to converge as compared with standard sigmoids.

For neurons based upon look-up tables, samples of a chosen sigmoid are put into a ROM or RAM to store the desired activation function. Alternatively, we use simplified activation functions that approximate the chosen activation function and are not demanding regarding processor time and memory. Thus, for instance, for the logistic function, its derivative can be expressed as $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, which is simple

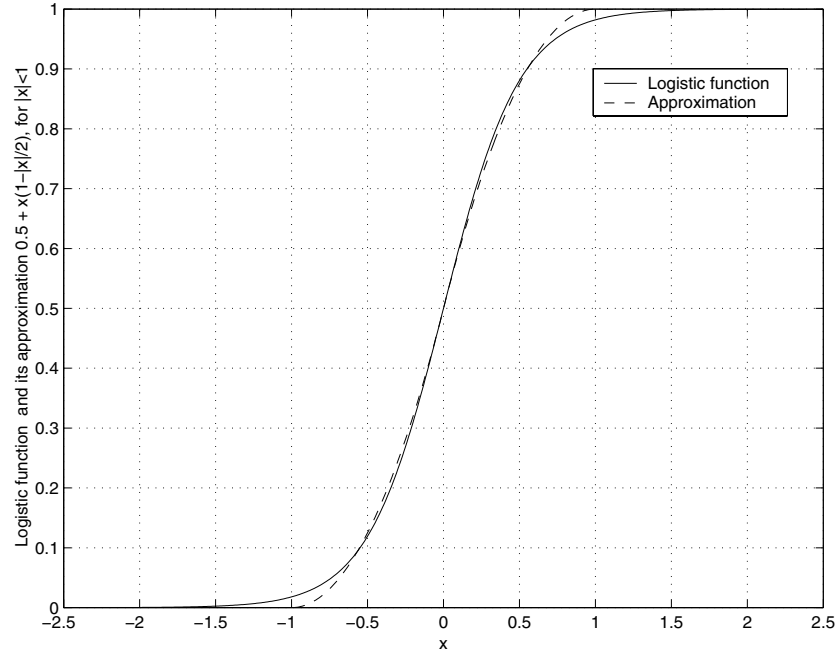


Figure 4.5 Logistic function and its approximation

to calculate. The logistic function can be approximated using

$$f(x) = \begin{cases} 0, & x \leq -1, \\ 0.5 + x(1 - |x|/2), & -1 < x < 1, \\ 1, & x \geq 1. \end{cases} \quad (4.13)$$

The maximal absolute deviation between this function and the logistic function is less than 0.02.¹² The function (4.13) is compared with the logistic function and shown in Figure 4.5. There are several other approximations. To save on computational complexity, we can approximate sigmoid functions with a series of straight lines, i.e. by a piecewise-linear functions. Another sigmoid was proposed by David Elliott (Elliott 1993)

$$\sigma(x) = \frac{x}{1 + |x|} \quad (4.14)$$

with derivative

$$\sigma' = \frac{1}{(1 + |x|)^2} = (1 - |\sigma|)^2,$$

which is also easy to calculate. The function (4.14) and its derivative are shown in Figure 4.6.

In a digital VLSI implementation of an MLP, the computation of the activation function of each neuron is performed using a look-up table (LUT), i.e. a RAM or

¹² <http://www.dontveter.com/bpr/bpr.html>

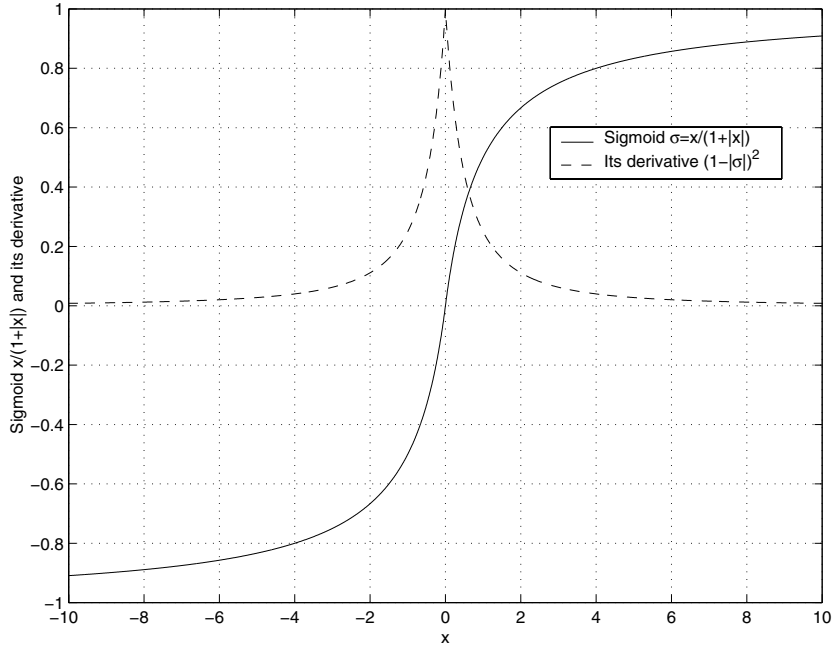


Figure 4.6 Sigmoid function and its derivative

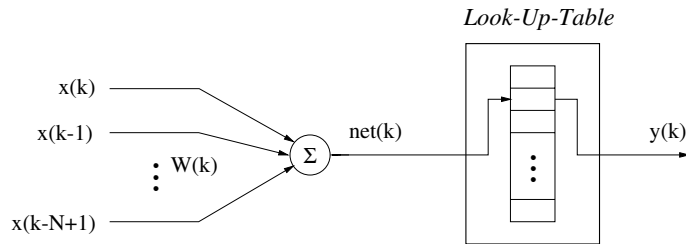


Figure 4.7 LUT neuron

ROM memory which is addressed in some way (Piazza *et al.* 1993). An adaptive LUT based neuron is depicted in Figure 4.7.

Although sigmoidal functions are a typical choice for MLPs, several other functions have been considered. Recently, the use of polynomial activation functions has been proposed (Chon and Cohen 1997; Piazza *et al.* 1992; Song and Manry 1993). Networks with polynomial neurons have been shown to be isomorphic to Volterra filters (Chon and Cohen 1997; Song and Manry 1993). However, calculating a polynomial activation

$$f(x) = a_0 + a_1(x) + \cdots + a_M x^M \quad (4.15)$$

for every neuron and every time instant is extremely computationally demanding and is unlikely to be acceptable for real-time applications. Since their calculation is much slower than simple arithmetic operations, other sigmoidal functions might be useful

for hardware implementations of neural networks for online applications. An overview of such functions is given in Duch and Jankowski (1999).

4.7 MLP versus RBF Networks

MLP- and RBF-based neural networks are the two most commonly used types of feed-forward networks. A fundamental difference between the two is the way in which hidden units combine values at their inputs. MLP networks use inner products, whereas RBFs use Euclidean or Mahalanobis distance as a combination function. An RBF is given by

$$f(\mathbf{x}) = \sum_{i=1}^N c_i G(\mathbf{x}; \mathbf{t}_i), \quad (4.16)$$

where $G(\cdot)$ is a basis function, c_i , $i = 1, \dots, N$, are coefficients, \mathbf{t}_i , $i = 1, \dots, N$, are centres of radial bases, and \mathbf{x} is the input vector.

Both multilayer perceptrons and RBFs have good approximation properties and are related for normalised inputs. In fact, an MLP network can always simulate a Gaussian RBF network, whereas the converse is true only for certain values of the bias parameter (Poggio and Girosi 1990; Yee *et al.* 1999).

4.8 Complex Activation Functions

Recent results suggest that despite the existence of the universal approximation property, approximation by real-valued neural networks might not be continuous (Kainen *et al.* 1999) for some standard types of neural networks, such as Heaviside perceptrons or Gaussian radial basis functions.¹³ For many functions there is not a best approximation in \mathbb{R} . However, there is always a unique best approximation in \mathbb{C} .

Many apparently real-valued mathematical problems can be better understood if they are embedded in the complex plane. Every variable is then represented by a complex number $z = x + jy$, where x and y are real numbers and $j = \sqrt{-1}$. Example problems cast into the complex plane include the analysis of transfer functions and polynomial equations. This has motivated researchers to generalise neural networks to the complex plane (Clarke 1990). Concerning the hardware realisation, the complex weights of the neural network represent impedance as opposed to resistance in real-valued networks.

If we again consider the approximation,

$$f(x) = \sum_{i=1}^N c_i \sigma(x - a_i), \quad (4.17)$$

where σ is a sigmoid function, different choices of σ will give different realisations of f . An extensive analysis of this problem is given in Helmke and Williamson (1995) and

¹³ Intuitively, since a measure of the quality of an approximation is a distance function, for instance, the \mathcal{L}_2 distance given by $(\int_a^b (f(x) - g(x))^2 dx)^{1/2}$, there might occur a case where we have to calculate an integral which is not possible to be calculated within the field of real numbers \mathbb{R} , but which is easy to calculate in the field of complex numbers \mathbb{C} – recall the function e^{-x^2} .

Williamson and Helmke (1995). Going back to elementary function approximation, if $\sigma(x) = x^{-1}$, then (4.17) represents a partial fraction expansion of a rational function f . The coefficients a_i and c_i are, respectively, poles and residuals of (4.17). Notice, however, that both a_i and c_i are allowed to be complex.¹⁴

A complex sigmoid is naturally obtained by analytic continuation of a real-valued sigmoid onto the complex plane. In order to extend a gradient-based learning algorithm for complex signals, the employed activation function should be analytic. Using analytic continuation to extend an activation function to the complex plane, however, has a consequence that, by the Liouville Theorem (Theorem C.1.4), the only bounded differentiable functions defined on the entire complex plane are constant functions. For commonly used activation functions, however, the singularities occur in a limited set.¹⁵ For the logistic function

$$\sigma(z) = \frac{1}{1 + e^{-z}} = u + jv,$$

if z approaches any value in the set $\{0 \pm j(2n+1)\pi, n \in \mathbb{Z}\}$, then $|\sigma(z)| \rightarrow \infty$. Similar conditions for the tanh are $\{0 \pm j((2n+1)/2)\pi, n \in \mathbb{Z}\}$, whereas for e^{-z^2} , we have singularities for $z = 0 + jy$ (Georgiou and Koutsougeras 1992).

Hence, a function obtained by an analytic continuation to the complex plane is generally speaking not an appropriate activation function. Generalising the discussion for real activation functions, properties that a function $\sigma(z) = u(x, y) + jv(x, y)$ should satisfy so that it represents a suitable activation function in the complex plane are (Georgiou and Koutsougeras 1992)

- u and v are nonlinear in x and y ;
- $\sigma(z)$ is bounded $\Rightarrow u$ and v are bounded;
- $\sigma(z)$ has bounded partial derivatives u_x, u_y, v_x, v_y , which satisfy the Cauchy–Riemann conditions (Mathews and Howell 1997);
- $\sigma(z)$ is not entire (not a constant).

Regarding fixed point iteration and global asymptotic stability of neural networks, which will be discussed in more detail in Chapter 7, complex-valued neural networks can generate the dynamics

$$z \leftarrow \Phi(z). \tag{4.18}$$

Functions of the form $cz(1-z)$, for instance, give rise to the Mandelbrot and Julia sets (Clarke 1990; Devaney 1999; Strogatz 1994). A single complex neuron with a feedback connection is thus capable of performing complicated discriminations and generation of nonlinear behaviour.

¹⁴ Going back to transfer function approximation in signal processing, functions of the type (4.17) are able to approximate a Butterworth function of any degree if (4.17) is allowed to have complex coefficients (such as in the case of an RLC realisation). On the other hand, functions with real coefficients (an RC network) cannot approximate a Butterworth function whose order is ≥ 2 .

¹⁵ The exponential function $\exp : \mathbb{C} \rightarrow \mathbb{C} \setminus \{0\}$ maps the set $\{z = a + (2k+1)j\pi\}$, $a \in \mathbb{R}$, $k \in \mathbb{Z}$ onto the negative real axis, which determines singularities of complex sigmoids.

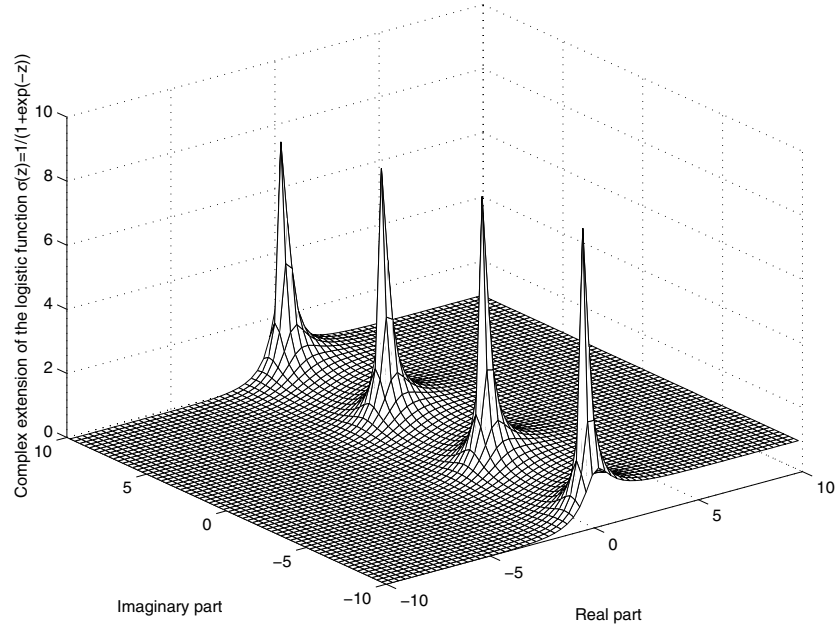


Figure 4.8 Complex extension of the logistic function $\sigma(z) = 1/(1 + e^{-z})$

To provide conditions on the capability of complex neural networks to approximate nonlinear dynamics, a density theorem for complex MLPs with nonanalytic activation function and a hidden layer is proved in Arena *et al.* (1998b). The often cited denseness conditions are, as pointed out by Cotter (1990), special cases of the Stone–Weierstrass theorem.

In the context of learning algorithms, Leung and Haykin (1991) developed their complex backpropagation algorithm considering the following activation function,

$$f(z) = \frac{1}{1 + e^{-z}} : \mathbb{C}^N \rightarrow \mathbb{C}, \quad (4.19)$$

whose magnitude is shown in Figure 4.8. This complex extension of the logistic function has singularities due to the complex exponential in the denominator of (4.19). It is safe to use (4.19) if the inputs are scaled to the range of the complex logistic function where it is analytic. In Benvenuto and Piazza (1992), the following activation function is proposed,

$$\sigma(z) = \sigma(x) + j\sigma(y), \quad (4.20)$$

where $\sigma(z)$ is a two-dimensional extension of a standard sigmoid. The magnitude of this function is shown in Figure 4.9. The function (4.20) is not analytic and bounded on \mathbb{C} . It is, however, discriminatory, and linear combinations of functions of this type are dense in \mathbb{C} (Arena *et al.* 1998a).

Another proposed complex sigmoid is (Benvenuto and Piazza 1992)

$$\sigma(z) = \frac{2c_1}{1 + e^{-c_2 z}} - c_1, \quad (4.21)$$

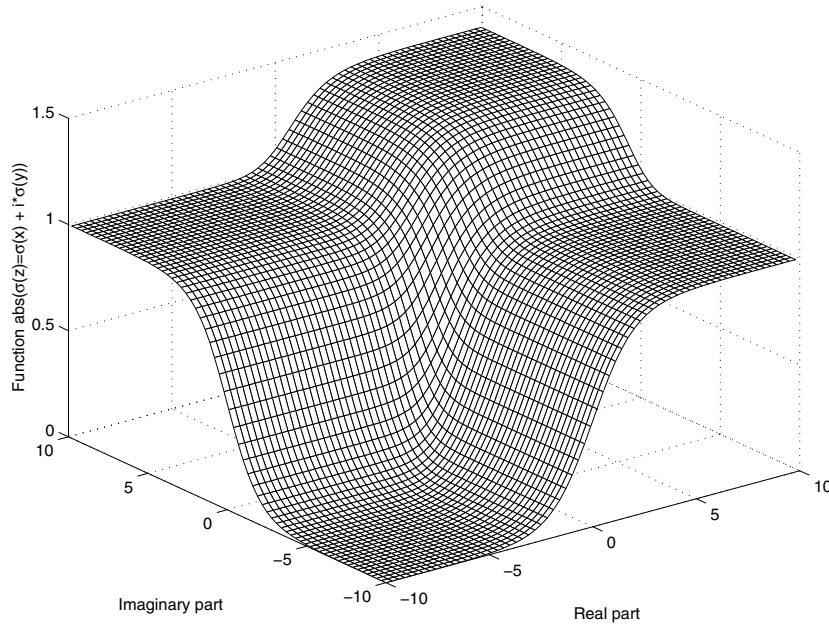


Figure 4.9 Complex sigmoid $\sigma(z) = \sigma(z_r) + j\sigma(z_i)$

where c_1, c_2 are suitable parameters. The derivative of this function is

$$\sigma'(z) = \frac{c_2}{2c_1}(c_1^2 - \sigma^2(z)). \quad (4.22)$$

Other work on complex backpropagation was proposed in Kim and Guest (1990).

A suitable complex activation function would have the property that an excitation near zero would remain close to zero, and large excitations would be mapped into bounded outputs. One such function is given by (Clarke 1990)

$$\sigma(z) = \frac{(\cos \theta + j \sin \theta)(z - s)}{1 - \alpha^* z}, \quad (4.23)$$

where θ is a rotation angle and α is a complex constant of magnitude less than one. The operator $(\cdot)^*$ denotes complex conjugation; the sign of the imaginary part of the asterisked variable is changed. This function is a conformal mapping of the unit disc in the complex plane onto itself and is unique. Further, σ maps large complex numbers into $-1/\alpha$ and thus satisfies the above criteria. The one flaw in σ is a singularity at $z = 1/\alpha$, but in view of Liouville's theorem this is unavoidable. The magnitude plot of this function is shown in Figure 4.10.

A simple function that satisfies all the above properties is (Georgiou and Koutsougeras 1992)

$$f(z) = \frac{z}{c + (1/r)|z|}, \quad (4.24)$$

where c and r are real positive constants. This function maps any point in the complex plane onto the open disc $\{z : |z| < r\}$, as shown in Figure 4.11.

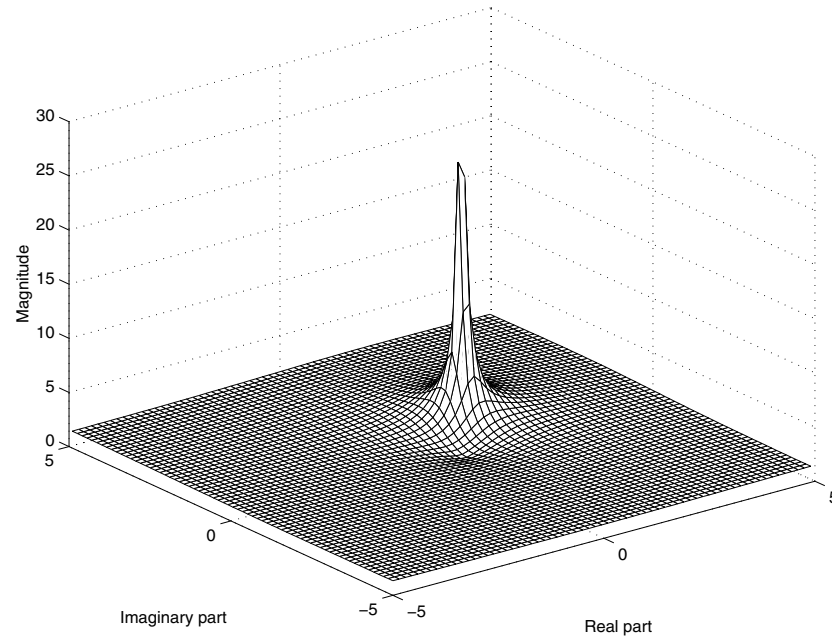


Figure 4.10 A complex activation function

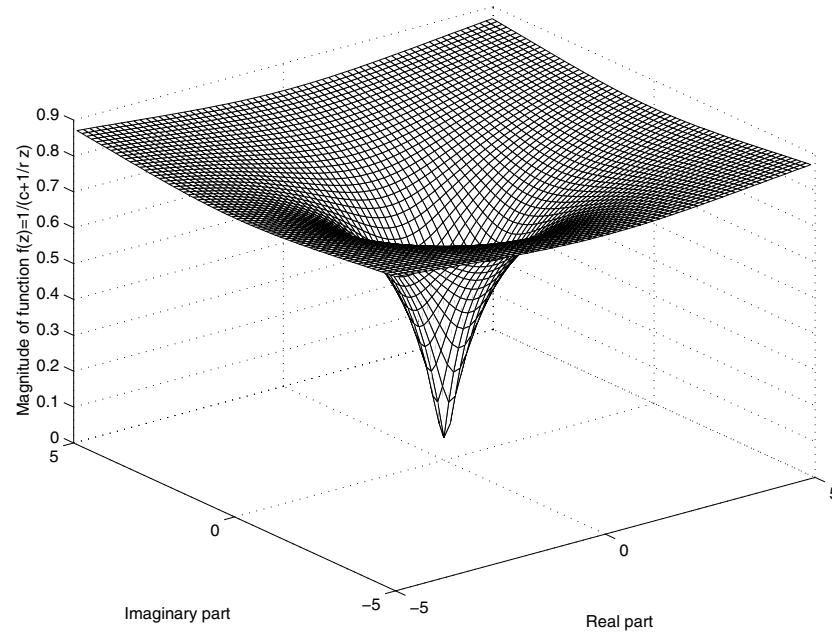


Figure 4.11 Magnitude of the function (4.24)

4.9 Complex Valued Neural Networks as Modular Groups of Compositions of Möbius Transformations

We next offer a different perspective upon some inherent properties of neural networks, such as fixed points, nesting and invertibility, by exposing the representations of neural networks in the framework of Möbius transformations. This framework includes the consideration of complex weights and inputs to the network together with complex sigmoidal activation functions.

4.9.1 Möbius Transformation

Observation 4.9.1.

- (i) The map $g : \mathbb{C} \rightarrow \mathbb{C}$, with $g(z) = e^z$ is holomorphic on \mathbb{C} .
- (ii) The complex sigmoidal activation function $f(z)$ of a neuron in a neural network is holomorphic and conformal.

Definition 4.9.2 (Möbius mapping (Apostol 1997)). Let a, b, c and d denote four complex constants with the restriction that $ad \neq bc$. The function

$$w = f(z) = \frac{az + b}{cz + d} \quad (4.25)$$

is called a Möbius transformation, bilinear transformation or linear fractional transformation.

The condition $ad \neq bc$ is necessary, since for complex variables z_1 and z_2 ,

$$f(z_1) - f(z_2) = \frac{(ad - bc)(z_1 - z_2)}{(cz_1 + d)(cz_2 + d)},$$

which is constant for $ad - bc = 0$. The Möbius transformation is analytic everywhere except for a pole at $z = -d/c$, and also one-to-one and onto a half-plane, and vice versa, which means that its inverse is also a Möbius transformation.

Remark 4.9.3. The Möbius transformation does not determine the coefficients a, b, c, d uniquely, i.e. if $\varphi \in \mathbb{C} \setminus \{0\}$, then coefficients $\varphi a, \varphi b, \varphi c, \varphi d$ correspond to the same transformation.

In addition, every Möbius transformation (except $f(z) = z$) has one or two fixed points¹⁶ z^* , such that $f(z^*) = z^*$ (Apostol 1997).

4.9.2 Activation Functions and Möbius Transformations

Equating the coefficients associated with the powers of z in the transfer function of a sigmoid activation function, to those in a general form of the Möbius transformation (4.25), i.e.

$$\frac{1 - e^{-\beta \text{net}}}{1 + e^{-\beta \text{net}}} = \frac{az + b}{cz + d} \quad (4.26)$$

¹⁶ See Definition 4.9.7.

we see that the tanh function satisfies the conditions of a Möbius transformation for $z = e^{-\beta \text{net}}$ and (Mandic 2000c)

$$a = -1, \quad b = 1, \quad c = 1, \quad d = 1. \quad (4.27)$$

Also, the condition $ad - bc \neq 0$ is satisfied. Therefore, the hyperbolic tangent activation function is a Möbius transformation and holomorphic. So too is the logistic function, for which $a = 0, b = 1, c = 1, d = 1$ and $ad - bc \neq 0$.

Proposition 4.9.4 (see Dumitras and Lazarescu 1997). *The sigmoidal transformation $f(z)$ performed by a neuron in a neural network on a complex signal $z = \alpha + j\beta$ is a Möbius transformation.*

To analyse an N -dimensional nested nonlinear system, as exhibited by a feedforward and recurrent neural network with hidden neurons, we use the notion of a modular group (Apostol 1997), which is explained in detail in Appendix C.1.1.

Example 4.9.5. Show that the transfer function between neurons in consecutive layers within a general neural network belongs to a modular group Γ of compositions of Möbius transformations.

Solution. Without loss in generality let us consider only two neurons from consecutive layers. Notice that their nonlinear functions are nested. Let us denote the functions performed by these two neurons by

$$H_1(z) = \frac{a_1 z + b_1}{c_1 z + d_1} \quad \text{and} \quad H_2(z) = \frac{a_2 z + b_2}{c_2 z + d_2}.$$

Then their composition (transfer function from a layer to a layer) is $H_1(H_2(z)) = H_1 \circ H_2$ and can be expressed as

$$\begin{aligned} H_1 \circ H_2 &= \left(a_1 \frac{a_2 z + b_2}{c_2 z + d_2} + b_1 \right) / \left(c_1 \frac{a_2 z + b_2}{c_2 z + d_2} + d_1 \right) \\ &= \frac{(a_1 a_2 + b_1 c_2) z + a_1 b_2 + b_1 d_2}{(a_2 c_1 + c_2 d_1) z + b_2 c_1 + d_1 d_2}, \end{aligned} \quad (4.28)$$

which belongs to group Γ . Also if the Möbius mappings performed by H_1 and H_2 are, respectively, described by matrices

$$M_1 = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \quad \text{and} \quad M_2 = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix},$$

then the composition $H_1 \circ H_2$ is described by

$$\begin{aligned} H(z) = H_1(z) \circ H_2(z) &\Leftrightarrow (M_1 \times M_2)z = \left(\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \times \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} \right) z \\ &= \begin{bmatrix} a_1 a_2 + b_1 c_2 & a_1 b_2 + b_1 d_2 \\ a_2 c_1 + c_2 d_1 & b_2 c_1 + d_1 d_2 \end{bmatrix} z = Mz, \end{aligned} \quad (4.29)$$

which again belongs to the group Γ . \square

Observation 4.9.6 (see Mandic 2000b,c). *The global input–output relationship in a neural network can be considered as a Möbius transformation.*

4.9.3 Existence and Uniqueness of Fixed Points in a Complex Neural Network via Theory of Modular Groups

Definition 4.9.7. A point x^* which is mapped onto itself under a map G so that $x^* = G(x^*)$ is called a fixed point.

Since by the Brouwer fixed point theorem (Zeidler 1986), a continuous function on a convex, closed and bounded set has a fixed point, so too Möbius transformations have fixed points. Therefore, it suffices to investigate the characteristics of the nonlinear activation function to obtain fixed points for a general neural network based nonlinear system. In that case, both the external input vector to the system $\mathbf{x}(k)$ and the parameter vector $\mathbf{w}(k)$ are assumed to form a time-invariant part of the variable net.

In Zakeri (1996), it is proven that a proper holomorphic map in the complex plane is uniquely determined up to post-composition with a Möbius transformation by its critical points. To connect fixed (critical) points and nested Möbius transformations, which occur in RNN-based architectures, we use the following observation.

Observation 4.9.8 (see Mandic 2000b). For a given architecture, fixed points of the complex activation function employed in a neural network determine the fixed points of the neural network considered.

Existence. The existence conditions for a fixed point of a sigmoid activation function are provided by recognising that a sigmoid activation function is a Möbius transformation (Proposition 4.9.4). Since every Möbius map has one or two fixed points (Mathews and Howell 1997), so too does a sigmoid activation function of a neuron. By Observation 4.9.6, a global input–output relationship in a neural network can be considered in the framework of compositions of Möbius transformations, and hence has fixed points.

Uniqueness. By Theorem C.1.5, transfer functions of two neurons in consecutive layers can be described by matrices $M_1 = T^{n_1}ST^{n_2}S \dots ST^{n_k}$ and $M_2 = T^{m_1}ST^{n_2}S \dots ST^{m_l}$. Using the property described in Zakeri (1996), the composition of their activation functions (nested nonlinearities) represented by the product $M_1 \times M_2$ has critical points defined by M_1 and M_2 .

As a consequence of complex activation functions being Möbius transformations, we also conclude that (Mandic 2000c)

- a general neural network has an inverse (the existence of an inverse);
- the inverse of a neural network is not necessarily unique (the uniqueness of an inverse).

Since the input–output relationship in a complex neural network can be considered in the framework of Möbius transformations (which has an inverse, which is again a Möbius transformation), a general complex neural network has an inverse (*existence*). On the other hand, Möbius transformations are uniformly determined up to a multiplication by a constant. This means that there is more than one solution to the inverse problem (*uniqueness*).

4.10 Summary

An overview of nonlinear activation functions used in neural networks has been provided. We have started from the problem of function approximation and trajectory learning and evaluated neural networks suitable for these problems. Properties of neural networks realised in hardware have also been addressed. For rigour, the analysis has been extended to neural networks with complex activation functions, for which we have built a unified framework via modular groups of Möbius transformations.

Existence and uniqueness conditions of fixed points and invertibility of such mappings have been derived. These results apply both for the general input–output relationship in a neural network as well as for a single neuron. This analysis provides a strong mathematical background for further analysis of neural networks for adaptive filtering and prediction.

5

Recurrent Neural Networks Architectures

5.1 Perspective

In this chapter, the use of neural networks, in particular recurrent neural networks, in system identification, signal processing and forecasting is considered. The ability of neural networks to model nonlinear dynamical systems is demonstrated, and the correspondence between neural networks and block-stochastic models is established. Finally, further discussion of recurrent neural network architectures is provided.

5.2 Introduction

There are numerous situations in which the use of linear filters and models is limited. For instance, when trying to identify a saturation type nonlinearity, linear models will inevitably fail. This is also the case when separating signals with overlapping spectral components.

Most real-world signals are generated, to a certain extent, by a nonlinear mechanism and therefore in many applications the choice of a nonlinear model may be necessary to achieve an acceptable performance from an adaptive predictor. Communications channels, for instance, often need nonlinear equalisers to achieve acceptable performance. The choice of model has crucial importance¹ and practical applications have shown that nonlinear models can offer a better prediction performance than their linear counterparts. They also reveal rich dynamical behaviour, such as limit cycles, bifurcations and fixed points, that cannot be captured by linear models (Gershenfeld and Weigend 1993).

By *system* we consider the actual underlying physics² that generate the data, whereas by *model* we consider a mathematical description of the system. Many variations of mathematical models can be postulated on the basis of datasets collected from observations of a system, and their suitability assessed by various performance

¹ System identification, for instance, consists of choice of the model, model parameter estimation and model validation.

² Technically, the notions of *system* and *process* are equivalent (Pearson 1995; Sjöberg *et al.* 1995).

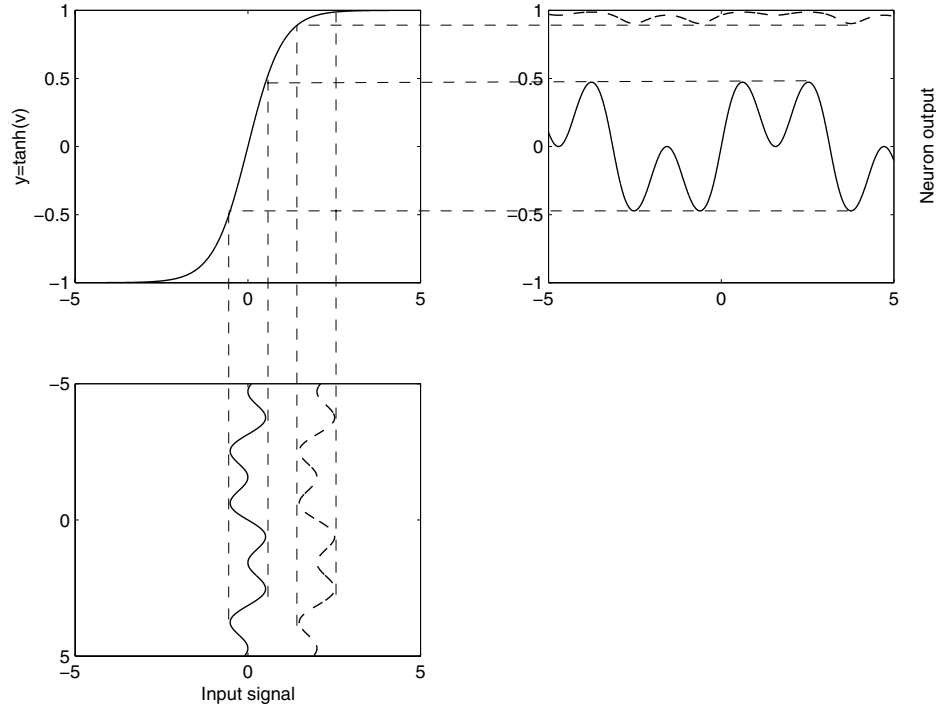


Figure 5.1 Effects of $y = \tanh(v)$ nonlinearity in a neuron model upon two example inputs

metrics. Since it is not possible to characterise nonlinear systems by their impulse response, one has to resort to less general models, such as homomorphic filters, morphological filters and polynomial filters. Some of the most frequently used polynomial filters are based upon Volterra series (Mathews 1991), a nonlinear analogue of the linear impulse response, threshold autoregressive models (TAR) (Priestley 1991) and Hammerstein and Wiener models. The latter two represent structures that consist of a linear dynamical model and a static zero-memory nonlinearity. An overview of these models can be found in Haber and Unbehauen (1990). Notice that for nonlinear systems, the ordering of the modules within a modular structure³ plays an important role.

To illustrate some important features associated with nonlinear neurons, let us consider a squashing nonlinear activation function of a neuron, shown in Figure 5.1. For two identical mixed sinusoidal inputs with different offsets, passed through this nonlinearity, the output behaviour varies from amplifying and slightly distorting the input signal (solid line in Figure 5.1) to attenuating and considerably nonlinearly distorting the input signal (broken line in Figure 5.1). From the viewpoint of system theory, neural networks represent nonlinear maps, mapping one metric space to another.

³ To depict this, for two modules performing nonlinear functions $H_1 = \sin(x)$ and $H_2 = e^x$, we have $H_1(H_2(x)) \neq H_2(H_1(x))$ since $\sin(e^x) \neq e^{\sin(x)}$. This is the reason to use the term *nesting* rather than cascading in modular neural networks.

Nonlinear system modelling has traditionally focused on Volterra–Wiener analysis. These models are nonparametric and computationally extremely demanding. The Volterra series expansion is given by

$$y(k) = h_0 + \sum_{i=0}^N h_1(i)x(k-i) + \sum_{i=0}^N \sum_{j=0}^N h_2(i,j)x(k-i)x(k-j) + \dots \quad (5.1)$$

for the representation of a causal system. A nonlinear system represented by a Volterra series is completely characterised by its Volterra kernels h_i , $i = 0, 1, 2, \dots$. The Volterra modelling of a nonlinear system requires a great deal of computation, and mostly second- or third-order Volterra systems are used in practice.

Since the Volterra series expansion is a Taylor series expansion with memory, they both fail when describing a system with discontinuities, such as

$$y(k) = A \operatorname{sgn}(x(k)), \quad (5.2)$$

where $\operatorname{sgn}(\cdot)$ is the signum function.

To overcome this difficulty, nonlinear parametric models of nonlinear systems, termed NARMAX, that are described by nonlinear difference equations, have been introduced (Billings 1980; Chon and Cohen 1997; Chon *et al.* 1999; Connor 1994). Unlike the Volterra–Wiener representation, the NARMAX representation of nonlinear systems offers compact representation.

The NARMAX model describes a system by using a nonlinear functional dependence between lagged inputs, outputs and/or prediction errors. A polynomial expansion of the transfer function of a NARMAX neural network does not comprise of delayed versions of input and output of order higher than those presented to the network. Therefore, the input of an insufficient order will result in undermodelling, which complies with Takens' embedding theorem (Takens 1981).

Applications of neural networks in forecasting, signal processing and control require treatment of dynamics associated with the input signal. Feedforward networks for processing of dynamical systems tend to capture the dynamics by including past inputs in the input vector. However, for dynamical modelling of complex systems, there is a need to involve feedback, i.e. to use recurrent neural networks. There are various configurations of recurrent neural networks, which are used by Jordan (1986) for control of robots, by Elman (1990) for problems in linguistics and by Williams and Zipser (1989a) for nonlinear adaptive filtering and pattern recognition. In Jordan's network, past values of network outputs are fed back into hidden units, in Elman's network, past values of the outputs of hidden units are fed back into themselves, whereas in the Williams–Zipser architecture, the network is fully connected, having one hidden layer.

There are numerous modular and hybrid architectures, combining linear adaptive filters and neural networks. These include the pipelined recurrent neural network and networks combining recurrent networks and FIR adaptive filters. The main idea here is that the linear filter captures the linear 'portion' of the input process, whereas a neural network captures the nonlinear dynamics associated with the process.

5.3 Overview

The basic modes of modelling, such as *parametric*, *nonparametric*, *white box*, *black box* and *grey box* modelling are introduced. Afterwards, the dynamical richness of neural models is addressed and feedforward and recurrent modelling for noisy time series are compared. Block-stochastic models are introduced and neural networks are shown to be able to represent these models. The chapter concludes with an overview of recurrent neural network architectures and recurrent neural networks for NARMAX modelling.

5.4 Basic Modes of Modelling

The notions of parametric, nonparametric, black box, grey box and white box modelling are explained. These can be used to categorise neural network algorithms, such as the direct gradient computation, *a posteriori* and normalised algorithms. The basic idea behind these approaches to modelling is *not to estimate what is already known*. One should, therefore, utilise prior knowledge and knowledge about the physics of the system, when selecting the neural network model prior to parameter estimation.

5.4.1 Parametric versus Nonparametric Modelling

A review of nonlinear input–output modelling techniques is given in Pearson (1995). Three classes of input–output models are *parametric*, *nonparametric* and *semiparametric* models. We next briefly address them.

- *Parametric* modelling assumes a fixed structure for the model. The model identification problem then simplifies to estimating a finite set of parameters of this fixed model. This estimation is based upon the prediction of real input data, so as to best match the input data dynamics. An example of this technique is the broad class of ARIMA/NARMA models. For a given structure of the model (NARMA for instance) we recursively estimate the parameters of the chosen model.
- *Nonparametric* modelling seeks a particular model structure from the input data. The actual model is not known beforehand. An example taken from nonparametric regression is that we look for a model in the form of $y(k) = f(x(k))$ without knowing the function $f(\cdot)$ (Pearson 1995).
- *Semiparametric* modelling is the combination of the above. Part of the model structure is completely specified and known beforehand, whereas the other part of the model is either not known or loosely specified.

Neural networks, especially recurrent neural networks, can be employed within estimators of all of the above classes of models. Closely related to the above concepts are white, grey and black box modelling techniques.

5.4.2 *White, Grey and Black Box Modelling*

To understand and analyse real-world physical phenomena, various mathematical models have been developed. Depending on some *a priori* knowledge about the process, data and model, we differentiate between three fairly general modes of modelling. The idea is to distinguish between three levels of prior knowledge, which have been ‘colour-coded’. An overview of the white, grey and black box modelling techniques can be found in Aguirre (2000) and Sjöberg *et al.* (1995).

Given data gathered from planet movements, then Kepler’s gravitational laws might well provide the initial framework in building a mathematical model of the process. This mode of modelling is referred to as *white box* modelling (Aguirre 2000), underlying its fairly deterministic nature. Static data are used to calculate the parameters, and to do that the underlying physical process has to be understood. It is therefore possible to build a *white box* model entirely from physical insight and prior knowledge. However, the underlying physics are generally not completely known, or are too complicated and often one has to resort to other types of modelling.

The exact form of the input–output relationship that describes a real-world system is most commonly unknown, and therefore modelling is based upon a chosen set of known functions. In addition, if the model is to approximate the system with an arbitrary accuracy, the set of chosen nonlinear continuous functions must be dense. This is the case with polynomials. In this light, neural networks can be viewed as another mode of functional representations. *Black box* modelling therefore assumes no previous knowledge about the system that produces the data. However, the chosen network structure belongs to architectures that are known to be flexible and have performed satisfactorily on similar problems. The aim hereby is to find a function F that approximates the process y based on the previous observations of process y_{PAST} and input u , as

$$y = F(y_{\text{PAST}}, u). \quad (5.3)$$

This ‘black box’ establishes a functional dependence between the input and output, which can be either linear or nonlinear. The downside is that it is generally not possible to learn about the true physical process that generates the data, especially if a linear model is used. Once the training process is complete, a neural network represents a *black box*, nonparametric process model. Knowledge about the process is embedded in the values of the network parameters (i.e. synaptic weights).

A natural compromise between the two previous models is so-called *grey box* modelling. It is obtained from *black box* modelling if some information about the system is known *a priori*. This can be a probability density function, general statistics of the process data, impulse response or attractor geometry. In Sjöberg *et al.* (1995), two subclasses of *grey box* models are considered: *physical* modelling, where a model structure is built upon understanding of the underlying physics, as for instance the state-space model structure; and *semiphysical* modelling, where, based upon physical insight, certain nonlinear combinations of data structures are suggested, and then estimated by *black box* methodology.

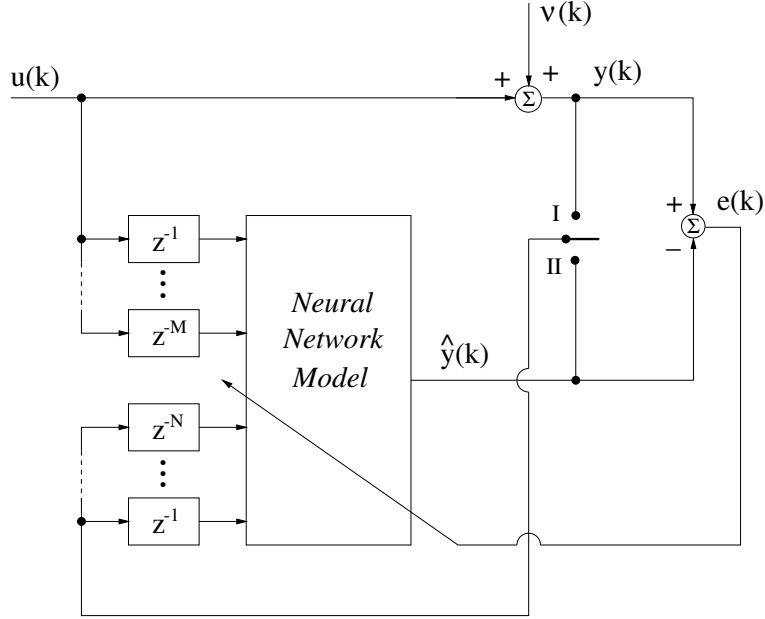


Figure 5.2 Nonlinear prediction configuration using a neural network model

5.5 NARMAX Models and Embedding Dimension

For neural networks, the number of input nodes specifies the dimension of the network input. In practice, the true state of the system is not observable and the mathematical model of the system that generates the dynamics is not known. The question arises: is the sequence of measurements $\{y(k)\}$ sufficient to reconstruct the nonlinear system dynamics? Under some regularity conditions, Takens' (1981) and Mane's (1981) embedding theorems establish this connection. To ensure that the dynamics of a nonlinear process estimated by a neural network are fully recovered, it is convenient to use Takens' embedding theorem (Takens 1981), which states that to obtain a faithful reconstruction of the system dynamics, the *embedding dimension* d must satisfy

$$d \geq 2D + 1, \quad (5.4)$$

where D is the dimension of the system attractor. Takens' embedding theorem (Takens 1981; Wan 1993) establishes a diffeomorphism between a finite window of the time series

$$[y(k-1), y(k-2), \dots, y(k-N)] \quad (5.5)$$

and the underlying state of the dynamic system which generates the time series. This implies that a nonlinear regression

$$y(k) = g[y(k-1), y(k-2), \dots, y(k-N)] \quad (5.6)$$

can model the nonlinear time series. An important feature of the delay-embedding theorem due to Takens (1981) is that it is physically implemented by delay lines.

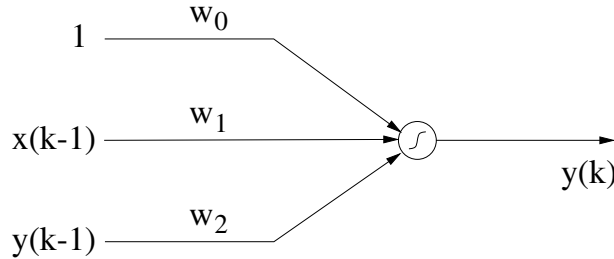


Figure 5.3 A NARMAX recurrent perceptron with $p = 1$ and $q = 1$

There is a deep connection between time-lagged vectors and underlying dynamics. Delay vectors are not just a representation of a state of the system, their length is the key to recovering the full dynamical structure of a nonlinear system. A general starting point would be to use a network for which the input vector comprises delayed inputs and outputs, as shown in Figure 5.2. For the network in Figure 5.2, both the input and the output are passed through delay lines, hence indicating the NARMAX character of this network. The switch in this figure indicates two possible modes of learning which will be explained in Chapter 6.

5.6 How Dynamically Rich are Nonlinear Neural Models?

To make an initial step toward comparing neural and other nonlinear models, we perform a Taylor series expansion of the sigmoidal nonlinear activation function of a single neuron model as (Billings *et al.* 1992)

$$\Phi(v(k)) = \frac{1}{1 + e^{-\beta v(k)}} = \frac{1}{2} + \frac{\beta}{4}v(k) - \frac{\beta^3}{48}v^3(k) + \frac{\beta^5}{480}v^5(k) - \frac{17\beta^7}{80640}v^7(k) + \dots \quad (5.7)$$

Depending on the steepness β and the activation potential $v(k)$, the polynomial representation (5.7) of the transfer function of a neuron exhibits a complex nonlinear behaviour.

Let us now consider a NARMAX recurrent perceptron with $p = 1$ and $q = 1$, as shown in Figure 5.3, which is a simple example of recurrent neural networks. Its mathematical description is given by

$$y(k) = \Phi(w_1x(k-1) + w_2y(k-1) + w_0). \quad (5.8)$$

Expanding (5.8) using (5.7) yields

$$y(k) = \frac{1}{2} + \frac{1}{4}[w_1x(k-1) + w_2y(k-1) + w_0] - \frac{1}{48}[w_1x(k-1) + w_2y(k-1) + w_0]^3 + \dots, \quad (5.9)$$

where $\beta = 1$. Expression (5.9) illustrates the dynamical richness of squashing activation functions. The associated dynamics, when represented in terms of polynomials are quite complex. Networks with more neurons and hidden layers will produce more complicated dynamics than those in (5.9). Following the same approach, for a general

recurrent neural network, we obtain (Billings *et al.* 1992)

$$\begin{aligned} y(k) = & c_0 + c_1x(k-1) + c_2y(k-1) + c_3x^2(k-1) \\ & + c_4y^2(k-1) + c_5x(k-1)y(k-1) + c_6x^3(k-1) \\ & + c_7y^3(k-1) + c_8x^2(k-1)y(k-1) + \dots \end{aligned} \quad (5.10)$$

Equation (5.10) does not comprise delayed versions of input and output samples of order higher than those presented to the network. If the input vector were of an insufficient order, undermodelling would result, which complies with Takens' embedding theorem. Therefore, when modelling an unknown dynamical system or tracking unknown dynamics, it is important to concentrate on the embedding dimension of the network. Representation (5.10) also models an offset (mean value) c_0 of the input signal.

5.6.1 Feedforward versus Recurrent Networks for Nonlinear Modelling

The choice of which neural network to employ to represent a nonlinear physical process depends on the dynamics and complexity of the network that is best for representing the problem in hand. For instance, due to feedback, recurrent networks may suffer from instability and sensitivity to noise. Feedforward networks, on the other hand, might not be powerful enough to capture the dynamics of the underlying nonlinear dynamical system. To illustrate this problem, we resort to a simple IIR (ARMA) linear system described by the following first-order difference equation

$$z(k) = 0.5z(k-1) + 0.1x(k-1). \quad (5.11)$$

The system (5.11) is stable, since the pole of its transfer function is at 0.5, i.e. within the unit circle in the z -plane. However, in a noisy environment, the output $z(k)$ is corrupted by noise $e(k)$, so that the noisy output $y(k)$ of system (5.11) becomes

$$y(k) = z(k) + e(k), \quad (5.12)$$

which will affect the quality of estimation based on this model. This happens because the noise terms accumulate during recursions⁴ (5.11) as

$$y(k) = 0.5y(k-1) + 0.1x(k-1) + e(k) - 0.5e(k-1). \quad (5.13)$$

An equivalent FIR (MA) representation of the same filter (5.11), using the method of long division, gives

$$z(k) = 0.1x(k-1) + 0.05x(k-2) + 0.025x(k-3) + 0.0125x(k-4) + \dots \quad (5.14)$$

and the representation of a noisy system now becomes

$$y(k) = 0.1x(k-1) + 0.05x(k-2) + 0.025x(k-3) + 0.0125x(k-4) + \dots + e(k). \quad (5.15)$$

⁴ Notice that if the noise $e(k)$ is zero mean and white it appears coloured in (5.13), i.e. correlated with previous outputs, which leads to biased estimates.

Clearly, the noise in (5.15) is not correlated with the previous outputs and the estimates are unbiased.⁵ The price to pay, however, is the infinite length of the exact representation of (5.11).

A similar principle applies to neural networks. In Chapter 6 we address the modes of learning in neural networks and discuss the bias/variance dilemma for recurrent neural networks.

5.7 Wiener and Hammerstein Models and Dynamical Neural Networks

Under relatively mild conditions,⁶ the output signal of a nonlinear model can be considered as a combination of outputs from some suitable submodels. The structure identification, model validation and parameter estimation based upon these submodels are more convenient than those of the whole model. Block oriented stochastic models consist of static nonlinear and dynamical linear modules. Such models often occur in practice, examples of which are

- the Hammerstein model, where a zero-memory nonlinearity is followed by a linear dynamical system characterised by its transfer function $H(z) = N(z)/D(z)$;
- the Wiener model, where a linear dynamical system is followed by a zero-memory nonlinearity.

5.7.1 Overview of Block-Stochastic Models

The definitions of certain stochastic models are given by the

1. Wiener system

$$y(k) = g(H(z^{-1})u(k)), \quad (5.16)$$

where $u(k)$ is the input to the system, $y(k)$ is the output,

$$H(z^{-1}) = \frac{C(z^{-1})}{D(z^{-1})}$$

is the z -domain transfer function of the linear component of the system and $g(\cdot)$ is a nonlinear function;

2. Hammerstein system

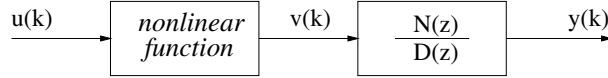
$$y(k) = H(z^{-1})g(u(k)); \quad (5.17)$$

3. Uryson system, defined by

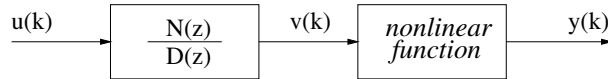
$$y(k) = \sum_{i=1}^M H_i(z^{-1})g_i(u(k)). \quad (5.18)$$

⁵ Under the usual assumption that the external additive noise $e(k)$ is not correlated with the input signal $x(k)$.

⁶ A finite degree polynomial steady-state characteristic.



(a) The Hammerstein stochastic model



(b) The Wiener stochastic model

Figure 5.4 Nonlinear stochastic models used in control and signal processing

Theoretically, there are finite size neural systems with dynamic synapses which can represent all of the above. Moreover, some modular neural architectures, such as the PRNN (Haykin and Li 1995), are able to represent block-cascaded Wiener–Hammerstein systems described by (Mandic and Chambers 1999c)

$$y(k) = \Phi_N(H_N(z^{-1})\Phi_{N-1}(H_{N-1}(z^{-1}) \cdots \Phi_1(H_1(z^{-1})u(k)))) \quad (5.19)$$

and

$$y(k) = H_N(z^{-1})\Phi_N(H_{N-1}(z^{-1})\Phi_{N-1} \cdots \Phi_1(H_1(z^{-1})u(k))) \quad (5.20)$$

under certain constraints relating the size of networks and order of block-stochastic models. Due to its parallel nature, however, a general Uryson model is not guaranteed to be representable this way.

5.7.2 Connection Between Block-Stochastic Models and Neural Networks

Block diagrams of Wiener and Hammerstein systems are shown in Figure 5.4. The nonlinear function from Figure 5.4(a) can be generally assumed to be a polynomial,⁷ i.e.

$$v(k) = \sum_{i=0}^M \lambda_i u^i(k). \quad (5.21)$$

The Hammerstein model is a conventional parametric model, usually used to represent processes with nonlinearities involved with the process inputs, as shown in Figure 5.4(a). The equation describing the output of a SISO Hammerstein system corrupted with additive output noise $\eta(k)$ is

$$y(k) = \Phi[u(k-1)] + \sum_{i=2}^{\infty} h_i \Phi[u(k-i)] + \nu(k), \quad (5.22)$$

where Φ is a nonlinear function which is continuous. Other requirements are that the linear dynamical subsystem is stable. This network is shown in Figure 5.5.

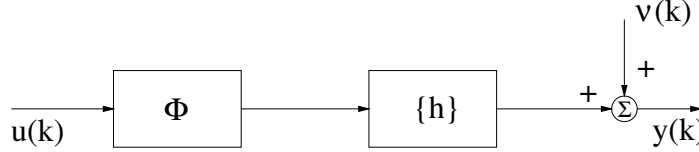


Figure 5.5 Discrete-time SISO Hammerstein model with observation noise

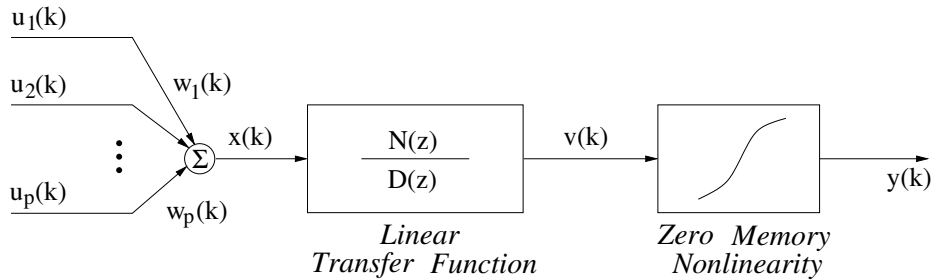


Figure 5.6 Dynamic perceptron

Neural networks with locally distributed dynamics (LDNN) can be considered as locally recurrent networks with global feedforward features. An example of these networks is the *dynamical multilayer perceptron* (DMLP) which consists of dynamical neurons and is shown in Figure 5.6. The model of this dynamic perceptron is described by

$$\left. \begin{aligned} y(k) &= \Phi(v(k)), \\ v(k) &= \sum_{i=0}^{\deg(N(z))} n_i(k)x(k-i) + 1 + \sum_{j=1}^{\deg(D(z))} d_j(k)v(k-j), \\ x(k) &= \sum_{l=1}^p w_l(k)u_l(k), \end{aligned} \right\} \quad (5.23)$$

where n_i and d_i denote, respectively, the coefficients of the polynomials in $N(z)$ and $D(z)$ and ‘1’ is included for a possible bias input. From Figure 5.6, the transfer function between $y(k)$ and $x(k)$ represents a Wiener system. Hence, combinations of dynamical perceptrons (such as a recurrent neural network) are able to represent block-stochastic Wiener–Hammerstein models. Gradient-based learning rules can be developed for a recurrent neural network representing block-stochastic models. Both the Wiener and Hammerstein models can exhibit a more general structure, as shown in Figure 5.7, for the Hammerstein model. Wiener and Hammerstein models can be combined to produce more complicated block-stochastic models. A representative of these models is the Wiener–Hammerstein model, shown in Figure 5.8. This figure shows a Wiener stochastic model, followed by a linear dynamical system represented by its transfer function $H_2(z) = N_2(z)/D_2(z)$, hence building a Wiener–Hammerstein

⁷ By the Weierstrass theorem, polynomials can approximate arbitrarily well any nonlinear function, including sigmoid functions.

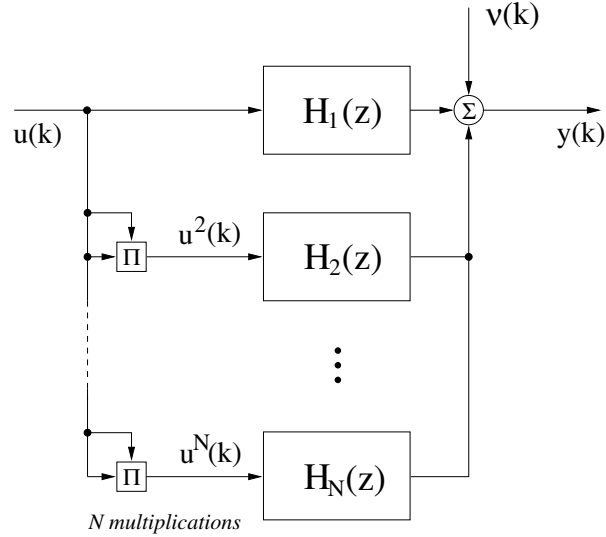


Figure 5.7 Generalised Hammerstein model

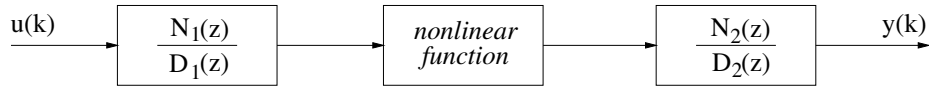


Figure 5.8 The Wiener-Hammerstein model

block-stochastic system. In practice, we can build complicated block cascaded systems this way.

Wiener and Hammerstein systems are frequently used to compensate each other (Kang *et al.* 1998). This includes finding an inverse of the first module in the combination. If these models are represented by neural networks, Chapter 4 provides a general framework for uniqueness, existence and convergence of inverse neural models. The following example from Billings and Voon (1986) shows that the Wiener model can be represented by a NARMA model, which, in turn can be modelled by a recurrent neural network.

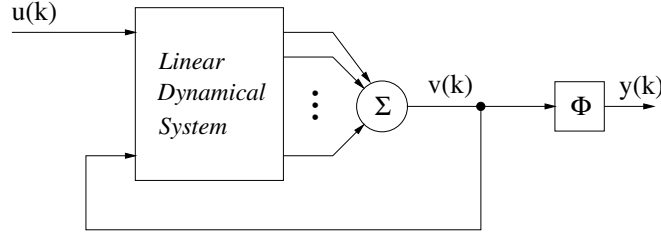
Example 5.7.1. The Wiener model

$$\left. \begin{aligned} w(k) &= 0.8w(k-1) + 0.4u(k-1), \\ y(k) &= w(k) + w^3(k) + e(k), \end{aligned} \right\} \quad (5.24)$$

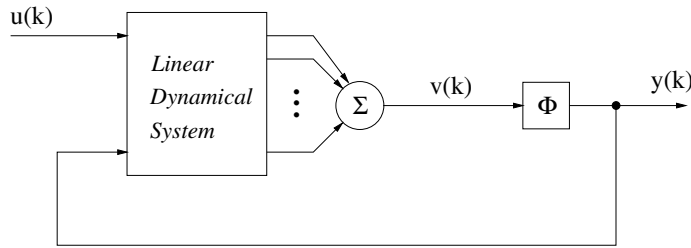
was identified as

$$\begin{aligned} y(k) &= 0.7578y(k-1) + 0.3891u(k-1) - 0.03723y^2(k-1) \\ &\quad + 0.3794y(k-1)u(k-1) + 0.0684u^2(k-1) + 0.1216y(k-1)u^2(k-1) \\ &\quad + 0.0633u^3(k-1) - 0.739e(k-1) - 0.368u(k-1)e(k-1) + e(k), \end{aligned} \quad (5.25)$$

which is a NARMA model, and hence can be realised by a recurrent neural network.



(a) Activation feedback scheme



(b) Output feedback scheme

Figure 5.9 Recurrent neural network architectures

5.8 Recurrent Neural Network Architectures

Two straightforward ways to include recurrent connections in neural networks are *activation feedback* and *output feedback*, as shown, respectively, in Figure 5.9(a) and Figure 5.9(b). These schemes are closely related to the state space representation of neural networks. A comprehensive and insightful account of canonical forms and state space representation of general neural networks is given in Nerrand *et al.* (1993) and Dreyfus and Idan (1998). In Figure 5.9, the blocks labelled ‘linear dynamical systems’ comprise of delays and multipliers, hence providing linear combination of their input signals. The output of a neuron shown in Figure 5.9(a) can be expressed as

$$\left. \begin{aligned} v(k) &= \sum_{i=0}^M w_{u,i}(k)u(k-i) + \sum_{j=1}^N w_{v,j}(k)v(k-j), \\ y(k) &= \Phi(v(k)), \end{aligned} \right\} \quad (5.26)$$

where $w_{u,i}$ and $w_{v,j}$ correspond to the weights associated with u and v , respectively.

The transfer function of a neuron shown in Figure 5.9(b) can be expressed as

$$\left. \begin{aligned} v(k) &= \sum_{i=0}^M w_{u,i}(k)u(k-i) + \sum_{j=1}^N w_{y,j}(k)y(k-j), \\ y(k) &= \Phi(v(k)), \end{aligned} \right\} \quad (5.27)$$

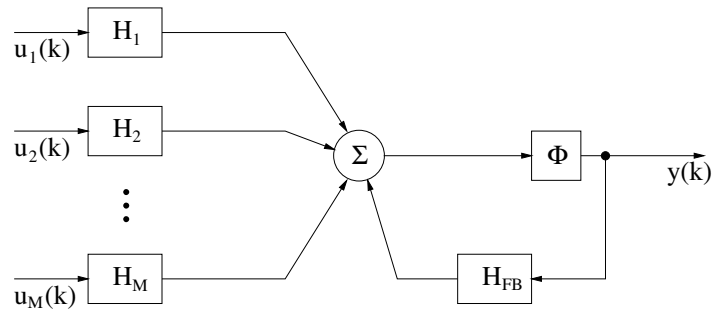


Figure 5.10 General LRGF architecture

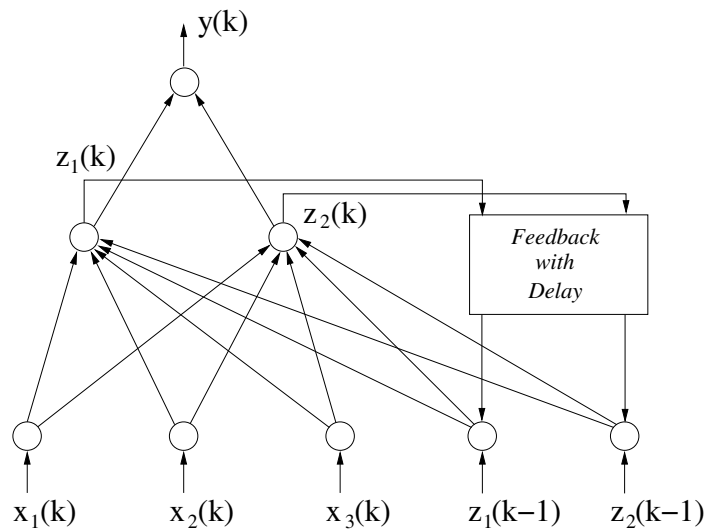


Figure 5.11 An example of Elman recurrent neural network

where $w_{y,j}$ correspond to the weights associated with the delayed outputs. A comprehensive account of types of synapses and short-term memories in dynamical neural networks is provided by Mozer (1993).

The networks mentioned so far exhibit a locally recurrent architecture, but when connected into a larger network, they have a feedforward structure. Hence they are referred to as locally recurrent–globally feedforward (LRGF) architectures. A general LRGF architecture is shown in Figure 5.10. This architecture allows for the dynamic synapses both within the input (represented by H_1, \dots, H_M) and the output feedback (represented by H_{FB}), hence comprising some of the aforementioned schemes.

The Elman network is a recurrent network with a hidden layer, a simple example of which is shown in Figure 5.11. This network consists of an MLP with an additional input which consists of delayed state space variables of the network. Even though it contains feedback connections, it is treated as a kind of MLP. The network shown in

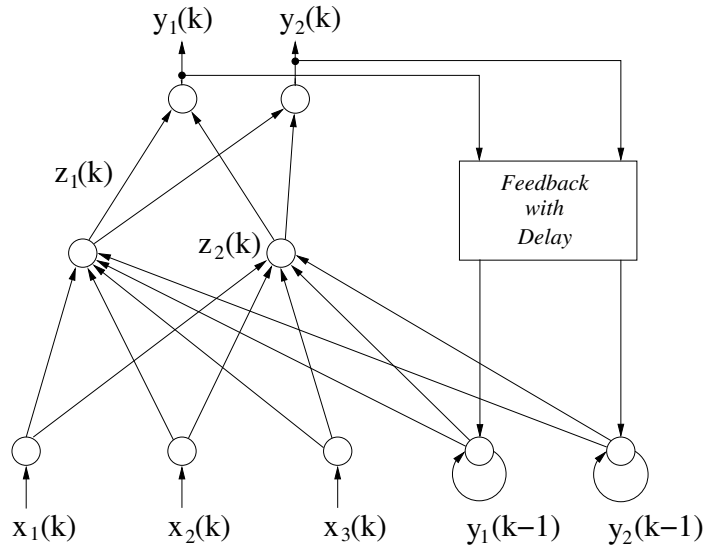


Figure 5.12 An example of Jordan recurrent neural network

Figure 5.12 is an example of the Jordan network. It consists of a multilayer perceptron with one hidden layer and a feedback loop from the output layer to an additional input called the context layer. In the context layer, there are self-recurrent loops. Both Jordan and Elman networks are structurally locally recurrent globally feedforward (LRGF), and are rather limited in including past information.

A network with a rich representation of past outputs, which will be extensively considered in this book, is a fully connected recurrent neural network, known as the Williams-Zipser network (Williams and Zipser 1989a), shown in Figure 5.13. We give a detailed introduction to this architecture. This network consists of three layers: the input layer, the processing layer and the output layer. For each neuron i , $i = 1, 2, \dots, N$, the elements u_j , $j = 1, 2, \dots, p + N + 1$, of the input vector to a neuron u (5.31), are weighted, then summed to produce an internal activation function of a neuron v (5.30), which is finally fed through a nonlinear activation function Φ (5.28), to form the output of the i th neuron y_i (5.29). The function Φ is a monotonically increasing sigmoid function with slope β , as for instance the logistic function,

$$\Phi(v) = \frac{1}{1 + e^{-\beta v}}. \tag{5.28}$$

At the time instant k , for the i th neuron, its weights form a $(p + N + 1) \times 1$ dimensional weight vector $\mathbf{w}_i^T(k) = [w_{i,1}(k), \dots, w_{i,p+N+1}(k)]$, where p is the number of external inputs, N is the number of feedback connections and $(\cdot)^T$ denotes the vector transpose operation. One additional element of the weight vector \mathbf{w} is the bias input weight. The feedback consists of the delayed output signals of the RNN. The following equations fully describe the RNN from Fig-

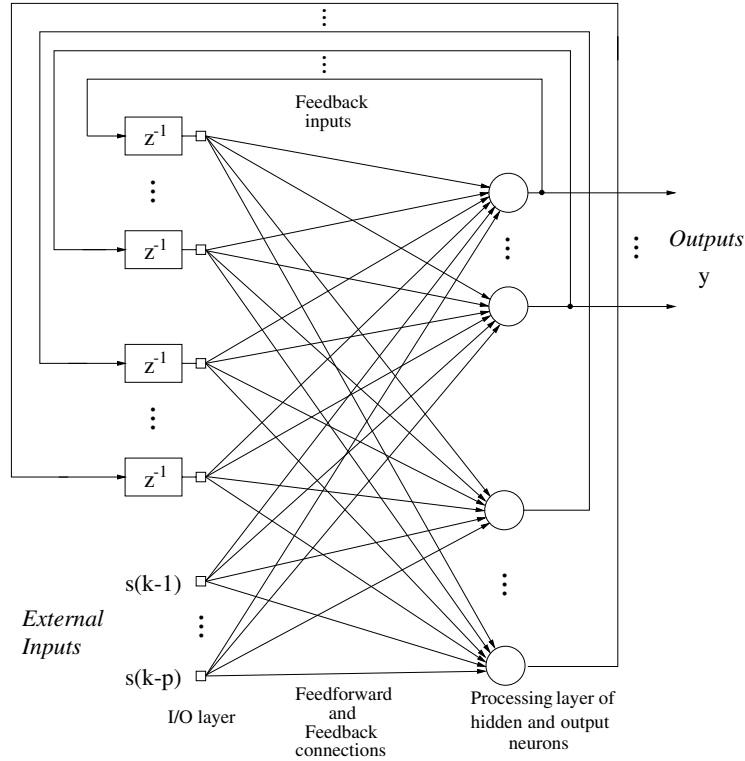


Figure 5.13 A fully connected recurrent neural network

ure 5.13,

$$y_i(k) = \Phi(v_i(k)), \quad i = 1, 2, \dots, N, \quad (5.29)$$

$$v_i(k) = \sum_{l=1}^{p+N+1} w_{i,l}(k)u_l(k), \quad (5.30)$$

$$\mathbf{u}_i^T(k) = [s(k-1), \dots, s(k-p), 1, y_1(k-1), y_2(k-1), \dots, y_N(k-1)], \quad (5.31)$$

where the $(p + N + 1) \times 1$ dimensional vector \mathbf{u} comprises both the external and feedback inputs to a neuron, as well as the unity valued constant bias input.

5.9 Hybrid Neural Network Architectures

These networks consist of a cascade of a neural network and a linear adaptive filter. If a neural network is considered as a complex adaptable nonlinearity, then hybrid neural networks resemble Wiener and Hammerstein stochastic models. An example of these networks is given in Khalaf and Nakayama (1999), for prediction of noisy time series. A neural subpredictor is cascaded with a linear FIR predictor, hence making a hybrid predictor. The block diagram of this type of neural network architecture is

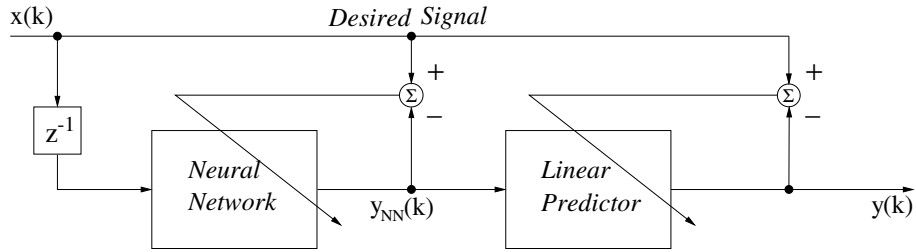


Figure 5.14 A hybrid neural predictor

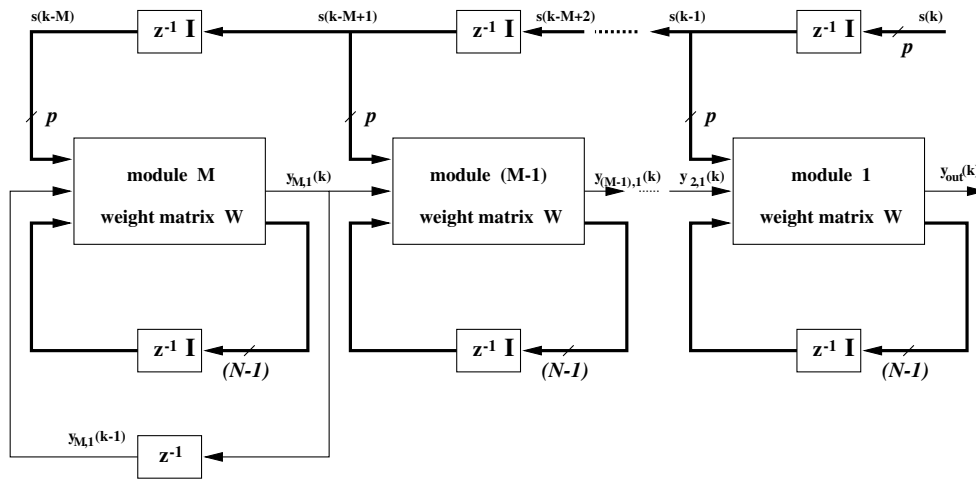


Figure 5.15 Pipelined recurrent neural network

given in Figure 5.14. The neural network from Figure 5.14 can be either a feedforward neural network or a recurrent neural network.

Another example of hybrid structures is the so-called pipelined recurrent neural network (PRNN), introduced by Haykin and Li (1995) and shown in Figure 5.15. It consists of a modular nested structure of small-scale fully connected recurrent neural networks and a cascaded FIR adaptive filter. In the PRNN configuration, the M modules, which are FCRNNs, are connected as shown in Figure 5.15. The cascaded linear filter is omitted. The description of this network follows the approach from Mandic *et al.* (1998) and Baltersee and Chambers (1998). The uppermost module of the PRNN, denoted by M , is simply an FCRNN, whereas in modules $(M - 1, \dots, 1)$, the only difference is that the feedback signal of the output neuron within module m , denoted by $y_{m,1}$, $m = 1, \dots, M - 1$, is replaced with the appropriate output signal $y_{m+1,1}$, $m = 1, \dots, M - 1$, from its left neighbour module $m + 1$. The $(p \times 1)$ -dimensional external signal vector $\mathbf{s}^T(k) = [s(k), \dots, s(k - p + 1)]$ is delayed by m time steps ($z^{-m}\mathbf{I}$) before feeding the module m , where z^{-m} , $m = 1, \dots, M$, denotes the m -step time delay operator and \mathbf{I} is the $(p \times p)$ -dimensional identity matrix. The weight vectors \mathbf{w}_n of each neuron n , are embodied in an $(p + N + 1) \times N$ dimensional weight matrix $\mathbf{W}(k) = [\mathbf{w}_1(k), \dots, \mathbf{w}_N(k)]$, with N being the number of neurons in

each module. All the modules operate using the same weight matrix \mathbf{W} . The overall output signal of the PRNN is $y_{\text{out}}(k) = y_{1,1}(k)$, i.e. the output of the first neuron of the first module. A full mathematical description of the PRNN is given in the following equations:

$$y_{i,n}(k) = \Phi(v_{i,n}(k)), \quad (5.32)$$

$$v_{i,n}(k) = \sum_{l=1}^{p+N+1} w_{n,l}(k)u_{i,l}(k), \quad (5.33)$$

$$\mathbf{u}_i^T(k) = [s(k-i), \dots, s(k-i-p+1), 1, \\ y_{i+1,1}(k), y_{i,2}(k-1), \dots, y_{i,N}(k-1)] \quad \text{for } 1 \leq i \leq M-1, \quad (5.34)$$

$$\mathbf{u}_M^T(k) = [s(k-M), \dots, s(k-M-p+1), 1, \\ y_{M,1}(k-1), y_{M,2}(k-1), \dots, y_{M,N}(k-1)] \quad \text{for } i = M. \quad (5.35)$$

At the time step k for each module i , $i = 1, \dots, M$, the one-step forward prediction error $e_i(k)$ associated with a module is then defined as a difference between the desired response of that module $s(k-i+1)$, which is actually the next incoming sample of the external input signal, and the actual output of the i th module $y_{i,1}(k)$ of the PRNN, i.e.

$$e_i(k) = s(k-i+1) - y_{i,1}(k), \quad i = 1, \dots, M. \quad (5.36)$$

Thus, the overall cost function of the PRNN becomes a weighted sum of all squared error signals,

$$E(k) = \sum_{i=1}^M \lambda^{i-1} e_i^2(k), \quad (5.37)$$

where $e_i(k)$ is defined in Equation (5.36) and λ , $\lambda \in (0, 1]$, is a forgetting factor.

Other architectures combining linear and nonlinear blocks include the so-called ‘sandwich’ structure which was used for estimation of Hammerstein systems (Ibnkahla *et al.* 1998). The architecture used was a linear–nonlinear–linear combination.

5.10 Nonlinear ARMA Models and Recurrent Networks

A general NARMA(p, q) recurrent network model can be expressed as (Chang and Hu 1997)

$$\hat{x}(k) = \Phi \left(\sum_{i=1}^p w_{1,i}(k)x(k-i) + w_{1,p+1}(k) + \sum_{j=p+2}^{p+q+1} w_{1,j}(k)\hat{e}(k+j-2-p-q) \right. \\ \left. + \sum_{l=p+q+2}^{p+q+N} w_{1,l}(k)y_{l-p-q}(k-1) \right). \quad (5.38)$$

A realisation of this model is shown in Figure 5.16. The NARMA(p, q) scheme shown in Figure 5.16 is a common Williams–Zipser type recurrent neural network, which

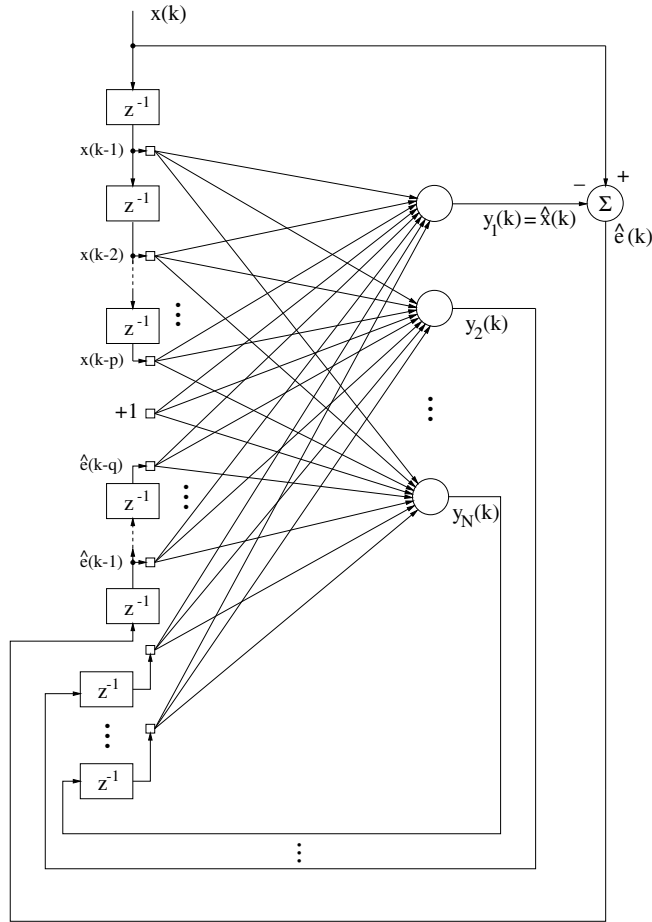


Figure 5.16 Alternative recurrent NARMA(p, q) network

consists of only two layers, the output layer of output and hidden neurons y_1, \dots, y_N , and the input layer of feedforward and feedback signals

$$x(k-1), \dots, x(k-p), +1, \hat{e}(k-1), \dots, \hat{e}(k-q), y_2(k-1), \dots, y_N(k-1).$$

The nonlinearity in this case is determined by both the nonlinearity associated with the output neuron of the recurrent neural network and nonlinearities in hidden neurons.

The inputs to this network, given in (5.38), however, comprise the prediction error terms (residuals) $\hat{e}(k-1), \dots, \hat{e}(k-q)$, which make the learning in such networks difficult. Namely, the well-known real-time recurrent learning (RTRL) algorithm (Haykin 1994; Williams and Zipser 1989a) was derived to minimise the instantaneous squared prediction error $\hat{e}(k)$, and hence cannot be applied directly to the RNN realisations of the NARMA(p, q) network, as shown above, since the inputs to the network comprise the delayed prediction error terms $\{\hat{e}\}$. It is therefore desirable to find another

equivalent representation of the NARMA(p, q) network, which would be more suited for the RTRL-based learning.

If, for the sake of clarity, we denote the predicted values \hat{x} by y , i.e. to match the notation common in RNNs with the NARMA(p, q) theory, and have $y_1(k) = \hat{x}(k)$, and keep the symbol x for the exact values of the input signal being predicted, the NARMA network from (5.38), can be approximated further as (Connor 1994)

$$\begin{aligned} y_1(k) &= h(x(k-1), x(k-2), \dots, x(k-p), \hat{e}(k-1), \hat{e}(k-2), \dots, \hat{e}(k-q)) \\ &= h(x(k-1), x(k-2), \dots, x(k-p), (x(k-1) - y_1(k-1)), \dots \\ &\quad \dots, (x(k-q) - y_1(k-q))) \\ &= H(x(k-1), x(k-2), \dots, x(k-p), y_1(k-1), \dots, y_1(k-q)). \end{aligned} \quad (5.39)$$

In that case, the scheme shown in Figure 5.16 should be redrawn, remaining topologically the same, with y_1 replacing the corresponding \hat{e} terms among the inputs to the network.

On the other hand, the alternative expression for the conditional mean predictor, depicted in Figure 5.16 can be written as

$$\hat{x}(k) = \Phi \left(\sum_{i=1}^p w_{1,i}(k)x(k-i) + w_{1,p+1}(k) + \sum_{j=p+2}^{p+q+1} w_{1,j}(k)\hat{x}(k+j-2-p-q) + \sum_{l=p+q+2}^{p+q+N} w_{1,l}(k)y_{l-p-q}(k-1) \right) \quad (5.40)$$

or, bearing in mind (5.39), the notation used earlier (Haykin and Li 1995; Mandic *et al.* 1998) for the examples on the prediction of speech, i.e. $x(k) = s(k)$, and that $y_1(k) = \hat{s}(k)$,

$$\hat{s}(k) = \Phi \left(\sum_{i=1}^p w_{1,i}(k)s(k-i) + w_{1,p+1}(k) + \sum_{j=p+2}^{p+q+1} w_{1,j}(k)y_1(k+j-2-p-q) + \sum_{l=p+q+2}^{p+q+N} w_{1,l}(k)y_{l-p-q}(k-1) \right), \quad (5.41)$$

which is the common RNN lookalike notation. This scheme offers a simpler solution to the NARMA(p, q) problem, as compared to the previous one, since the only nonlinear function used is the activation function of a neuron Φ , while the set of signals being processed is the same as in the previous scheme. Furthermore, the scheme given in (5.41) and depicted in Figure 5.17 resembles the basic ARMA structure.

Li (1992) has shown that the recurrent network of (5.41) with a sufficiently large number of neurons and appropriate weights can be found by performing the RTRL algorithm such that the sum of squared prediction errors $E < \delta$ for an arbitrary $\delta > 0$. In other words, $\|\mathbf{s} - \hat{\mathbf{s}}\|_D < \delta$, where $\|\cdot\|_D$ denotes the \mathcal{L}_2 norm with respect to the training set D . Moreover, this scheme, shown also in Figure 5.17, fits into the well-known learning strategies, such as the RTRL algorithm, which recommends this

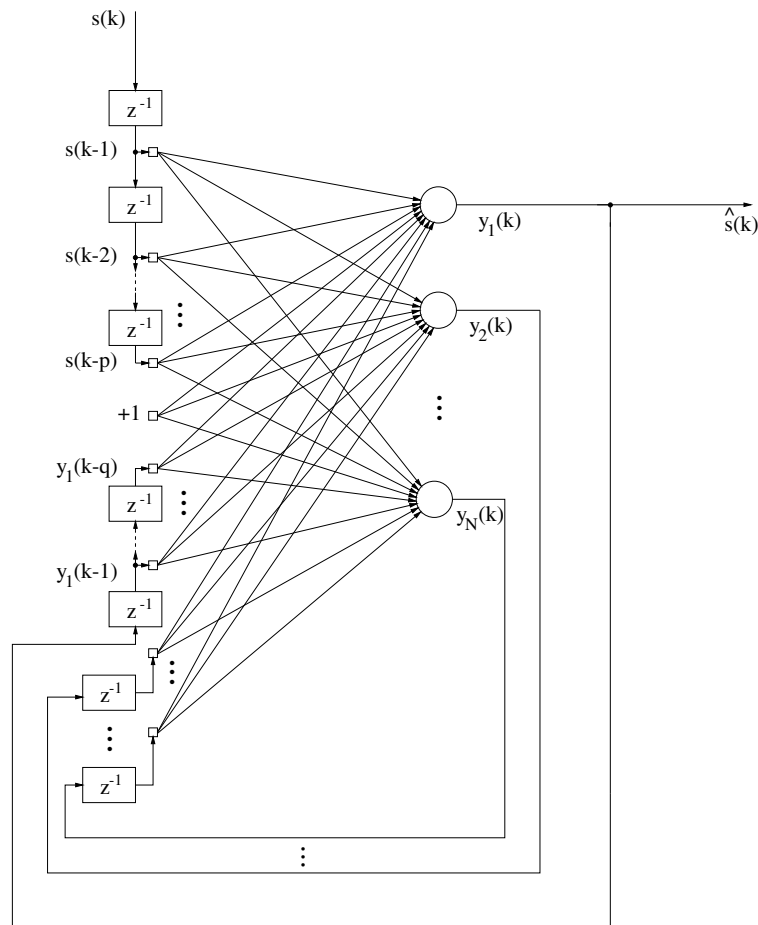


Figure 5.17 Recurrent NARMA(p, q) implementation of prediction model

scheme for NARMA/NARMAX nonlinear prediction applications (Baldi and Atiya 1994; Draye *et al.* 1996; Kosmatopoulos *et al.* 1995; McDonnell and Waagen 1994; Nerrand *et al.* 1994; Wu and Niranjana 1994).

5.11 Summary

A review of recurrent neural network architectures in the fields of nonlinear dynamical modelling, system identification, control, signal processing and forecasting has been provided. A relationship between neural network models and NARMA/NARMAX models, as well as Wiener and Hammerstein structures has been established. Particular attention has been devoted to the fully connected recurrent neural network and its use in NARMA/NARMAX modelling has been highlighted.

6

Neural Networks as Nonlinear Adaptive Filters

6.1 Perspective

Neural networks, in particular recurrent neural networks, are cast into the framework of nonlinear adaptive filters. In this context, the relation between recurrent neural networks and polynomial filters is first established. Learning strategies and algorithms are then developed for neural adaptive system identifiers and predictors. Finally, issues concerning the choice of a neural architecture with respect to the bias and variance of the prediction performance are discussed.

6.2 Introduction

Representation of nonlinear systems in terms of NARMA/NARMAX models has been discussed at length in the work of Billings and others (Billings 1980; Chen and Billings 1989; Connor 1994; Nerrand *et al.* 1994). Some cognitive aspects of neural nonlinear filters are provided in Maass and Sontag (2000). Pearson (1995), in his article on nonlinear input–output modelling, shows that block oriented nonlinear models are a subset of the class of Volterra models. So, for instance, the Hammerstein model, which consists of a static nonlinearity $f(\cdot)$ applied at the output of a linear dynamical system described by its z -domain transfer function $H(z)$, can be represented¹ by the Volterra series.

In the previous chapter, we have shown that neural networks, be they feedforward or recurrent, cannot generate time delays of an order higher than the dimension of the input to the network. Another important feature is the capability to generate subharmonics in the spectrum of the output of a nonlinear neural filter (Pearson 1995). The key property for generating subharmonics in nonlinear systems is recursion, hence, recurrent neural networks are necessary for their generation. Notice that, as

¹ Under the condition that the function f is analytic, and that the Volterra series can be thought of as a generalised Taylor series expansion, then the coefficients of the model (6.2) that do not vanish are $h_{i,j,\dots,z} \neq 0 \Leftrightarrow i = j = \dots = z$.

pointed out in Pearson (1995), block-stochastic models are, generally speaking, not suitable for this application.

In Hakim *et al.* (1991), by using the Weierstrass polynomial expansion theorem, the relation between neural networks and Volterra series is established, which is then extended to a more general case and to continuous functions that cannot be expanded via a Taylor series expansion.² Both feedforward and recurrent networks are characterised by means of a Volterra series and vice versa.

Neural networks are often referred to as ‘adaptive neural networks’. As already shown, adaptive filters and neural networks are formally equivalent, and neural networks, employed as nonlinear adaptive filters, are generalisations of linear adaptive filters. However, in neural network applications, they have been used mostly in such a way that the network is first trained on a particular training set and subsequently used. This approach is not an online adaptive approach, which is in contrast with linear adaptive filters, which undergo continual adaptation.

Two groups of learning techniques are used for training recurrent neural networks: a direct gradient computation technique (used in nonlinear adaptive filtering) and a recurrent backpropagation technique (commonly used in neural networks for offline applications). The real-time recurrent learning (RTRL) algorithm (Williams and Zipser 1989a) is a technique which uses direct gradient computation, and is used if the network coefficients change slowly with time. This technique is essentially an LMS learning algorithm for a nonlinear IIR filter. It should be noticed that, with the same computation time, it might be possible to unfold the recurrent neural network into the corresponding feedforward counterparts and hence to train it by backpropagation. The backpropagation through time (BPTT) algorithm is such a technique (Werbos 1990).

Some of the benefits involved with neural networks as nonlinear adaptive filters are that no assumptions concerning Markov property, Gaussian distribution or additive measurement noise are necessary (Lo 1994). A neural filter would be a suitable choice even if mathematical models of the input process and measurement noise are not known (black box modelling).

6.3 Overview

We start with the relationship between Volterra and bilinear filters and neural networks. Recurrent neural networks are then considered as nonlinear adaptive filters and neural architectures for this case are analysed. Learning algorithms for online training of recurrent neural networks are developed inductively, starting from corresponding algorithms for linear adaptive IIR filters. Some issues concerning the problem of vanishing gradient and bias/variance dilemma are finally addressed.

6.4 Neural Networks and Polynomial Filters

It has been shown in Chapter 5 that a small-scale neural network can represent high-order nonlinear systems, whereas a large number of terms are required for an equiv-

² For instance nonsmooth functions, such as $|x|$.

alent Volterra series representation. For instance, as already shown, after performing a Taylor series expansion for the output of a neural network depicted in Figure 5.3, with input signals $u(k-1)$ and $u(k-2)$, we obtain

$$y(k) = c_0 + c_1u(k-1) + c_2u(k-2) + c_3u^2(k-1) + c_4u^2(k-2) \\ + c_5u(k-1)u(k-2) + c_6u^3(k-1) + c_7u^3(k-2) + \dots, \quad (6.1)$$

which has the form of a general Volterra series, given by

$$y(k) = h_0 + \sum_{i=0}^N h_1(i)x(k-i) + \sum_{i=0}^N \sum_{j=0}^N h_2(i,j)x(k-i)x(k-j) + \dots, \quad (6.2)$$

Representation by a neural network is therefore more compact. As pointed out in Schetzen (1981), Volterra series are not suitable for modelling saturation type nonlinear functions and systems with nonlinearities of a high order, since they require a very large number of terms for an acceptable representation. The order of Volterra series and complexity of kernels $h(\cdot)$ increase exponentially with the order of the delay in system (6.2). This problem restricts practical applications of Volterra series to small-scale systems.

Nonlinear system identification, on the other hand, has been traditionally based upon the Kolmogorov approximation theorem (neural network existence theorem), which states that a neural network with a hidden layer can approximate an arbitrary nonlinear system. Kolmogorov's theorem, however, is not that relevant in the context of networks for learning (Girosi and Poggio 1989b). The problem is that inner functions in Kolmogorov's formula (4.1), although continuous, have to be highly non-smooth. Following the analysis from Chapter 5, it is straightforward that multilayered and recurrent neural networks have the ability to approximate an arbitrary nonlinear system, whereas Volterra series fail even for simple saturation elements.

Another convenient form of nonlinear system is the bilinear (truncated Volterra) system described by

$$y(k) = \sum_{j=1}^{N-1} c_j y(k-j) + \sum_{i=0}^{N-1} \sum_{j=1}^{N-1} b_{i,j} y(k-j)x(k-i) + \sum_{i=0}^{N-1} a_i x(k-i). \quad (6.3)$$

Despite its simplicity, this is a powerful nonlinear model and a large class of nonlinear systems (including Volterra systems) can be approximated arbitrarily well using this model. Its functional dependence (6.3) shows that it belongs to a class of general recursive nonlinear models. A recurrent neural network that realises a simple bilinear model is depicted in Figure 6.1. As seen from Figure 6.1, multiplicative input nodes (denoted by ' \times ') have to be introduced to represent the bilinear model. Bias terms are omitted and the chosen neuron is linear.

Example 6.4.1. Show that the recurrent network shown in Figure 6.1 realises a bilinear model. Also show that this network can be described in terms of NARMAX models.

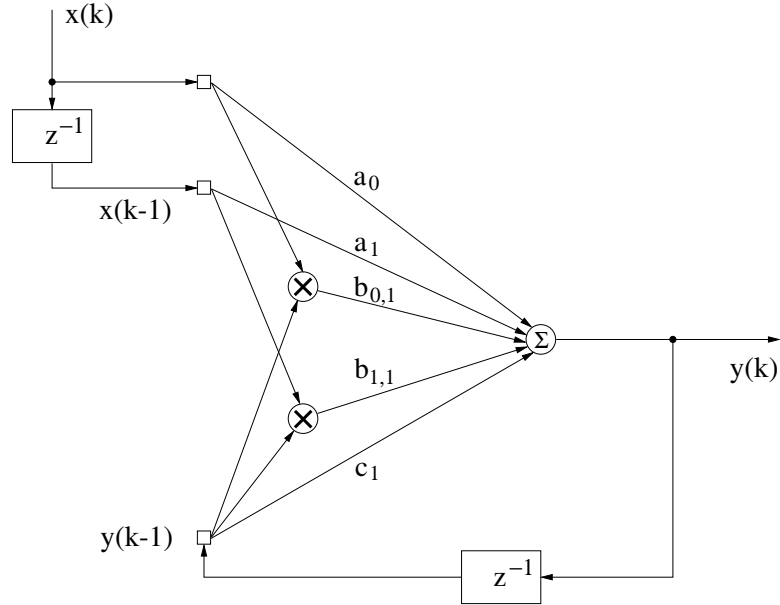


Figure 6.1 Recurrent neural network representation of the bilinear model

Solution. The functional description of the recurrent network depicted in Figure 6.1 is given by

$$y(k) = c_1 y(k-1) + b_{0,1} x(k) y(k-1) + b_{1,1} x(k-1) y(k-1) + a_0 x(k) + a_1 x(k-1), \quad (6.4)$$

which belongs to the class of bilinear models (6.3). The functional description of the network from Figure 6.1 can also be expressed as

$$y(k) = F(y(k-1), x(k), x(k-1)), \quad (6.5)$$

which is a NARMA representation of model (6.4). \square

Example 6.4.1 confirms the duality between Volterra, bilinear, NARMA/NARMAX and recurrent neural models. To further establish the connection between Volterra series and a neural network, let us express the activation potential of nodes of the network as

$$\text{net}_i(k) = \sum_{j=0}^M w_{i,j} x(k-j), \quad (6.6)$$

where $\text{net}_i(k)$ is the activation potential of the i th hidden neuron, $w_{i,j}$ are weights and $x(k-j)$ are inputs to the network. If the nonlinear activation functions of neurons are expressed via an L th-order polynomial expansion³ as

$$\Phi(\text{net}_i(k)) = \sum_{l=0}^L \xi_{il} \text{net}_i^l(k), \quad (6.7)$$

³ Using the Weierstrass theorem, this expansion can be arbitrarily accurate. However, in practice we resort to a moderate order of this polynomial expansion.

then the neural model described in (6.6) and (6.7) can be related to the Volterra model (6.2). The actual relationship is rather complicated, and Volterra kernels are expressed as sums of products of the weights from input to hidden units, weights associated with the output neuron, and coefficients ξ_{il} from (6.7). Chon *et al.* (1998) have used this kind of relationship to compare the Volterra and neural approach when applied to processing of biomedical signals.

Hence, to avoid the difficulty of excessive computation associated with Volterra series, an input–output relationship of a nonlinear predictor that computes the output in terms of past inputs and outputs may be introduced as⁴

$$\hat{y}(k) = F(y(k-1), \dots, y(k-N), u(k-1), \dots, u(k-M)), \quad (6.8)$$

where $F(\cdot)$ is some nonlinear function. The function F may change for different input variables or for different regions of interest. A NARMAX model may therefore be a correct representation only in a region around some operating point. Leontaritis and Billings (1985) rigorously proved that a discrete time nonlinear time invariant system can always be represented by model (6.8) in the vicinity of an equilibrium point provided that

- the response function of the system is finitely realisable, and
- it is possible to linearise the system around the chosen equilibrium point.

As already shown, some of the other frequently used models, such as the bilinear polynomial filter, given by (6.3), are obviously cases of a simple NARMAX model.

6.5 Neural Networks and Nonlinear Adaptive Filters

To perform nonlinear adaptive filtering, tracking and system identification of nonlinear time-varying systems, there is a need to introduce dynamics in neural networks. These dynamics can be introduced via recurrent neural networks, which are the focus of this book.

The design of linear filters is conveniently specified by a frequency response which we would like to match. In the nonlinear case, however, since a transfer function of a nonlinear filter is not available in the frequency domain, one has to resort to different techniques. For instance, the design of nonlinear filters may be thought of as a nonlinear constrained optimisation problem in Fock space (deFigueiredo 1997).

In a recurrent neural network architecture, the feedback brings the delayed outputs from hidden and output neurons back into the network input vector $\mathbf{u}(k)$, as shown in Figure 5.13. Due to gradient learning algorithms, which are sequential, these delayed outputs of neurons represent filtered data from the previous discrete time instant. Due to this ‘memory’, at each time instant, the network is presented with the raw,

⁴ As already shown, this model is referred to as the NARMAX model (nonlinear ARMAX), since it resembles the linear model

$$\hat{y}(k) = a_0 + \sum_{j=1}^N a_j y(k-j) + \sum_{i=1}^M b_i u(k-i).$$

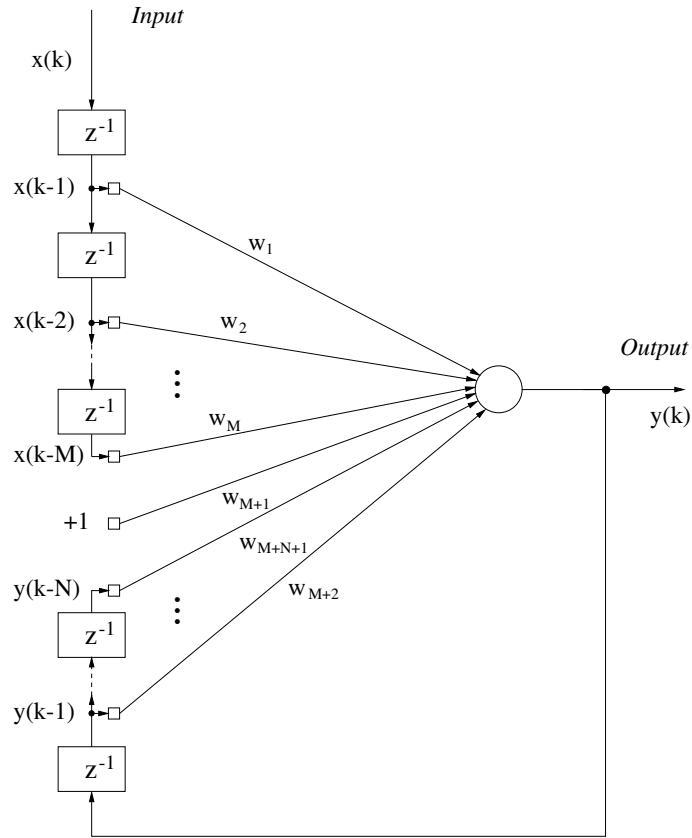


Figure 6.2 NARMA recurrent perceptron

possibly noisy, external input data $s(k), s(k-1), \dots, s(k-M)$ from Figure 5.13 and Equation (5.31), and filtered data $y_1(k-1), \dots, y_N(k-1)$ from the network output. Intuitively, this filtered input history helps to improve the processing performance of recurrent neural networks, as compared with feedforward networks. Notice that the history of past outputs is never presented to the learning algorithm for feedforward networks. Therefore, a recurrent neural network should be able to process signals corrupted by additive noise even in the case when the noise distribution is varying over time.

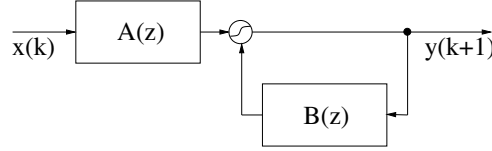
On the other hand, a nonlinear dynamical system can be described by

$$\mathbf{u}(k+1) = \Phi(\mathbf{u}(k)) \quad (6.9)$$

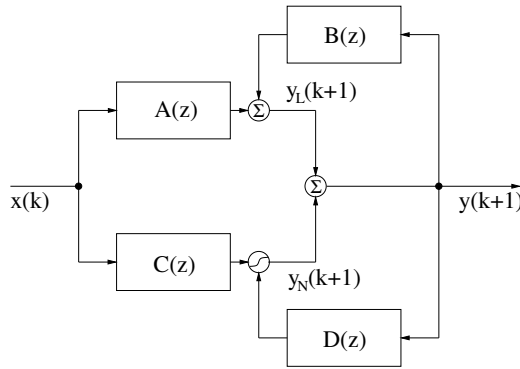
with an observation process

$$y(k) = \varphi(\mathbf{u}(k)) + \epsilon(k), \quad (6.10)$$

where $\epsilon(k)$ is observation noise (Haykin and Principe 1998). Takens' embedding theorem (Takens 1981) states that the geometric structure of system (6.9) can be recovered



(a) A recurrent nonlinear neural filter



(b) A recurrent linear/nonlinear neural filter structure

Figure 6.3 Nonlinear IIR filter structures

from the sequence $\{y(k)\}$ in a D -dimensional space spanned by⁵

$$\mathbf{y}(k) = [y(k), y(k-1), \dots, y(k-(D-1))] \tag{6.11}$$

provided that $D \geq 2d + 1$, where d is the dimension of the state space of system (6.9). Therefore, one advantage of NARMA models over FIR models is the parsimony of NARMA models, since an upper bound on the order of a NARMA model is twice the order of the state (phase) space of the system being analysed.

The simplest recurrent neural network architecture is a recurrent perceptron, shown in Figure 6.2. This is a simple, yet effective architecture. The equations which describe the recurrent perceptron shown in Figure 6.2 are

$$\left. \begin{aligned} y(k) &= \Phi(v(k)), \\ v(k) &= \mathbf{u}^T(k)\mathbf{w}(k), \end{aligned} \right\} \tag{6.12}$$

where $\mathbf{u}(k) = [x(k-1), \dots, x(k-M), 1, y(k-1), \dots, y(k-N)]^T$ is the input vector, $\mathbf{w}(k) = [w_1(k), \dots, w_{M+N+1}(k)]^T$ is the weight vector and $(\cdot)^T$ denotes the vector transpose operator.

⁵ Model (6.11) is in fact a NAR/NARMAX model.

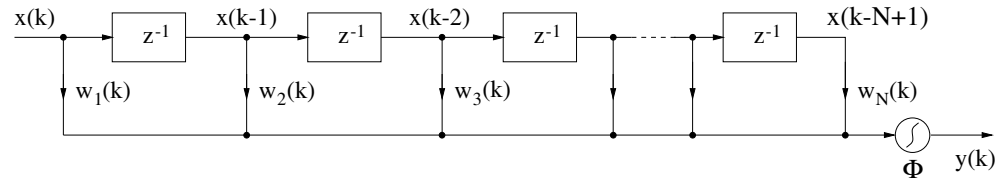


Figure 6.4 A simple nonlinear adaptive filter

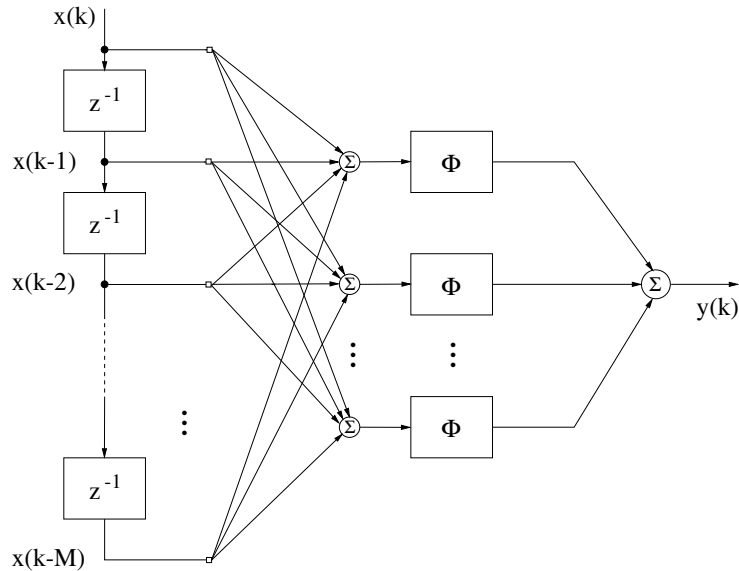


Figure 6.5 Fully connected feedforward neural filter

A recurrent perceptron is a recursive adaptive filter with an arbitrary output function as shown in Figure 6.3. Figure 6.3(a) shows the recurrent perceptron structure as a nonlinear infinite impulse response (IIR) filter. Figure 6.3(b) depicts the parallel linear/nonlinear structure, which is one of the possible architectures. These structures stem directly from IIR filters and are described in McDonnell and Waagen (1994), Connor (1994) and Nerrand *et al.* (1994). Here, $A(z)$, $B(z)$, $C(z)$ and $D(z)$ denote the z -domain linear transfer functions. The general structure of a fully connected, multilayer neural feedforward filter is shown in Figure 6.5 and represents a generalisation of a simple nonlinear feedforward perceptron with dynamic synapses, shown in Figure 6.4. This structure consists of an input layer, layer of hidden neurons and an output layer. Although the output neuron shown in Figure 6.5 is linear, it could be nonlinear. In that case, attention should be paid that the dynamic ranges of the input signal and output neuron match.

Another generalisation of a fully connected recurrent neural filter is shown in Figure 6.6. This network consists of nonlinear neural filters as depicted in Figure 6.5, applied to both the input and output signal, the outputs of which are summed together. This is a fairly general structure which resembles the architecture of a lin-

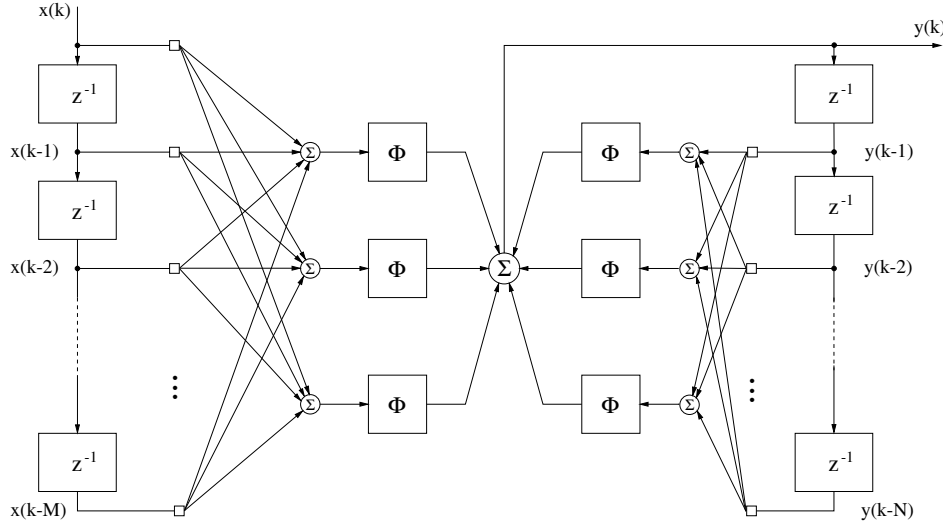


Figure 6.6 Fully connected recurrent neural filter

ear IIR filter and is the extension of the NARMAX recurrent perceptron shown in Figure 6.2.

Narendra and Parthasarathy (1990) provide deep insight into structures of neural networks for identification of nonlinear dynamical systems. Due to the duality between system identification and prediction, the same architectures are suitable for prediction applications. From Figures 6.3–6.6, we can identify four general architectures of neural networks for prediction and system identification. These architectures come as combinations of linear/nonlinear parts from the architecture shown in Figure 6.6, and for the nonlinear prediction configuration are specified as follows.

- (i) The output $y(k)$ is a linear function of previous outputs and a nonlinear function of previous inputs, given by

$$y(k) = \sum_{j=1}^N a_j(k)y(k-j) + F(u(k-1), u(k-2), \dots, u(k-M)), \quad (6.13)$$

where $F(\cdot)$ is some nonlinear function. This architecture is shown in Figure 6.7(a).

- (ii) The output $y(k)$ is a nonlinear function of past outputs and a linear function of past inputs, given by

$$y(k) = F(y(k-1), y(k-2), \dots, y(k-N)) + \sum_{i=1}^M b_i(k)u(k-i). \quad (6.14)$$

This architecture is depicted in Figure 6.7(b).

- (iii) The output $y(k)$ is a nonlinear function of both past inputs and outputs. The functional relationship between the past inputs and outputs can be expressed

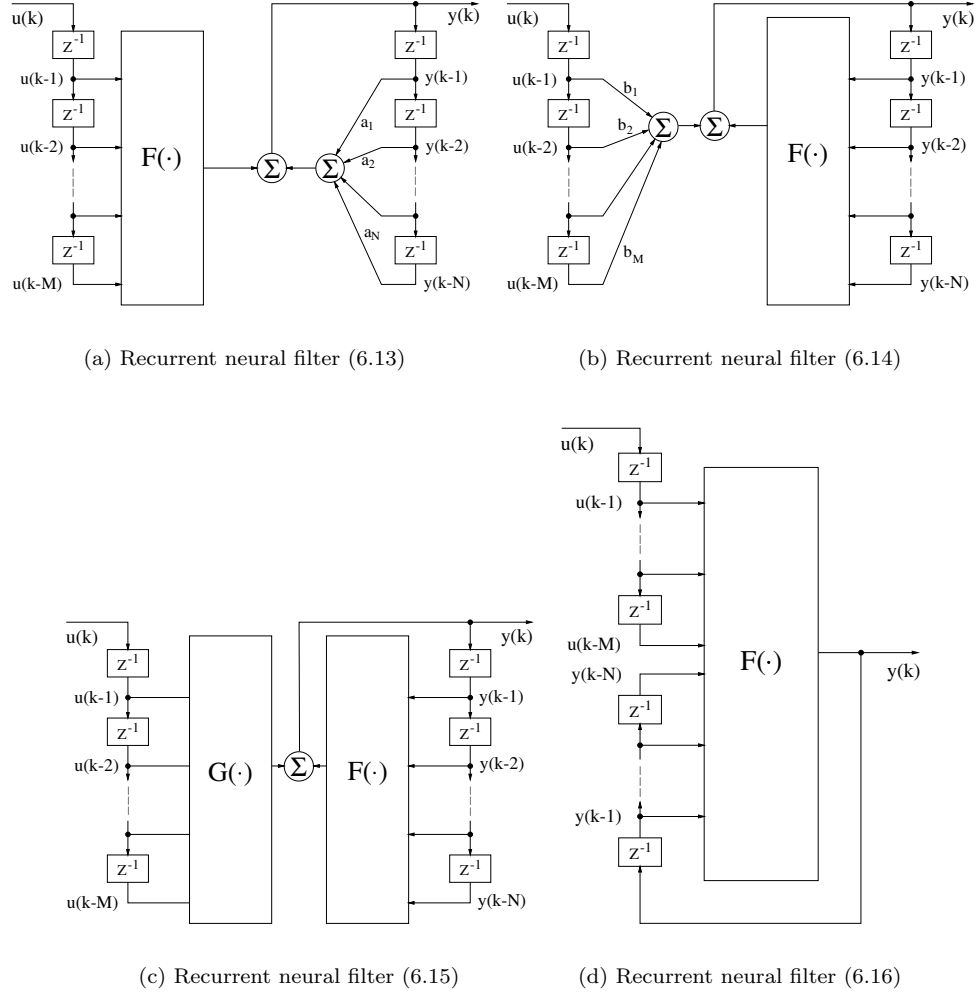


Figure 6.7 Architectures of recurrent neural networks as nonlinear adaptive filters

in a separable manner as

$$y(k) = F(y(k-1), \dots, y(k-N)) + G(u(k-1), \dots, u(k-M)). \quad (6.15)$$

This architecture is depicted in Figure 6.7(c).

(iv) The output $y(k)$ is a nonlinear function of past inputs and outputs, as

$$y(k) = F(y(k-1), \dots, y(k-N), u(k-1), \dots, u(k-M)). \quad (6.16)$$

This architecture is depicted in Figure 6.7(d) and is most general.

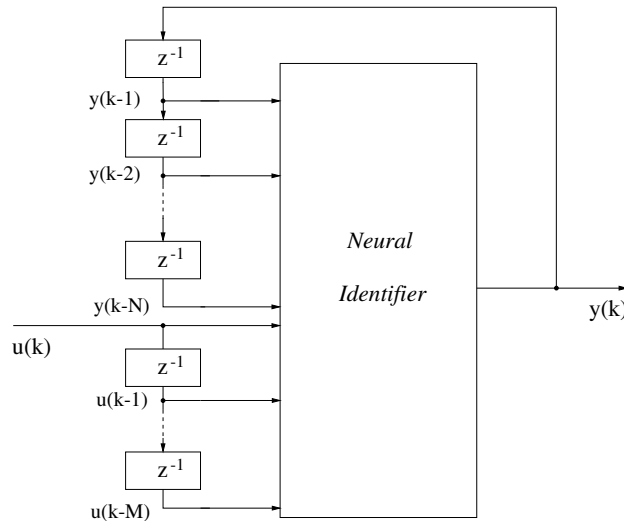


Figure 6.8 NARMA type neural identifier

6.6 Training Algorithms for Recurrent Neural Networks

A natural error criterion, upon which the training of recurrent neural networks is based, is in the form of the accumulated squared prediction error over the whole dataset, given by

$$E(k) = \frac{1}{M + 1} \sum_{m=0}^M \lambda(m)e^2(k - m), \tag{6.17}$$

where M is the length of the dataset and $\lambda(m)$ are weights associated with a particular instantaneous error $e(k - m)$. For stationary signals, usually $\lambda(m) = 1, m = 1, 2, \dots, M$, whereas in the nonstationary case, since the statistics change over time, it is unreasonable to take into account the whole previous history of the errors. For this case, a forgetting mechanism is usually employed, whereby $0 < \lambda(m) < 1$. Since many real-world signals are nonstationary, online learning algorithms commonly use the squared instantaneous error as an error criterion, i.e.

$$E(k) = \frac{1}{2}e^2(k). \tag{6.18}$$

Here, the coefficient $\frac{1}{2}$ is included for convenience in the derivation of the algorithms.

6.7 Learning Strategies for a Neural Predictor/Identifier

A NARMA/NARMAX type neural identifier is depicted in Figure 6.8. When considering a neural predictor, the only difference is the position of the neural module within the system structure, as shown in Chapter 2. There are two main training strategies to estimate the weights of the neural network shown in Figure 6.8. In the first approach, the links between the real system and the neural identifier are as depicted in Figure 6.9. During training, the configuration shown in Figure 6.9 can be

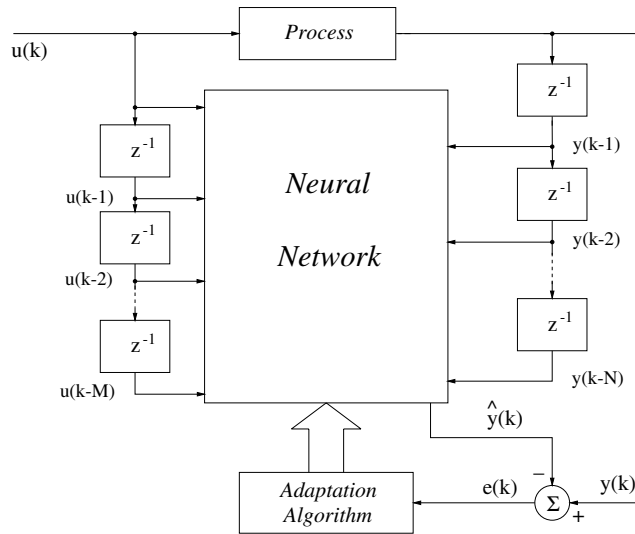


Figure 6.9 The nonlinear series-parallel (teacher forcing) learning configuration

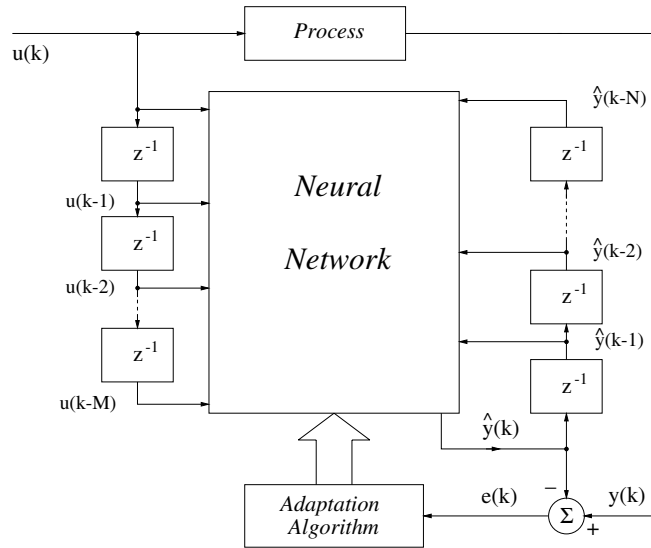


Figure 6.10 The nonlinear parallel (supervised) learning configuration

described by

$$\hat{y}(k) = f(u(k), \dots, u(k - M), y(k - 1), \dots, y(k - N)), \tag{6.19}$$

which is referred to as the *nonlinear series-parallel model* (Alippi and Piuri 1996; Qin *et al.* 1992). In this configuration, the desired signal $y(k)$ is presented to the network, which produces biased estimates (Narendra 1996).

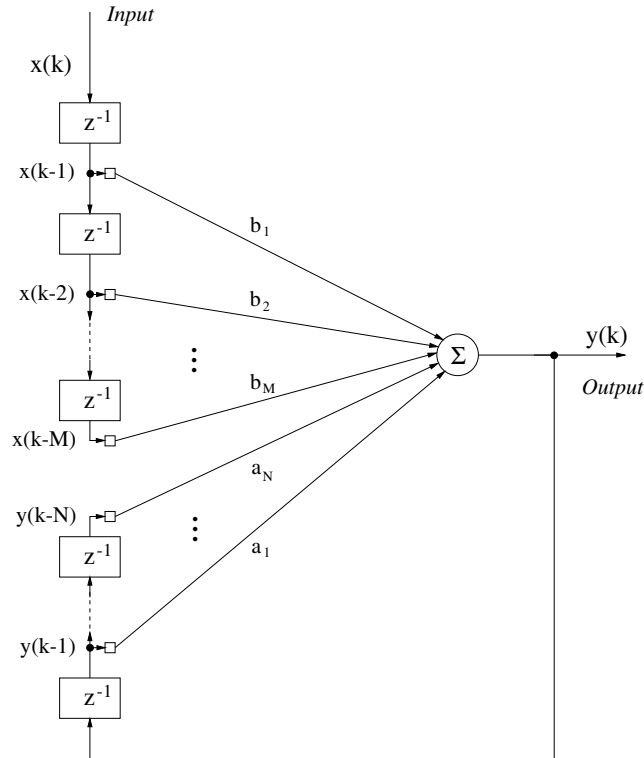


Figure 6.11 Adaptive IIR filter

To overcome such a problem, a training configuration depicted in Figure 6.10 may be considered. This configuration is described by

$$\hat{y}(k) = f(u(k), \dots, u(k - M), \hat{y}(k - 1), \dots, \hat{y}(k - N)). \tag{6.20}$$

Here, the previous estimated outputs $\hat{y}(k)$ are fed back into the network. It should be noticed that these two configurations require different training algorithms. The configuration described by Figure 6.10 and Equation (6.20) is known as the *nonlinear parallel model* (Alippi and Piuri 1996; Qin *et al.* 1992) and requires the use of a recursive training algorithm, such as the RTRL algorithm.

The nonlinear prediction configuration using a recurrent neural network is shown in Figure 5.2, where the signal to be predicted $u(k)$ is delayed through a tap delay line and fed into a neural predictor.

6.7.1 Learning Strategies for a Neural Adaptive Recursive Filter

To introduce learning strategies for recurrent neural networks, we start from corresponding algorithms for IIR adaptive filters. An IIR adaptive filter can be thought of as a recurrent perceptron from Figure 6.2, for which the neuron is linear, i.e. it performs only summation instead of both summation and nonlinear mapping. An IIR

adaptive filter in the prediction configuration is shown in Figure 6.11. A comprehensive account of adaptive IIR filters is given in Regalia (1994).

Two classes of adaptive learning algorithms used for IIR systems are the *equation error* and *output error* algorithms (Shynk 1989). In the equation error configuration, the desired signal $d(k)$ is fed back into the adaptive filter, whereas in the output error configuration, the signals that are fed back are the estimated outputs $\hat{y}(k)$.

6.7.2 Equation Error Formulation

The output $y_{EE}(k)$ of the equation error IIR filter strategy is given by

$$y_{EE}(k) = \sum_{i=1}^N a_i(k)d(k-i) + \sum_{j=1}^M b_j(k)x(k-j), \quad (6.21)$$

where $\{a_i(k)\}$ and $\{b_j(k)\}$ are adjustable coefficients which correspond, respectively, to the feedback and input signals. Since the functional relationship (6.21) does not comprise of delayed outputs $y_{EE}(k)$, this filter *does not* have feedback and the output $y_{EE}(k)$ depends *linearly* on its coefficients. This means that the learning algorithm for this structure is in fact a kind of LMS algorithm for an FIR structure with inputs $\{d(k)\}$ and $\{x(k)\}$. A more compact expression for filter (6.21) is given by

$$y_{EE}(k) = A(k, z^{-1})d(k) + B(k, z^{-1})x(k), \quad (6.22)$$

where

$$A(k, z^{-1}) = \sum_{i=1}^N a_i(k)z^{-i} \quad \text{and} \quad B(k, z^{-1}) = \sum_{j=1}^M b_j(k)z^{-j}.$$

The equation error $e_{EE}(k) = d(k) - y_{EE}(k)$ can be expressed as

$$e_{EE}(k) = d(k) - [1 - A(k, z^{-1})]d(k) - B(k, z^{-1})x(k), \quad (6.23)$$

whereby the name of the method is evident. Since $e_{EE}(k)$ is a linear function of coefficients $\{a_i(k)\}$ and $\{b_j(k)\}$, the error performance surface in this case is quadratic, with a single global minimum. In the presence of noise, however, this minimum is in a different place from in the output error formulation.

6.7.3 Output Error Formulation

The output $y_{OE}(k)$ of the output error learning strategy is given by

$$y_{OE}(k) = \sum_{i=1}^N a_i(k)y_{OE}(k-i) + \sum_{j=1}^M b_j(k)x(k-j). \quad (6.24)$$

A more compact form of Equation (6.24) can be expressed as

$$y_{OE}(k) = \frac{B(k, z^{-1})}{1 - A(k, z^{-1})}x(k). \quad (6.25)$$

The output error $e_{\text{OE}}(k) = d(k) - y_{\text{OE}}(k)$ is the difference between the teaching signal $d(k)$ and output $y_{\text{OE}}(k)$, hence the name of the method. The output $y_{\text{OE}}(k)$ is a function of the coefficients and past outputs, and so too is the error $e_{\text{OE}}(k)$. As a consequence, the error performance surface for this strategy has potentially multiple local minima, especially if the order of the model is smaller than the order of the process.

Notice that the equation error can be expressed as a filtered version of the output error as

$$e_{\text{EE}}(k) = [1 - A(k, z^{-1})]e_{\text{OE}}(k). \quad (6.26)$$

6.8 Filter Coefficient Adaptation for IIR Filters

We first present the coefficient adaptation algorithm for an *output error* IIR filter. In order to derive the relations for filter coefficient adaptation, let us define the gradient $\nabla_{\boldsymbol{\theta}}(E(k))$ for the instantaneous cost function $E(k) = \frac{1}{2}e^2(k)$ as

$$\nabla_{\boldsymbol{\theta}}E(k) = \frac{\partial E(k)}{\partial \boldsymbol{\theta}(k)} = e_{\text{OE}}(k)\nabla_{\boldsymbol{\theta}}e_{\text{OE}}(k) = -e_{\text{OE}}(k)\nabla_{\boldsymbol{\theta}}y_{\text{OE}}(k), \quad (6.27)$$

where $\boldsymbol{\theta}(k) = [b_1(k), \dots, b_M(k), a_1(k), \dots, a_N(k)]^T$. The gradient vector consists of partial derivatives of the output with respect to filter coefficients

$$\nabla_{\boldsymbol{\theta}}y_{\text{OE}}(k) = \left[\frac{\partial y_{\text{OE}}(k)}{\partial b_1(k)}, \dots, \frac{\partial y_{\text{OE}}(k)}{\partial b_M(k)}, \frac{\partial y_{\text{OE}}(k)}{\partial a_1(k)}, \dots, \frac{\partial y_{\text{OE}}(k)}{\partial a_N(k)} \right]^T. \quad (6.28)$$

To derive the coefficient update equations, notice that the inputs $\{x(k)\}$ are independent from the feedback coefficients $a_i(k)$. Now, take the derivatives of both sides of (6.24) first with respect to $a_i(k)$ and then with respect to $b_j(k)$ to obtain

$$\left. \begin{aligned} \frac{\partial y_{\text{OE}}(k)}{\partial a_i(k)} &= y_{\text{OE}}(k-i) + \sum_{m=1}^N a_m(k) \frac{\partial y_{\text{OE}}(k-m)}{\partial a_i(k)}, \\ \frac{\partial y_{\text{OE}}(k)}{\partial b_j(k)} &= x(k-j) + \sum_{m=1}^N a_m(k) \frac{\partial y_{\text{OE}}(k-m)}{\partial b_j(k)}. \end{aligned} \right\} \quad (6.29)$$

There is a difficulty in practical applications of this algorithm, since the partial derivatives in Equation (6.29) are with respect to current values of $a_m(k)$ and $b_m(k)$, which makes Equation (6.29) nonrecursive. Observe that if elements of $\boldsymbol{\theta}$ were independent of $\{y(k-i)\}$, then the gradient calculation would be identical to the FIR case. However, we have delayed samples of $y_{\text{OE}}(k)$ involved in calculation of $\boldsymbol{\theta}(k)$ and an approximation to algorithm (6.29), known as the pseudolinear regression algorithm (PRA), is used. It is reasonable to assume that with a sufficiently small learning rate η , the coefficients will adapt slowly, i.e.

$$\boldsymbol{\theta}(k) \approx \boldsymbol{\theta}(k-1) \approx \dots \approx \boldsymbol{\theta}(k-N). \quad (6.30)$$

The previous approximation is particularly good for N small. From (6.29) and (6.30), we finally have the equations for LMS IIR gradient adaptation,

$$\left. \begin{aligned} \frac{\partial y_{\text{OE}}(k)}{\partial a_i(k)} &\approx y_{\text{OE}}(k-i) + \sum_{m=1}^N a_m(k) \frac{\partial y_{\text{OE}}(k-m)}{\partial a_i(k-m)}, \\ \frac{\partial y_{\text{OE}}(k)}{\partial b_j(k)} &\approx x(k-j) + \sum_{m=1}^N a_m(k) \frac{\partial y_{\text{OE}}(k-m)}{\partial b_j(k-m)}. \end{aligned} \right\} \quad (6.31)$$

The partial derivatives

$$\frac{\partial y_{\text{OE}}(k-m)}{\partial a_i(k-m)} \quad \text{and} \quad \frac{\partial y_{\text{OE}}(k-m)}{\partial b_j(k-m)}$$

admit computation in a recursive fashion. For more details see Treichler (1987), Regalia (1994) and Shynk (1989).

To express this algorithm in a more compact form, let us introduce the weight vector $\mathbf{w}(k)$ as

$$\mathbf{w}(k) = [b_1(k), b_2(k), \dots, b_M(k), a_1(k), a_2(k), \dots, a_N(k)]^T \quad (6.32)$$

and the IIR filter input vector $\mathbf{u}(k)$ as

$$\mathbf{u}(k) = [x(k-1), \dots, x(k-M), y(k-1), \dots, y(k-N)]^T. \quad (6.33)$$

With this notation, we have, for instance, $w_2(k) = b_2(k)$, $w_{M+2}(k) = a_2(k)$, $u_M(k) = x(k-M)$ or $u_{M+1}(k) = y(k-1)$. Now, Equation (6.31) can be rewritten in a compact form as

$$\frac{\partial y_{\text{OE}}(k)}{\partial w_i(k)} \approx u_i(k) + \sum_{m=1}^N w_{m+M}(k) \frac{\partial y_{\text{OE}}(k-m)}{\partial w_i(k-m)}. \quad (6.34)$$

If we denote

$$\pi_i(k) = \frac{\partial y_{\text{OE}}(k)}{\partial w_i(k)}, \quad i = 1, \dots, M+N,$$

then (6.34) becomes

$$\pi_i(k) \approx u_i(k) + \sum_{m=1}^N w_{m+M}(k) \pi_i(k-m). \quad (6.35)$$

Finally, the weight update equation for a linear IIR adaptive filter can be expressed as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)e(k)\boldsymbol{\pi}(k), \quad (6.36)$$

where $\boldsymbol{\pi}(k) = [\pi_1(k), \dots, \pi_{M+N}(k)]^T$.

The adaptive IIR filter in a system identification configuration for the output error formulation is referred to as a model reference adaptive system (MRAS) in the control literature.

6.8.1 Equation Error Coefficient Adaptation

The IIR filter input vector $\mathbf{u}(k)$ for *equation error* adaptation can be expressed as

$$\mathbf{u}(k) = [x(k-1), \dots, x(k-M), d(k-1), \dots, d(k-N)]^T. \quad (6.37)$$

Both the external input vector $\mathbf{x}(k) = [x(k-1), \dots, x(k-M)]^T$ and the vector of teaching signals $\mathbf{d}(k) = [d(k-1), \dots, d(k-N)]^T$ are not generated through the filter and are independent of the filter weights. Therefore, the weight adaptation for an *equation error* IIR adaptive filter can be expressed as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)e(k)\mathbf{u}(k), \quad (6.38)$$

which is identical to the formula for adaptation of FIR adaptive filters. In fact, an *equation error* IIR adaptive filter can be thought of as a dual input FIR adaptive filter.

6.9 Weight Adaptation for Recurrent Neural Networks

The output of a recurrent perceptron, shown in Figure 6.2, with weight vectors $\mathbf{w}_a(k) = [w_{M+2}, \dots, w_{M+N+1}]^T$ and $\mathbf{w}_b(k) = [w_1, w_2, \dots, w_M, w_{M+1}]^T$, which comprise weights associated with delayed outputs and inputs, respectively, is given by

$$\left. \begin{aligned} y(k) &= \Phi(\text{net}(k)), \\ \text{net}(k) &= \sum_{j=1}^M w_j(k)x(k-j) + w_{M+1}(k) + \sum_{m=1}^N w_{m+M+1}(k)y(k-m), \end{aligned} \right\} \quad (6.39)$$

where $w_i(k) \in \mathbf{w}(k) = [\mathbf{w}_b^T(k), \mathbf{w}_a^T(k)]^T$, $i = 1, \dots, M + N + 1$. The instantaneous output error in this case is given by

$$e(k) = d(k) - y(k), \quad (6.40)$$

where $d(k)$ denotes the desired (teaching) signal, whereas the cost function is $E = \frac{1}{2}e^2(k)$. In order to obtain the weight vector $\mathbf{w}(k+1)$, we have to calculate the gradient $\nabla_{\mathbf{w}}E(k)$ and the weight update vector $\Delta\mathbf{w}(k)$ for which the elements $\Delta w_i(k)$, $i = 1, \dots, M + N + 1$, are

$$\Delta w_i(k) = -\eta \frac{\partial E(k)}{\partial w_i(k)} = -\eta e(k) \frac{\partial e(k)}{\partial w_i(k)} = +\eta e(k) \frac{\partial y(k)}{\partial w_i(k)}. \quad (6.41)$$

From (6.39), we have

$$\frac{\partial y(k)}{\partial w_i(k)} = \Phi'(\text{net}(k)) \frac{\partial \text{net}(k)}{\partial w_i(k)}. \quad (6.42)$$

Following the analysis provided for IIR adaptive filters (6.34), we see that the partial derivatives of outputs with respect to weights form a recursion. Thus, we have

$$\frac{\partial y(k)}{\partial w_i(k)} \approx \Phi'(\text{net}(k)) \left[u_i(k) + \sum_{m=1}^N w_{m+M+1}(k) \frac{\partial y(k-m)}{\partial w_i(k-m)} \right], \quad (6.43)$$

where vector $\mathbf{u}(k) = [x(k), \dots, x(k-M), 1, y(k), \dots, y(k-N)]^T$ comprises the set of all input signals to a recurrent perceptron, including the delayed inputs, delayed outputs and bias, and $i = 1, \dots, M+N+1$. If we introduce notation

$$\pi_i(k) = \frac{\partial y(k)}{\partial w_i(k)},$$

then (6.43) can be rewritten as

$$\pi_i(k) = \Phi'(\text{net}(k)) \left[u_i(k) + \sum_{m=1}^N w_{m+M+1}(k) \pi_i(k-m) \right]. \quad (6.44)$$

In control theory, coefficients $\pi_i(k)$ are called *sensitivities*. It is convenient to assume zero initial conditions for sensitivities (6.44) (Haykin 1994), i.e.

$$\pi_i(0) = 0, \quad i = 1, \dots, M+N+1.$$

The analysis presented so far is the basis of the real-time recurrent learning (RTRL) algorithm. The derivation of this online direct-gradient algorithm for a general recurrent neural network is more involved and is given in Appendix D.

Finally, the weight update equation for a nonlinear adaptive filter in the form of a recurrent perceptron can be expressed as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)e(k)\boldsymbol{\pi}(k), \quad (6.45)$$

where $\boldsymbol{\pi}(k) = [\pi_1(k), \dots, \pi_{M+N+1}(k)]^T$. In order to calculate vector $\boldsymbol{\pi}(k)$, we have to store the following matrix:

$$\mathbf{H}(k) = \begin{bmatrix} \pi_1(k-1) & \pi_2(k-1) & \cdots & \pi_{M+N+1}(k-1) \\ \pi_1(k-2) & \pi_2(k-2) & \cdots & \pi_{M+N+1}(k-2) \\ \vdots & \vdots & \ddots & \vdots \\ \pi_1(k-N) & \pi_2(k-N) & \cdots & \pi_{M+N+1}(k-N) \end{bmatrix}. \quad (6.46)$$

The learning procedure described above is the so-called *supervised learning* (or *output error learning*) algorithm for a recurrent perceptron.

6.9.1 Teacher Forcing Learning for a Recurrent Perceptron

The input vector $\mathbf{u}(k)$ for *teacher forced* adaptation of a recurrent perceptron can be expressed as

$$\mathbf{u}(k) = [x(k-1), \dots, x(k-M), 1, d(k-1), \dots, d(k-N)]^T. \quad (6.47)$$

The analysis of this algorithm is analogous to that presented in Section 6.8.1. Hence, the weight adaptation for a *teacher forced* recurrent perceptron can be expressed as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)e(k)\Phi'(\text{net}(k))\mathbf{u}(k), \quad (6.48)$$

which is identical to the formula for adaptation of dual-input nonlinear FIR adaptive filters.

6.9.2 Training Process for a NARMA Neural Predictor

Algorithms for training of recurrent neural networks have been extensively studied since the late 1980s. The real-time recurrent learning (RTRL) algorithm (Robinson and Fallside 1987; Williams and Zipser 1989a) enabled training of simple RNNs, whereas Pineda provided recurrent backpropagation (RBP) (Pineda 1987, 1989). RTRL-based training of the RNN employed as a nonlinear adaptive filter is based upon minimising the instantaneous squared error at the output of the first neuron of the RNN (Williams and Zipser 1989a), which can be expressed as $\min(e^2(k)) = \min([s(k) - y_1(k)]^2)$, where $e(k)$ denotes the error at the output of the RNN and $s(k)$ is the teaching signal. It is an *output error* algorithm. The correction $\Delta \mathbf{W}(k)$ to the weight matrix $\mathbf{W}(k)$ of the RNN for prediction is calculated as

$$\Delta \mathbf{W}(k) = -\eta \frac{\partial E(k)}{\partial \mathbf{W}(k)} = \eta e(k) \frac{\partial y_1(k)}{\partial \mathbf{W}(k)}, \quad (6.49)$$

which turns out to be based upon a recursive calculation of the gradients of the outputs of the neurons (Mandic *et al.* 1998; Williams and Zipser 1989a). A detailed gradient descent training process (RTRL) for RNNs is given in Appendix D.

Similarly to the analysis for IIR filters and recurrent perceptrons, in order to make the algorithm run in real time, an approximation has to be made, namely that for a small learning rate η , the following approximation,

$$\frac{\partial y_i(k-m)}{\partial \mathbf{W}(k)} \approx \frac{\partial y_i(k-m)}{\partial \mathbf{W}(k-m)}, \quad i, m = 1, \dots, N, \quad (6.50)$$

holds for slowly time-varying input statistics.

Another frequently used algorithm for training recurrent neural networks is a variant of the extended Kalman filter algorithm called the linearised recursive least-squares (LRLS) algorithm (Baltersee and Chambers 1998; Mandic *et al.* 1998). Its derivation is rather mathematically involved and is given in Appendix D. This algorithm is related to the previously mentioned gradient-based algorithms, and it modifies both the weights and the states of the network on an equal basis.

6.10 The Problem of Vanishing Gradients in Training of Recurrent Neural Networks

Recently, several empirical studies have shown that when using gradient-descent learning algorithms, it might be difficult to learn simple temporal behaviour with long time dependencies (Bengio *et al.* 1994; Mozer 1993), i.e. those problems for which the output of a system at time instant k depends on network inputs presented at times $\tau \ll k$. Bengio *et al.* (1994) analysed learning algorithms for systems with long time dependencies and showed that for gradient-based training algorithms, the information about the gradient contribution K steps in the past vanishes for large K . This effect is referred to as the problem of *vanishing gradient*, which partially explains why gradient descent algorithms are not very suitable to estimate systems and signals with long time dependencies. For instance, common recurrent neural networks encounter

problems when learning information with long time dependencies, which is a problem in prediction of nonlinear and nonstationary signals.

The forgetting behaviour experienced in neural networks is formalised in Definition 6.10.1 (Frasconi *et al.* 1992).

Definition 6.10.1 (forgetting behaviour). A recurrent network exhibits forgetting behaviour if

$$\lim_{K \rightarrow \infty} \frac{\partial z_i(k)}{\partial z_j(k-K)} = 0 \quad \forall k \in \mathcal{K}, i \in \mathcal{O}, j \in \mathcal{I}, \quad (6.51)$$

where z are state variables, \mathcal{I} denotes the set of input neurons, \mathcal{O} denotes the set of output neurons and \mathcal{K} denotes the time index set.

A state space representation of recurrent NARX neural networks can be expressed as

$$z_i(k+1) = \begin{cases} \Phi(\mathbf{u}(k), \mathbf{z}(k)), & i = 1, \\ z_i(k), & i = 2, \dots, N, \end{cases} \quad (6.52)$$

where the output $y(k) = z_1(k)$ and $z_i, i = 1, 2, \dots, N$, are state variables of a recurrent neural network. To represent mathematically the problem of vanishing gradients, recall that the weight update for gradient-based methods for a neural network with one output neuron can be expressed as

$$\Delta \mathbf{w}(k) = \eta e(k) \left(\frac{\partial y(k)}{\partial \mathbf{w}(k)} \right) = \eta e(k) \sum_i \left(\frac{\partial y(k)}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial \mathbf{w}(k)} \right). \quad (6.53)$$

Expanding Equation (6.53) and using the chain rule, we have

$$\Delta \mathbf{w}(k) = \eta e(k) \sum_i \left(\frac{\partial y_i(k)}{\partial z_i(k)} \sum_{l=1}^k \frac{\partial z_i(k)}{\partial z_i(k-l)} \frac{\partial z_i(k-l)}{\partial \mathbf{w}(k-l)} \right). \quad (6.54)$$

Partial derivatives of the state space variables $\partial z_i(k)/\partial z_i(l)$ from (6.54) build a Jacobian matrix $\mathbf{J}(k, k-l)$,⁶ which is given by

$$\mathbf{J}(k) = \begin{bmatrix} \frac{\partial y(k)}{\partial z_1(k)} & \frac{\partial y(k)}{\partial z_2(k)} & \dots & \frac{\partial y(k)}{\partial z_N(k)} \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}. \quad (6.55)$$

If all the eigenvalues of Jacobian (6.55) are inside the unit circle, then the corresponding transition matrix of $\mathbf{J}(k)$ is an exponentially decreasing function of k . As a consequence, the states of the network will remain within a set defined by a hyperbolic attractor, and the adaptive system will not be able to escape from this fixed point.

⁶ Notice that the Jacobian (6.55) represents a companion matrix. The stability of these matrices is analysed in Mandic and Chambers (2000d) and will be addressed in Chapter 7.

Hence, a small perturbation in the weight vector \mathbf{w} affects mostly the near past. This means that even if there was a weight update $\Delta\mathbf{w}(k)$ that would move the current point in the state space of the network from a present attractor, the gradient $\nabla_{\mathbf{w}}E(k)$ would not carry this information, due to the effects of *vanishing gradient*. Due to this effect, the network for which the dynamics are described above, is not able to estimate/represent long term dependencies in the input signal/system (Haykin 1999b).

Several approaches have been suggested to circumvent the problem of vanishing gradient in training RNNs. Most of them rest upon embedding memory in neural networks, whereas several propose improved learning algorithms, such as the extended Kalman filter algorithm (Mandic *et al.* 1998), Newton type algorithms, annealing algorithms (Mandic and Chambers 1999c; Rose 1998) and *a posteriori* algorithms (Mandic and Chambers 1998c). The deterministic annealing approach, for instance, offers (Rose 1998) (i) the ability to avoid local minima, (ii) applicability to many different structures/architectures, (iii) the ability to minimise the cost function even when its gradients tend to vanish for time indices from the distant past.

Embedded memory is particularly significant in recurrent NARX and NARMAX neural networks (Lin *et al.* 1997). This embedded memory can help to speed up propagation of gradient information, and hence help to reduce the effect of vanishing gradient (Lin *et al.* 1996). There are various methods to introduce memory and temporal information into neural networks. These include (Kim 1998) (i) creating a spatial representation of temporal pattern, (ii) putting time delays into the neurons or their connections, (iii) employing recurrent connections, (iv) using neurons with activations that sum inputs over time, (v) using some combination of these. The PRNN, for instance, uses the combination of (i) and (iii).

6.11 Learning Strategies in Different Engineering Communities

Learning strategies classification for a broad class of adaptive systems is given in Nerrand *et al.* (1994), where learning algorithms are classified as directed, semidirected and undirected. *Directed* algorithms are suitable mostly for the modelling of noiseless dynamical systems, or for systems with noise added to the state variables of the black-box model of the system, whereas *undirected* algorithms are suitable for predicting the output of systems for which the output is corrupted by additive white noise. For instance, the RTRL algorithm is a undirected algorithm, whereas *a posteriori* (data-reusing) algorithms are undirected-directed.

There is a need to relate terms coming from different communities, which refer to the same learning strategies. This correspondence for terms used in the signal processing, system identification, neural networks and adaptive systems communities is shown in Table 6.1.

6.12 Learning Algorithms and the Bias/Variance Dilemma

The optimal prediction performance would provide a compromise between the bias and the variance of the prediction error achieved by a chosen model. An analogy with

Table 6.1 Terms related to learning strategies used in different communities

Signal Processing	System ID	Neural Networks	Adaptive Systems
Output Error	Parallel	Supervised	Unidirected
Equation Error	Series-Parallel	Teacher Forcing	Directed

polynomials, which to a certain extent applies for trajectory-tracking problems, shows that if a number of points is approximated by a polynomial of an insufficient order, the polynomial fit has errors, i.e. the polynomial curve cannot pass through every point. On the other hand, if the order of the polynomial is greater than the number of points through which to fit, the polynomial will pass exactly through every point, but will oscillate in between, hence having a large variance.

If we consider a measure of the error as

$$E[(d(k) - y(k))^2], \quad (6.56)$$

then adding and subtracting a dummy variable $E[y(k)]$ within the square brackets of (6.56) yields (Principe *et al.* 2000)

$$E[(d(k) - y(k))^2] = E[E(y(k)) - d(k)]^2 + E[(y(k) - E[y(k)])^2]. \quad (6.57)$$

The first term on the right-hand side of (6.57) represents the squared bias term, whereas the second term on the right-hand side of (6.57) represents the variance. An inspection of (6.57) shows that teacher forcing (equation error) algorithms suffer from biased estimates, since $d(k) \neq E[y(k)]$ in the first term of (6.57), whereas the supervised (output error) algorithms might suffer from increased variance.

If the noise term $\epsilon(k)$ is included within the output, for the case when we desire to approximate mapping $f(x(k))$ by a neural network for which the output is $y(k)$, we want to minimise

$$E[y(k) - f(x)]. \quad (6.58)$$

If

$$\left. \begin{aligned} y(k) &= f^*(k) + \epsilon(k), & f^*(k) &= E[y|x], \\ E[\epsilon|x] &= 0, & \bar{f} &= E[f(x)], \end{aligned} \right\} \quad (6.59)$$

where, for convenience, the time index k is dropped, we have (Gemon 1992)

$$E[f(x)] = E[\epsilon^2] + E[f^* - \bar{f}]^2 + E[(f - \bar{f})^2] = \sigma^2 + B^2 + \text{var}(f). \quad (6.60)$$

In Equation (6.60), σ^2 denotes the noise variance, $B^2 = E[f^* - \bar{f}]^2$ is the squared bias and $\text{var}(f) = E[(f - \bar{f})^2]$ denotes the variance. The term σ^2 cannot be reduced, since it is due to the observation noise. The second and third term in (6.60) can be reduced by choosing an appropriate architecture and learning strategy, as shown before in this chapter. A thorough analysis of the bias/variance dilemma can be found in Gemon (1992) and Haykin (1994).

6.13 Recursive and Iterative Gradient Estimation Techniques

Since, for real-time applications, the coefficient update in the above algorithms is finished before the next input sample arrives, there is a possibility to reiterate learning algorithms about the current point in the state space of the network. Therefore, a relationship between the iteration index l and time index k can be established. The possibilities include the following.

- A purely recursive algorithm – one gradient iteration (coefficients update) per sampling interval k , i.e. $l = 1$. This is the most commonly used technique for recursive filtering algorithms.
- Several coefficients updates per sampling interval, i.e. $l > 1$ – this may improve the nonlinear filter performance, as for instance in *a posteriori* algorithms, where $l \geq 2$ for every time instant (Mandic and Chambers 2000e).
- P coefficients with greatest magnitude updated every sampling period – this helps to reduce computational complexity (Douglas 1997).
- Coefficients are only updated every K sampling periods – useful for processing signals with slowly varying statistics.

6.14 Exploiting Redundancy in Neural Network Design

Redundancy encountered in neural network design can improve the robustness and fault tolerance in the neural network approach to a specific signal processing task, as well as serve as a basis for network topology adaptation due to statistical changes in the input data. It gives more degrees of freedom than necessary for a particular task. In the case of neural networks for time series prediction, the possible sources of redundancy can be

- Redundancy in the global network architecture, which includes design with
 - more layers of neurons than necessary, which enables network topology adaptation while in use and improves robustness in the network,
 - more neurons within the layers than necessary, which improves robustness and fault-tolerance in the network,
 - memory neurons (Poddar and Unnikrishnan 1991), which are specialised neurons added to the network to store the past activity of the network. Memory neurons are time replicaes of processing neurons, in all the network layers except the output layer.
- Redundancy among the internal connections within the network. According to the level of redundancy in the network we can define
 - fully recurrent networks (Connor 1994; McDonnell and Waagen 1994),
 - partially recurrent networks (Bengio 1995; Haykin and Li 1995),

- time delays in the network as a source of redundancy (Baldi and Atiya 1994; Haykin and Li 1995; Nerrand *et al.* 1994).
- Data reusing, where we expect to make the trajectory along the error performance surface ‘less stochastic’.⁷

6.15 Summary

It has been shown that a small neural network can represent high-order nonlinear systems, whereas a very large number of terms are required for an equivalent Volterra series representation. We have shown that when modelling an unknown dynamical system, or tracking unknown dynamics, it is important to concentrate on the embedding dimension of the network.

Architectures for neural networks as nonlinear adaptive filters have been introduced and learning strategies, such as *equation error* and *output error* strategy have been explained. Connection between the learning strategies from different engineering communities has been established. The online real-time recurrent learning algorithm for general recurrent neural networks has been derived inductively, starting from a linear adaptive IIR filter, via a recurrent perceptron, through to a general case. Finally, sources of redundancies in RNN architectures have been addressed.

⁷ We wish to speed up the convergence of learning trajectory along the error performance surface. It seems that reusing of ‘good’ data can improve the convergence rate of a learning algorithm. This data reusing makes a learning algorithm *iterative* as well as recursive.

7

Stability Issues in RNN Architectures

7.1 Perspective

The focus of this chapter is on stability and convergence of relaxation realised through NARMA recurrent neural networks. Unlike other commonly used approaches, which mostly exploit Lyapunov stability theory, the main mathematical tool employed in this analysis is the contraction mapping theorem (CMT), together with the fixed point iteration (FPI) technique. This enables derivation of the asymptotic stability (AS) and global asymptotic stability (GAS) criteria for neural relaxive systems. For rigour, existence, uniqueness, convergence and convergence rate are considered and the analysis is provided for a range of activation functions and recurrent neural networks architectures.

7.2 Introduction

Stability and convergence are key issues in the analysis of dynamical adaptive systems, since the analysis of the dynamics of an adaptive system can boil down to the discovery of an attractor (a stable equilibrium) or some other kind of fixed point. In neural associative memories, for instance, the locally stable equilibrium states (attractors) store information and form neural memory. Neural dynamics in that case can be considered from two aspects, convergence of state variables (memory recall) and the number, position, local stability and domains of attraction of equilibrium states (memory capacity). Conveniently, LaSalle's invariance principle (LaSalle 1986) is used to analyse the state convergence, whereas stability of equilibria are analysed using some sort of linearisation (Jin and Gupta 1996). In addition, the dynamics and convergence of learning algorithms for most types of neural networks may be explained and analysed using fixed point theory.

Let us first briefly introduce some basic definitions. The full definitions and further details are given in Appendix I. Consider the following linear, finite dimensional,

autonomous system¹ of order N

$$y(k) = \sum_{i=1}^N a_i(k)y(k-i) = \mathbf{a}^T(k)\mathbf{y}(k-1). \quad (7.1)$$

Definition 7.2.1 (see Kailath (1980) and LaSalle (1986)). The system (7.1) is said to be *asymptotically stable* in $\Omega \subseteq \mathbb{R}^N$, if for any $\mathbf{y}(0)$, $\lim_{k \rightarrow \infty} y(k) = 0$, for $\mathbf{a}(k) \in \Omega$.

Definition 7.2.2 (see Kailath (1980) and LaSalle (1986)). The system (7.1) is *globally asymptotically stable* if for any initial condition and any sequence $\mathbf{a}(k)$, the response $y(k)$ tends to zero asymptotically.

For NARMA systems realised via neural networks, we have

$$y(k+1) = \Phi(\mathbf{y}(k), \mathbf{w}(k)). \quad (7.2)$$

Let $\Phi(k, k_0, \mathbf{Y}_0)$ denote the trajectory of the state change for all $k \geq k_0$, with $\Phi(k_0, k_0, \mathbf{Y}_0) = \mathbf{Y}_0$. If $\Phi(k, k_0, \mathbf{Y}^*) = \mathbf{Y}^*$ for all $k \geq 0$, then \mathbf{Y}^* is called an *equilibrium* point. The largest set $\mathcal{D}(\mathbf{Y}^*)$ for which this is true is called the *domain of attraction* of the equilibrium \mathbf{Y}^* . If $\mathcal{D}(\mathbf{Y}^*) = \mathbb{R}^N$ and if \mathbf{Y}^* is asymptotically stable, then \mathbf{Y}^* is said to be asymptotically stable in large or *globally asymptotically stable*.

It is important to clarify the difference between asymptotic stability and absolute stability. Asymptotic stability may depend upon the input (initial conditions), whereas global asymptotic stability does not depend upon initial conditions. Therefore, for an absolutely stable neural network, the system state will converge to one of the asymptotically stable equilibrium states regardless of the initial state and the input signal. The equilibrium points include the isolated minima as well as the maxima and saddle points. The maxima and saddle points are not stable equilibrium points. Robust stability for the above discussed systems is still under investigation (Bauer *et al.* 1993; Jury 1978; Mandic and Chambers 2000c; Premaratne and Mansour 1995).

In conventional nonlinear systems, the system is said to be globally asymptotically stable, or asymptotically stable in large, if it has a unique equilibrium point which is globally asymptotically stable in the sense of Lyapunov. In this case, for an arbitrary initial state $\mathbf{x}(0) \in \mathbb{R}^N$, the state trajectory $\phi(k, \mathbf{x}(0), \mathbf{s})$ will converge to the unique equilibrium point \mathbf{x}^* , satisfying

$$\mathbf{x}^* = \lim_{k \rightarrow \infty} \phi[k, \mathbf{x}(0), \mathbf{s}]. \quad (7.3)$$

Stability in this context has been considered in terms of Lyapunov stability and M -matrices (Forti and Tesi 1994; Liang and Yamaguchi 1997). To apply the Lyapunov method to a dynamical system, a neural system has to be mapped onto a new system for which the origin is at an equilibrium point. If the network is stable, its ‘energy’ will decrease to a minimum as the system approaches and attains its equilibrium state. If a function that maps the objective function onto an ‘energy function’ can be found, then the network is guaranteed to converge to its equilibrium state (Hopfield and

¹ Stability of systems of this type is discussed in Appendix H.

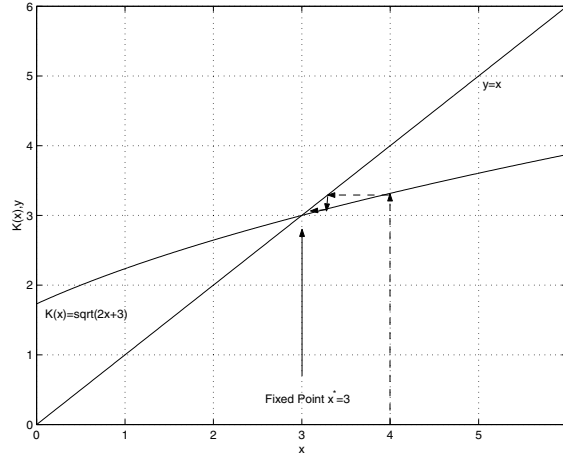


Figure 7.1 FPI solution for roots of $F(x) = x^2 - 2x - 3$

Tank 1985; Luh *et al.* 1998). The Lyapunov stability of neural networks is studied in detail in Han *et al.* (1989) and Jin and Gupta (1996).

The concept of fixed point will be central to much of what follows, for which the basic theorems and principles are introduced in Appendix G.

Point x^* is called a *fixed point* of a function K if it satisfies $K(x^*) = x^*$, i.e. the value x^* is unchanged under the application of function K . For instance, the roots of function $F(x) = x^2 - 2x - 3$ can be found by rearranging $x_{k+1} = K(x_k) = \sqrt{2x_k + 3}$ via fixed point iteration. The roots of the above function are -1 and 3 . The FPI which started from $x_0 = 4$ converges to within 10^{-5} of the exact solution in nine steps, which is depicted in Figure 7.1. This example is explained in more detail in Appendix G.

One of the virtues of neural networks is their processing power, which rests upon their ability to converge to a set of fixed points in the state space. Stability analysis, therefore, is essential for the derivation of conditions that assure convergence to these fixed points. Stability, although necessary, is not sufficient for effective processing (see Appendix H), since in practical applications, it is desirable that a neural system converges to only a preselected set of fixed points. In the remainder of this chapter, two different aspects of equilibrium, i.e. the *static* aspect (existence and uniqueness of equilibrium states) and the *dynamic* aspect (global stability, rate of convergence), are studied. While analysing global asymptotic stability,² it is convenient to study the static problem of the existence and uniqueness of the equilibrium point first, which is the necessary condition for GAS.

² It is important to note that the iterates of random Lipschitz functions converge if the functions are contracting on the average (Diaconis and Freedman 1999). The theory of random operators is a probabilistic generalisation of operator theory. The study of probabilistic operator theory and its applications was initiated by the Prague school under the direction of Antonin Spacek, in the 1950s (Bharucha-Reid 1976). They recognised that it is necessary to take into consideration the fact that the operators used to describe the behaviour of systems may not be known exactly. The application of this theory in signal processing is still under consideration and can be used to analyse stochastic learning algorithms (Chambers *et al.* 2000).

7.3 Overview

The role of the nonlinear activation function in the global asymptotic convergence of recurrent neural networks is studied. For a fixed input and weights, a repeated application of the nonlinear difference equation which defines the output of a recurrent neural network is proven to be a relaxation, provided the activation function satisfies the conditions required for a contraction mapping. This relaxation is shown to exhibit linear asymptotic convergence. Nesting of modular recurrent neural networks is demonstrated to be a fixed point iteration in a spatial form.

7.4 A Fixed Point Interpretation of Convergence in Networks with a Sigmoid Nonlinearity

To solve many problems in the field of optimisation, neural control and signal processing, dynamic neural networks need to be designed to have only a unique equilibrium point. The equilibrium point ought to be globally stable to avoid the risk of spurious responses or the problem of local minima. Global asymptotic stability (GAS) has been analysed in the theory of both linear and nonlinear systems (Barnett and Storey 1970; Golub and Van Loan 1996; Haykin 1996a; Kailath 1980; LaSalle 1986; Priestley 1991). For nonlinear systems, it is expected that convergence in the GAS sense depends not only on the values of the parameter vector, but also on the parameters of the nonlinear function involved. As systems based upon sigmoid functions exhibit stability in the bounded input bounded output (BIBO) sense, due to the saturation type sigmoid nonlinearity, we investigate the characteristics of the nonlinear activation function to obtain GAS for a general RNN-based nonlinear system. In that case, both the external input vector to the system $\mathbf{x}(k)$ and the parameter vector $\mathbf{w}(k)$ are assumed to be a time-invariant part of the system under fixed point iteration.

7.4.1 Some Properties of the Logistic Function

To derive the conditions which the nonlinear activation function of a neuron should satisfy to enable convergence of real-time learning algorithms, activation functions of a neuron are analysed in the framework of contraction mappings and fixed point iteration.

Observation 7.4.1. *The logistic function*

$$\Phi(x) = \frac{1}{1 + e^{-\beta x}} \quad (7.4)$$

is a contraction on $[a, b] \in \mathbb{R}$ for $0 < \beta < 4$ and the iteration

$$x_{i+1} = \Phi(x_i) \quad (7.5)$$

converges to a unique solution x^* from $\forall x_0 \in [a, b] \in \mathbb{R}$.

Proof. By the contraction mapping theorem (CMT) (Appendix G), function K is a contraction on $[a, b] \in \mathbb{R}$ if



Figure 7.2 The contraction mapping

- (i) $x \in [a, b] \Rightarrow K(x) \in [a, b]$,
- (ii) $\exists \gamma < 1 \in \mathbb{R}^+$ s.t. $|K(x) - K(y)| \leq \gamma|x - y| \quad \forall x, y \in [a, b]$.

The condition (i) is illustrated in Figure 7.2. The logistic function (7.4) is strictly monotonically increasing, since its first derivative is strictly greater than zero. Hence, in order to prove that Φ is a contraction on $[a, b] \in \mathbb{R}$, it is sufficient to prove that it contracts the upper and lower bound of interval $[a, b]$, i.e. a and b , which in turn gives

- $a - \Phi(a) \leq 0$,
- $b - \Phi(b) \geq 0$.

These conditions will be satisfied if the function Φ is smaller in magnitude than the curve $y = x$, i.e. if

$$|x| > \left| \frac{1}{1 + e^{-\beta x}} \right|, \quad \beta > 0. \tag{7.6}$$

Condition (ii) can be proven using the mean value theorem (MVT) (Luenberger 1969). Namely, as the logistic function Φ (7.4) is differentiable, for $\forall x, y \in [a, b]$, $\exists \xi \in (a, b)$ such that

$$|\Phi(x) - \Phi(y)| = |\Phi'(\xi)(x - y)| = |\Phi'(\xi)||x - y|. \tag{7.7}$$

The first derivative of the logistic function (7.4) is

$$\Phi'(x) = \left(\frac{1}{1 + e^{-\beta x}} \right)' = \frac{\beta e^{-\beta x}}{(1 + e^{-\beta x})^2}, \tag{7.8}$$

which is strictly positive, and for which the maximum value is $\Phi'(0) = \beta/4$. Hence, for $\beta \leq 4$, the first derivative $\Phi' \leq 1$. Finally, for $\gamma < 1 \Leftrightarrow \beta < 4$, function Φ given in (7.4) is a contraction on $[a, b] \in \mathbb{R}$.

Convergence of FPI: if x^* is a zero of $x - \Phi(x) = 0$, or in other words the fixed point of function Φ , then for $\gamma < 1$ ($\beta < 4$)

$$|x_i - x^*| = |\Phi(x_{i-1}) - \Phi(x^*)| \leq \gamma|x_{i-1} - x^*|. \tag{7.9}$$

Thus, since for $\gamma < 1 \Rightarrow \{\gamma\}^i \xrightarrow{i} 0$

$$|x_i - x^*| \leq \gamma^i|x_0 - x^*| \Rightarrow \lim_{i \rightarrow \infty} x_i = x^* \tag{7.10}$$

and iteration $x_{i+1} = \Phi(x_i)$ converges to some $x^* \in [a, b]$. □

Convergence/divergence of the FPI clearly depends on the size of slope β in Φ . Considering the general nonlinear system Equation (7.2), this means that for a fixed input vector to the iterative process and fixed weights of the network, an FPI solution depends on the slope (first derivative) of the nonlinear activation function and some measure of the weight vector. If the solution exists, that is the only value to which

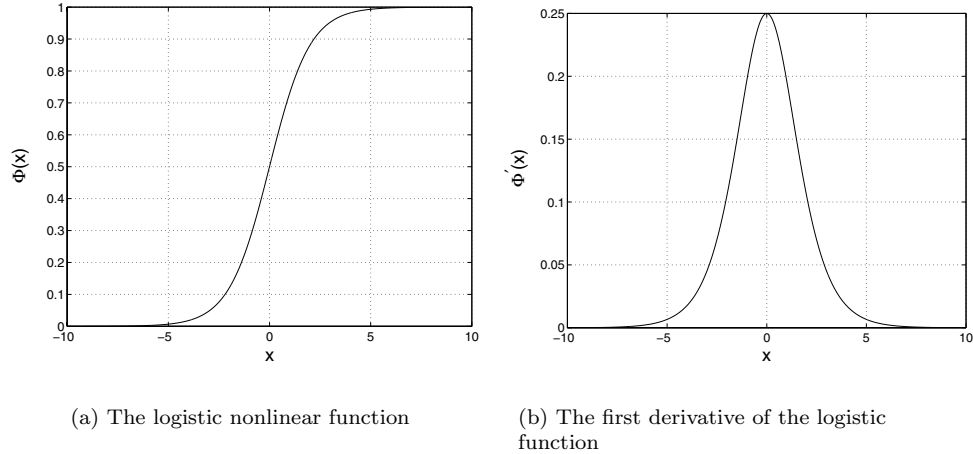


Figure 7.3 The logistic function and its derivative

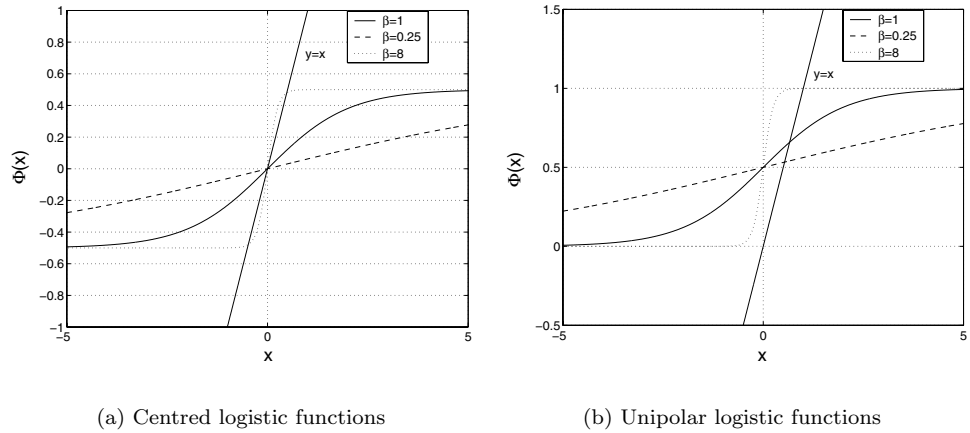


Figure 7.4 Various logistic functions

such a relaxation algorithm converges. Figure 7.3 shows the logistic function and its first derivative for $\beta = 1$. To depict Observation 7.4.1 further, we use a centred logistic function ($\Phi - \text{mean}(\Phi)$), as shown in Figure 7.4(a). For Φ a contraction, the condition (i) from CMT (Appendix G) must be satisfied. That is the case if the values of Φ are smaller in magnitude than the corresponding values of the function $y = x$. As shown in Figure 7.4(a), that condition is satisfied for a range of logistic functions with the slope $0 < \beta < 4$. Indeed, e.g. for $\beta = 8$, the logistic function has an intersection with the function $y = x$ (dotted curve in Figure 7.4(a)), which means that for $\beta > 4$, there are regions in Φ where $(a - \Phi(a)) \not\leq 0$, which violates condition (i) of CMT and Observation 7.4.1.

7.4.2 Logistic Function, Rate of Convergence and Fixed Point Theory

The rate of convergence of a fixed point iteration can be judged by the closeness of x_{k+1} to x^* relative to the closeness of x_k to x^* (Dennis and Schnabel 1983; Gill *et al.* 1981).

Definition 7.4.2. A sequence $\{x_k\}$ is said to converge towards its fixed point x^* with order r if

$$0 \leq \lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r} < \infty, \quad (7.11)$$

where $r \in \mathbb{N}$ is the largest number such that the above inequality holds.

Since we are interested in the value of r that occurs in the limit, r is sometimes called the *asymptotic convergence rate*. If $r = 1$, the sequence is said to exhibit *linear convergence*, if $r = 2$, the sequence is said to exhibit *quadratic convergence*.

Definition 7.4.3. For a sequence $\{x_k\}$ which has an order of convergence r , the *asymptotic error constant* of the fixed point iteration is the value $\gamma \in \mathbb{R}^+$ which satisfies

$$\gamma = \lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r}. \quad (7.12)$$

When $r = 1$, i.e. for linear convergence, γ must be strictly less than unity in order for convergence to occur (Gill *et al.* 1981).

Example 7.4.4. Show that the convergent FPI process

$$x_{i+1} = \Phi(x_i) \quad (7.13)$$

exhibits a *linear asymptotic convergence* for which the error constant equals $|\Phi'(x^*)|$.

Solution. Consider the ratio $|e_{i+1}|/|e_i|$ of successive errors, where $e_i = x_i - x^*$

$$\frac{|e_{i+1}|}{|e_i|} = \frac{|x_{i+1} - x^*|}{|x_i - x^*|} = \frac{|\Phi(x_i) - \Phi(x^*)|}{|x_i - x^*|} \stackrel{\text{MVT}}{=} |\Phi'(\xi)| \quad (7.14)$$

for some $\xi \in (x_i, x^*)$. Having in mind that the iteration (7.13) converges to x^* when $i \rightarrow \infty$

$$\lim_{i \rightarrow \infty} \frac{|e_{i+1}|}{|e_i|} = \lim_{i \rightarrow \infty} |\Phi'(\xi)| = |\Phi'(x^*)|. \quad (7.15)$$

Therefore, iteration (7.13) exhibits linear asymptotic convergence with convergence rate $|\Phi'(x^*)|$. \square

Example 7.4.5. Derive the error bound $e_i = |x_i - x^*|$ for the FPI process

$$x_{i+1} = \Phi(x_i). \quad (7.16)$$

Solution. Rewrite the error bound as

$$x_i - x^* = \Phi(x_{i-1}) - \Phi(x_i) + \Phi(x_i) - \Phi(x^*) \quad (7.17)$$

and therefore

$$|x_i - x^*| \leq \gamma |x_{i-1} - x_i| + \gamma |x_i - x^*|. \quad (7.18)$$

Table 7.1 Fixed point iterates for the logistic function

Starting value x_0	-10	10
First iterate	0.000 045	1
Second iterate	0.5	0.7311
Third iterate	0.6225	0.6750
Fourth iterate	0.6508	0.6626
Fifth iterate	0.6572	0.6598
Sixth iterate	0.6586	0.6592
Seventh iterate	0.6589	0.6591

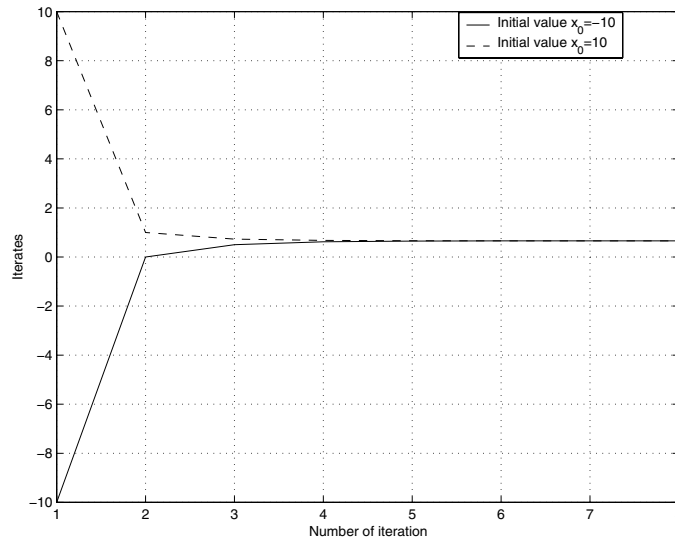


Figure 7.5 FPI for a logistic function and different initial values

Hence

$$|x_i - x^*| \leq \frac{\gamma}{1 - \gamma} |x_{i-1} - x_i|. \tag{7.19}$$

□

Example 7.4.6. Show that when repeatedly applying logistic function Φ the interval $[-10, 10]$ degenerates towards a point $\zeta \in [-10, 10]$.

Solution. Observation 7.4.1 provides a general background for this example. Notice that $\beta = 1$. In order to show that a function converges in the FPI sense, it is sufficient to show that it contracts the bound points of the interval $[-10, 10]$, since it is a strictly monotonically increasing function. Let us therefore set up the iteration

$$x_{i+1} = \Phi(x_i), \quad x_0 \in \{-10, 10\}. \tag{7.20}$$

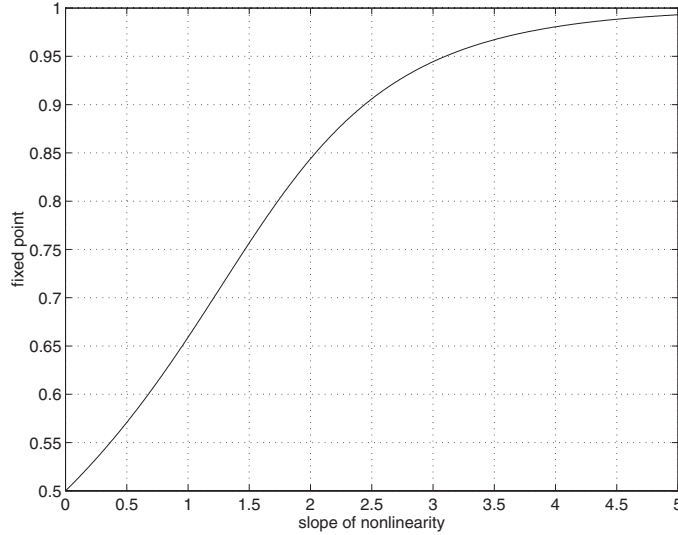


Figure 7.6 Fixed points for the *logistic* nonlinearity, as a function of slope β and starting point $x_0 = 10$

The results of the iteration are given in Table 7.1 and Figure 7.5. As seen from Table 7.1, for both initial values, function Φ provides a contraction of the underlying interval, i.e. it provides a set of mappings

$$\left. \begin{aligned} \Phi : [-10, 10] &\rightarrow [0.000\ 045, 1], \\ \Phi : [0.000\ 045, 1] &\rightarrow [0.5, 0.7311], \\ &\vdots \\ \Phi : \zeta &\rightarrow \zeta. \end{aligned} \right\} \quad (7.21)$$

Indeed, the iterates from either starting point $x_0 \in \{-10, 10\}$ converge to a value $\zeta \in [0.6589, 0.6591] \in [-10, 10]$. It can be shown that after 24 iterations, the fixed point ζ is

$$\Phi : [-10, 10] \xrightarrow{i} \zeta = 0.659\ 046\ 068\ 407\ 41, \quad (7.22)$$

which is shown in Figure 7.5. □

Example 7.4.7. Plot the fixed points of the logistic function

$$\Phi(x) = \frac{1}{1 + e^{-\beta x}} \quad (7.23)$$

for a range of β .

Solution. The result of the experiment is shown in Figure 7.6. From Figure 7.6, the values of the fixed point increase with β and converge to unity when β increases. □

Example 7.4.8. Show that the logistic function from Example 7.4.6, exhibits a linear asymptotic convergence for which the convergence rate is $\gamma = 0.2247$.

Table 7.2 Error convergence for the FPI of the logistic function

	$x_0 = -10$	e_i	e_i/e_{i-1}	$x_0 = 10$	e_i	e_i/e_{i-1}
First iterate	0.000 045	0.659	—	1	0.341	—
Second iterate	0.5	0.159	0.2413	0.7311	0.0721	0.2114
Third iterate	0.6225	0.0365	0.2296	0.6750	0.016	0.2219
Fourth iterate	0.6508	0.0082	0.2247	0.6626	0.0036	0.2246
Fifth iterate	0.6572	0.0018	0.2247	0.6598	0.0008	0.2247
Sixth iterate	0.6586	0.0004	0.2247	0.6592	0.0002	0.2247
Seventh iterate	0.6589	0.0001	0.2247	0.6591	0.0001	0.2247

Solution. To show that the rate of convergence of the iterative process (7.13) is $|\Phi'(x^*)|$, let us calculate $\Phi'(x^*) \approx \Phi'(0.659) = 0.2247$. Let us now upgrade Table 7.1 in order to show the rate of convergence. The results are shown in Table 7.2. As $\Phi'(x^*) \approx 0.2247$, it is expected that, according to CMT, the ratio of successive errors converges to $\Phi'(x^*)$. Indeed, for either initial value in the FPI, the errors $e_i = x_i - x^*$ decrease with the order of iteration and the ratio of successive errors e_{i+1}/e_i converges to 0.2247 and reaches that value after as few iterations as $i = 4$ for $x_0 = -10$ and $i = 5$ for $x_0 = 10$. \square

Properties of the tanh activation function in this context are given in Krčmar *et al.* (2000).

Remark 7.4.9. The function

$$\tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}$$

provides contraction mapping for $0 < \beta < 1$.

This is easy to show, following the analysis for the logistic function and noting that $\tanh'(\beta x) = 4\beta/(e^{-\beta x} + e^{\beta x})^2$, which is strictly positive and for which the maximum value is $\beta = 1$ for $x = 0$. Convergence of FPI for $\beta = 1$ and $\beta = 1.2$ for a tanh activation function is shown in Figure 7.7. The graphs show convergence from two different starting values, $y = -10$ and $y = 10$. For $\beta = 1$, relaxations from both starting values converge towards zero, whereas for $\beta = 1.2$, which is greater than the bound given in Remark 7.4.9, we have two different fixed points. For convergence of learning algorithms for adaptive filters based upon neural networks, we desire only one stable fixed point, and the further emphasis will be on bounds on the weights and nonlinearity which preserve this condition.

7.5 Convergence of Nonlinear Relaxation Equations Realised Through a Recurrent Perceptron

We next analyse convergence towards an equilibrium based upon a recurrent perceptron using contraction mapping and corresponding fixed point iteration. Unlike in the linear case, the external input data to (7.2) do not need to be a zero vector, but simply kept constant.

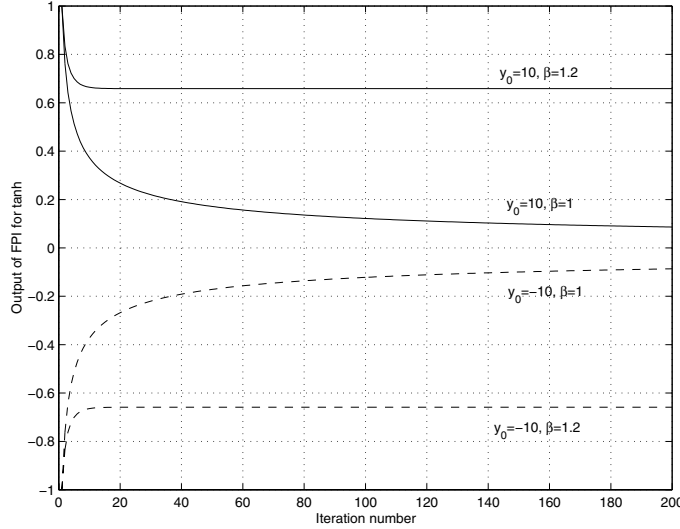


Figure 7.7 Fixed points for the tanh activation function

Proposition 7.5.1 (see Mandic and Chambers 1999b). *GAS relaxation for a recurrent perceptron given by*

$$y(k+1) = \Phi(\mathbf{u}(k)^T \mathbf{w}(k)), \quad (7.24)$$

where $\mathbf{u}_k^T = [y(k-1), \dots, y(k-N), 1, x(k-1), \dots, x(k-M)]$, is a contraction mapping and converges to some value $y^* \in (0, 1)$ for $\beta \sum_{j=1}^N |w_j(k)| < 4$.

Proof. Equation (7.24) can be written as

$$y(k+1) = \Phi\left(\sum_{j=1}^{N+M+1} w_j z_j(k)\right), \quad (7.25)$$

where $z_j(k)$ is the j th element of input $\mathbf{u}(k)$. The iteration (7.25) is biased and can be expressed as

$$y(k+1) = \Phi(y(k), \dots, y(k-N+1), \text{const.}). \quad (7.26)$$

The existence, uniqueness and convergence features of mapping (7.24), follow from properties of the logistic function. Iteration (7.24), for a contractive Φ converges to a fixed point $y^* = \Phi(y^* + \text{const.})$, where the constant is given by

$$\text{const.} = \left(\sum_{j=N+1}^{N+M+1} w_j z_j(k)\right).$$

It is assumed that the weights are not time-variant. Since the condition for convergence of the logistic function to a fixed point is $0 < \beta < 4$, it follows that the slope in the logistic function β and the weights w_1, \dots, w_N in the weight vector \mathbf{w} are not

Table 7.3 Fixed point iterates for the NARMA perceptron

Starting value y_0	-10	10
First iterate	0.006 68	0.795 71
Second iterate	0.445 10	0.520 35
Third iterate	0.487 69	0.494 82
Fourth iterate	0.491 73	0.492 41
Fifth iterate	0.492 11	0.492 18
Sixth iterate	0.492 15	0.492 16

independent and that the effective slope in the logistic function now becomes the product $\beta \sum_{j=1}^N w_j$. Therefore

$$\left| \beta \sum_{j=1}^N w_j \right| \leq \beta \sum_{j=1}^N |w_j| < 4 \quad \Leftrightarrow \quad \|\mathbf{w}\|_1 < \frac{4}{\beta} \quad (7.27)$$

is the condition of GAS convergence of (7.2) realised through a recurrent NARMA perceptron. \square

A comparison of the nonlinear GAS result (7.27) with its linear counterpart shows that they are both based upon the $\|\cdot\|_1$ norm of the corresponding coefficient vector. In the nonlinear case, however, the measure of nonlinearity is also included.

Example 7.5.2. Show that the iteration

$$y(k) = \Phi(y(k-1)) = \frac{1}{1 + e^{-0.25y(k-1)+0.5}} \quad (7.28)$$

with initial values $y_0 = -10$ and $y_0 = 10$ converges towards a point $y^* \in [-10, 10]$.

Solution. Note that $\beta = 0.25$ and $w = 1$. The numerical values for iteration (7.28) are given in Table 7.3. Indeed, the iterates from either starting point converge to a value $y^* \in [0.492\,15, 0.492\,16] \subset [-10, 10]$. It can be shown that after 15 iterations for $y_0 = -10$ and 16 iterations for $y_0 = 10$, the fixed point to which the FPI (7.28) converges is $y^* = 0.492\,159\,969\,021\,68$. \square

Corollary 7.5.3 (see Mandic and Chambers 1999b). *In the case of the realisation of (7.2) by a NARMA recurrent perceptron, convergence towards a point in the FPI sense does not depend on the number of external input signals, nor on their values, as long as they are finite.*

The convergence rate is the ratio of the distances between the current and previous iterate of an FPI and a fixed point y^* , i.e. $(y(k) - y^*) / (y(k-1) - y^*)$. This reveals how quickly an FPI process converges towards a point.

Observation 7.5.4 (see Mandic and Chambers 1999b). *A realisation of an iterative process (7.2) by a recurrent perceptron converges towards a fixed point y^* exhibiting linear convergence with convergence rate $\Phi'(y^*)$ (Example 7.4.8).*

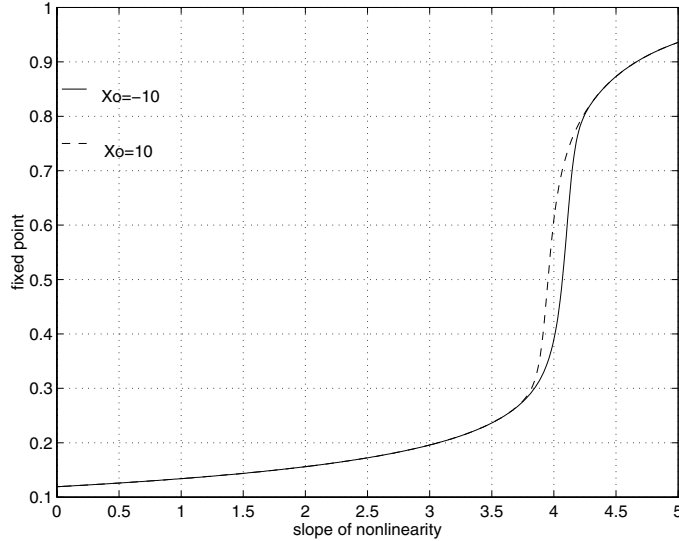


Figure 7.8 Fixed points for the biased *logistic* nonlinearity

Example 7.5.5. Plot the fixed points of the biased logistic function

$$\Phi(x) = \frac{1}{1 + e^{-\beta x + \text{bias}}} \tag{7.29}$$

for a range of β and bias = 2.

Solution. To depict the effects of varying β , noise was added to the system. From Figure 7.8, the values of fixed points increase with β and converge to unity when β increases. However, for β large enough, the fixed points to which the iteration $x_{i+1} = \Phi(x_i)$ converges might not be unique. Indeed, the broken line in Figure 7.8 represents the iteration whose starting value was $x_0 = 10$, while the solid line in Figure 7.8 represents the case with $x_0 = -10$. For a range of β around $\beta = 4$, the iterations from different starting points do not converge to the same value. The values of fixed points for the biased logistic function differ from the corresponding values for the pure logistic function. Moreover, the fixed points differ for various values of the bias in the biased logistic function. \square

Remark 7.5.6. For stability of FPI for a tanh activation function replace the bound $\beta < 4$ by $\beta < 1$, i.e. $\|\mathbf{w}\|_1 < 1/\beta$.

7.6 Relaxation in Nonlinear Systems Realised by an RNN

Let $\mathbf{Y}_i = [y_1^i, \dots, y_N^i]^T$ be a vector comprising the outputs of a general RNN at iteration i of the FPI. The input vector to a network is $\mathbf{u}_i = [y_1^i, \dots, y_N^i, 1, x_{N+1}, \dots, x_{N+M+1}]^T$. The weight matrix \mathbf{W} consists of N rows and $N + M + 1$ columns. Then, by a CMT in \mathbb{R}^N , the iterative process applied on the general RNN converges, if $\mathbf{M} = [a, b]^N$ is a closed subset of \mathbb{R}^N such that

- (i) $\Phi : \mathcal{M} \rightarrow \mathcal{M}$;
(ii) if for some norm $\|\cdot\|$, $\exists \gamma < 1$ such that $\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq \gamma \|\mathbf{x} - \mathbf{y}\|$, $\forall \mathbf{x}, \mathbf{y} \in \mathcal{M}$ the equation

$$\mathbf{x} = \Phi(\mathbf{x}) \quad (7.30)$$

has a unique solution $\mathbf{x}^* \in \mathcal{M}$, and the iteration

$$\mathbf{x}_{i+1} = \Phi(\mathbf{x}_i) \quad (7.31)$$

converges to \mathbf{x}^* for any starting value $\mathbf{x}_0 \in \mathcal{M}$.

Actually, since the function Φ in this case is a multivariate function, $\Phi = [\Phi_1, \dots, \Phi_N]^T$, where N is the number of neurons of the RNN, we have a set of mappings

$$\left. \begin{aligned} y_1^i &= \Phi_1(\mathbf{u}_{i-1}^T \mathbf{W}_1), \\ &\vdots \\ y_N^i &= \Phi_N(\mathbf{u}_{i-1}^T \mathbf{W}_N), \end{aligned} \right\} \quad (7.32)$$

where $\{\mathbf{W}_i\}$ are the appropriate columns in \mathbf{W} . An obvious problem is that the convergence is norm dependent. Therefore, that condition should be replaced by some condition based upon the features of Φ .

Let us denote the Jacobian of Φ by \mathbf{J} . If $\mathcal{M} \in \mathbb{R}^N$ is a convex set and Φ is continuously differentiable on $\mathcal{M} = [a, b]^N \subset \mathbb{R}^N$ and satisfies the conditions of the CMT, then

$$\max_{\mathbf{z} \in \mathcal{M}} \|\mathbf{J}(\mathbf{z})\| \leq \gamma. \quad (7.33)$$

For convergence, the FPI at every neuron should be convergent. The following analysis gives the bound for the elements of the weight matrix \mathbf{W} of the RNN with respect to the derivatives of the components of $\Phi = [\Phi_1, \dots, \Phi_N]$. Recall that for the case of a single recurrent perceptron, the condition for GAS was

$$\sum_{j=1}^N |w_j| < \frac{4}{\beta} \quad \Leftrightarrow \quad \|\mathbf{w}\|_1 < \frac{4}{\beta} = \frac{1}{\Phi'_{\max}}.$$

However, for a network of N neurons, it is possible to have a convergent FPI, even if some of the neurons violate the previous conditions. When it comes to the monotonic convergence, it is important that the process at every neuron converges uniformly. This is straightforward to show, since for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$, which are processed by a neural network, we have

$$\begin{aligned} |\Phi(\mathbf{x}) - \Phi(\mathbf{y})| &= \sum_{i=1}^N \left| \sum_{j=1}^N w_{i,j} \Phi_j(x_j) - \sum_{j=1}^N w_{i,j} \Phi_j(y_j) \right| \\ &\leq \sum_{i=1}^N \sum_{j=1}^N |w_{i,j}| |\Phi_j(x_j) - \Phi_j(y_j)| \\ &\leq \sum_{j=1}^N |\Phi'_{\max}| |x_j - y_j| \sum_{i=1}^N |w_{i,j}|. \end{aligned} \quad (7.34)$$

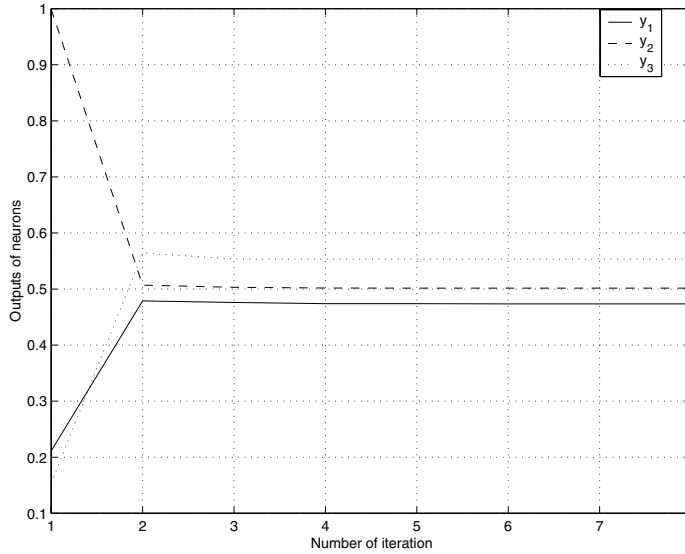


Figure 7.9 FPI for a general RNN

For uniform convergence at every particular neuron, it is the diagonal weights of the weight matrix (self-feedback) which together with the slope β_i have an influence on the convergence in the FPI sense. As in the case of a recurrent NARMA perceptron, the feedback of a general RNN may consist of a number n of delayed versions of its output, in addition to the state feedback from the remaining neurons in the network. In that case, the number of feedback inputs to the network becomes $N + n - 1$ and the condition for GAS becomes

$$\max_{1 \leq k \leq N} \{|w_{k,k}|, |w_{k,N+1}|, \dots, |w_{k,N+n-1}|\} < \frac{4}{(N + n - 1) \max_{1 \leq i \leq N} \beta_i}. \quad (7.35)$$

Observation 7.6.1. *The rate of convergence of relaxation in RNNs does not depend on the length of the tap delay input line.*

Proof. It is already shown that all the variables related to the MA part of the underlying NARMA process form a constant during the FPI iteration, while the feedback variables are updated in every iteration. Hence, no matter how many external input signals, their contribution to the FPI relaxation is embodied in a constant. Therefore, the iteration

$$\mathbf{Y}_{i+1} = \Phi(\mathbf{Y}_i, \mathbf{X}, \mathbf{W}) \quad (7.36)$$

does not depend on the number of external input samples. □

Example 7.6.2. Analyse the convergence of the iteration process for a general RNN with three neurons and six external input signals and a logistic activation function.

Solution. Let us choose the initial values $\mathbf{X}_0 = \text{rand}(10, 1) * 1$, $\mathbf{W} = \text{rand}(10, 3) * 2 - 1$, using the notation of MATLAB, and start the iteration process. Here $\text{rand}(M, N)$

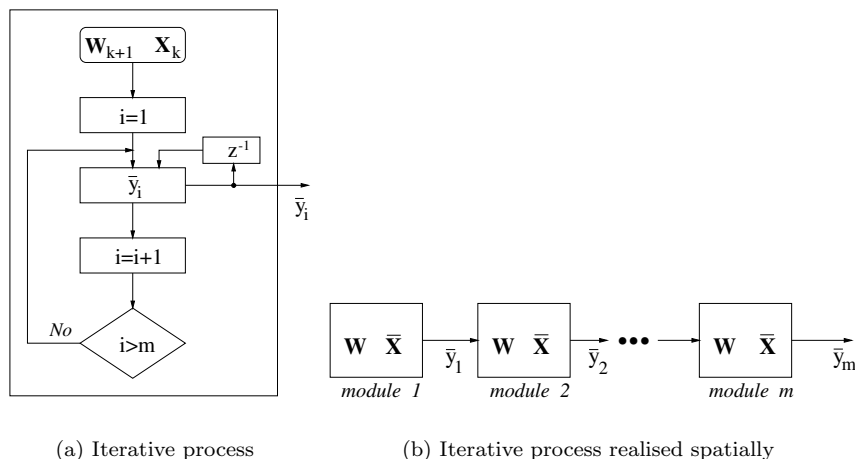


Figure 7.10 Spatial realisation of an iterative process

denotes an $(M \times N)$ -dimensional matrix of uniformly distributed random numbers $\in [0, 1]$. The convergence of the outputs of neurons in the FPI sense is depicted in Figure 7.9. For every neuron, the iteration process converges or, if in vector form, the output vector of the RNN converges to a fixed vector of the iteration. \square

7.7 The Iterative Approach and Nesting

Nesting corresponds to the procedure of reducing the interval size in set theory. In signal processing, however, nesting is essentially a nonlinear spatial structure which corresponds to the cascaded structure in linear signal processing (Baltersee and Chambers 1998; Haykin and Li 1995; Mandic and Chambers 1998b; Mandic *et al.* 1998). The RNN-based nested sigmoid scheme can be written as (Haykin 1994; Poggio and Girosi 1990)

$$F(W, X) = \Phi \left(\sum_n w_n \Phi \left(\sum_i v_i \Phi \left(\dots \Phi \left(\sum_j u_j X_j \right) \dots \right) \right) \right), \quad (7.37)$$

where Φ is a sigmoidal function. This corresponds to a multilayer network of units that sum their inputs with ‘weights’ $W = \{w_n, v_i, \dots, u_j, \dots\}$ and then perform a sigmoidal transformation of this sum. Our aim is to show that nesting can exhibit contraction mapping and that repeatedly applied nesting can lead to convergence in the FPI sense. Therefore, instead of having a spatial, nested, pipelined structure, nesting can be obtained through a temporal, iterative, relaxive structure (Mandic and Chambers 2000c), as shown in Figure 7.10. Quantities that change under iteration in Figure 7.10 have a bar above the symbol.

Observation 7.7.1. *The compound nested logistic functions*

$$\begin{aligned}
 \hat{x} &= \Phi(x_N) \\
 &= \Phi(\Phi(x_{N-1})) \\
 &\quad \vdots \\
 &= \Phi\left(\underbrace{\Phi(\Phi(\dots(\Phi(x_1))\dots))}_N\right)
 \end{aligned} \tag{7.38}$$

provide a contraction mapping for $\beta < 4$ and the FPI converges towards a point $x^* \in [a, b]$.

Proof. Notice that the nesting process (7.38) represents an implicitly written fixed point iteration process

$$x_{i+1} = \Phi(x_i) \quad \Leftrightarrow \quad x_{i+1} = \Phi(\Phi(x_{i-1})) = \Phi\left(\underbrace{\Phi(\Phi(\dots(\Phi(x_1))\dots))}_N\right). \tag{7.39}$$

Hence, nesting (7.38) and fixed point iteration (7.13) are a realisation of the same process and have already been considered. Let us therefore just show the diagram of the effects of the nesting process for the logistic function with slope $\beta = 1$, depicted in Figure 7.11. From Figure 7.11, it is apparent that nesting (7.38) provides contraction mapping of its argument. Hence, it is expected that the nesting process (7.38) with N stages converges towards the point $x^* \in [|\Phi'(x^*)|^N a, |\Phi'(x^*)|^N b]$. For N small, the fixed point iteration achieved through a nesting process (7.38) may not reach its fixed point. However, from Tables 7.1 and 7.2 and Figure 7.11, even with $N = 4$, the error $|x_4 - x^*| < 0.01$, which suffices for practical applications. \square

To summarise:

- for the nesting process to be a contraction mapping, the range of slopes β for the logistic function Φ should be bounded, with $0 < \beta < 4$;
- the nesting process of a sufficient order applied to an interval $[a, b] \in \mathbb{R}$ converges to a point $x^* \in [a, b]$, which is a fixed point of the fixed point iteration $x_{i+1} = \Phi(x_i)$;
- the nesting process (7.38) exhibits a *linear asymptotic convergence* whose rate is $|\Phi'(x^*)|$, where x^* is the fixed point of mapping Φ .

The nesting process (7.38) provides the iteration spatially, rather than temporally. Such a strategy is known as *pipelining* and is widely used in advanced computer architectures (Hwang and Briggs 1986). Using the *pipelining* strategy, a task is divided in subtasks, each of them being represented by a module. Pipelining corresponds to unfolding the finite iterative process into a spatial structure of the same length as the number of iterations in the former. Now, from (7.37), the pipelined structure represents indeed a spatial realisation of an essentially temporal iterative process, and converges under the same conditions as the nesting process (7.38). A realisation of

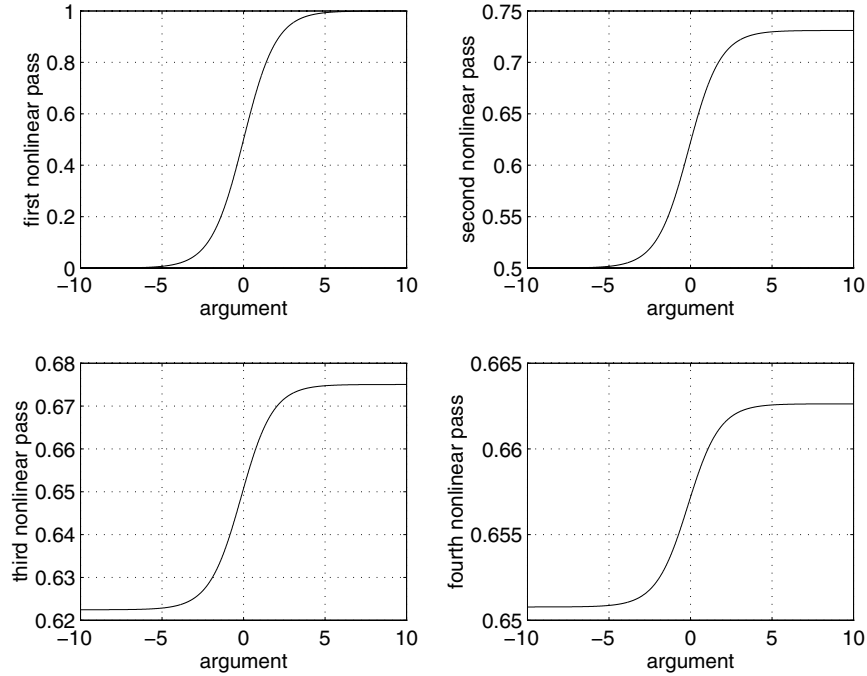


Figure 7.11 Nested logistic nonlinearity

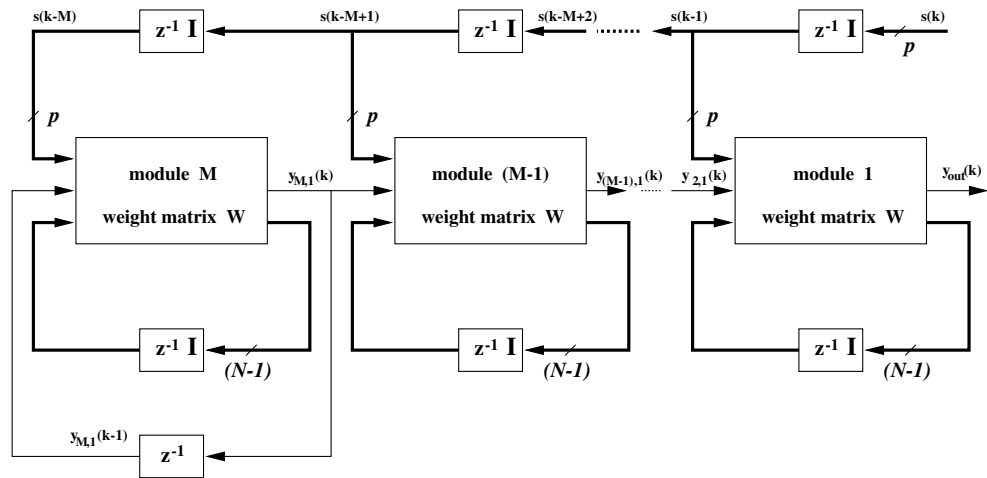


Figure 7.12 Pipelined Recurrent Neural Network

process (7.38) is the so-called pipelined recurrent neural network (PRNN) (Haykin and Li 1995), shown in Figure 7.12, which provides a spatial form of the iteration (7.37). Therefore, for instance, instead of having a temporal FPI on a recurrent perceptron (Figure 6.2), it suffices, for a finite-length FPI, to consider a spatial PRNN structure.

7.8 Upper Bounds for GAS Relaxation within FCRNNs

Neural systems of the form

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\sigma[\mathbf{W}\mathbf{x}(k) + \mathbf{s}] \quad (7.40)$$

have been widely considered (Jin *et al.* 1994). Here, \mathbf{x} is the state vector of the network and $\sigma(\cdot)$ is a vector of nonlinear activation functions. On the other hand, the weight matrix \mathbf{W} of a recurrent neural network can be split up into the feedback part (index a) and the feedforward part (index b), which gives $\mathbf{Y}(k+1) = \Phi(\mathbf{W}_a\mathbf{Y}(k) + \mathbf{W}_b\mathbf{x}(k))$, which can degenerate into the form (7.40). Namely, for a contractive activation function Φ , we have (Mandic and Chambers 2000e)

$$\Phi(a+b) < \Phi(a) + \Phi(b) < a + \Phi(b), \quad (7.41)$$

and results for system (7.40) provide the upper bound for stability of the fully connected RNN system described above (Mandic *et al.* 2000).

7.9 Summary

The relationships between the number of neurons in the RNN, the slope in the activation function β and a measure of \mathbf{W} have been provided, which guarantee convergence of a relaxation process realised by fully connected recurrent neural networks. Based upon the fixed point iteration (FPI), it has been shown that these conditions rest entirely upon the slope of the activation function β and a measure of the $\|\cdot\|_1$ norm of the weight vector of a recurrent perceptron. A connection between nesting and FPI, which is the basis of the GAS convergence, has been established, and a pipelined recurrent neural network (PRNN) has been shown to be a spatial realisation of the FPI process. The results obtained can be applied when recurrent neural networks are used as computational models, in particular, as optimisation models. The results can also be used as stability analysis tools for some classes of nonlinear control systems.

8

Data-Reusing Adaptive Learning Algorithms

8.1 Perspective

In this chapter, a class of data-reusing learning algorithms for recurrent neural networks is analysed. This is achieved starting from a case of feedforward neurons, through to the case of networks with feedback, trained with gradient descent learning algorithms. It is shown that the class of data-reusing algorithms outperforms the standard (*a priori*) algorithms for nonlinear adaptive filtering in terms of the instantaneous prediction error. The relationships between the *a priori* and *a posteriori* errors, learning rate and the norm of the input vector are derived in this context.

8.2 Introduction

The so-called *a posteriori* error estimates provide us with, roughly speaking, some information after computation. From a practical point of view, they are valuable and useful, since real-life problems are often nonlinear, large, ill-conditioned, unstable or have multiple solutions and singularities (Hlavacek and Krizek 1998). The *a posteriori* error estimators are local in a computational sense, and the computational complexity of *a posteriori* error estimators should be far less expensive than the computation of an exact numerical solution of the problem. An account of the essence of *a posteriori* techniques is given in Appendix F.

In the area of linear adaptive filters, the most comprehensive overviews of *a posteriori* techniques can be found in Treichler (1987) and Ljung and Soderstrom (1983). These techniques are also known as data-reusing techniques (Douglas and Rupp 1997; Roy and Shynk 1989; Schnauffer and Jenkins 1993; Sheu *et al.* 1992). The quality of an *a posteriori* error estimator is often measured by its *efficiency index*, i.e. the ratio of the estimated error to the true error. It has been shown that the *a posteriori* approach in the neural network framework introduces a kind of normalisation of the employed learning algorithm (Mandic and Chambers 1998c). Consequently, it is expected that the instantaneous *a posteriori* output error $\bar{e}(k)$ is smaller in magnitude than the

corresponding *a priori* error $e(k)$ for a non-expansive nonlinearity Φ (Mandic and Chambers 1998c; Treichler 1987).

8.2.1 Towards an *A Posteriori* Nonlinear Predictor

To obtain an *a posteriori* RNN-based nonlinear predictor, let us, for simplicity, consider a NARMA recurrent perceptron, the output of which can be expressed as

$$y(k) = \Phi(\mathbf{u}^T(k)\mathbf{w}(k)), \quad (8.1)$$

where the *information vector*

$$\mathbf{u}(k) = [x(k-1), \dots, x(k-M), 1, y(k-1), \dots, y(k-N)]^T \quad (8.2)$$

comprises both the external input and feedback signals. As the updated weight vector $\mathbf{w}(k+1)$ is available before the arrival of the next input vector $\mathbf{u}(k+1)$, an *a posteriori* output estimate $\bar{y}(k)$ can be formed as

$$\bar{y}(k) = \Phi(\mathbf{u}^T(k)\mathbf{w}(k+1)). \quad (8.3)$$

The corresponding instantaneous *a priori* and *a posteriori* errors at the output neuron of a neural network are given, respectively, as

$$e(k) = d(k) - y(k) \quad \text{a priori error}, \quad (8.4)$$

$$\bar{e}(k) = d(k) - \bar{y}(k) \quad \text{a posteriori error}, \quad (8.5)$$

where $d(k)$ is some teaching signal. The *a posteriori* outputs (8.3) can be used to form an *a posteriori information vector*

$$\bar{\mathbf{u}}(k) = [x(k-1), \dots, x(k-M), 1, \bar{y}(k-1), \dots, \bar{y}(k-N)]^T, \quad (8.6)$$

which can replace the *a priori* information vector (8.2) in the output (8.3) and weight update calculations (6.43)–(6.45). This also results in greater accuracy (Ljung and Soderstrom 1983). An alternate representation of such an algorithm is the so-called *a posteriori* error gradient descent algorithm (Ljung and Soderstrom 1983; Treichler 1987), explained later in this chapter.

A simple data-reusing algorithm for linear adaptive filters

The procedure of calculating the instantaneous error, output and weight update may be repeated for a number of times, keeping the same external input vector $\mathbf{x}(k)$ and teaching signal $d(k)$, which results in improved error estimation. Let us consider such a *data-reusing* LMS algorithm for FIR adaptive filters, described by (Mandic and Chambers 2000e)

$$\left. \begin{aligned} e_i(k) &= d(k) - \mathbf{x}^T(k)\mathbf{w}_i(k), \\ \mathbf{w}_{i+1}(k) &= \mathbf{w}_i(k) + \eta e_i(k)\mathbf{x}(k), \\ \text{subject to } |e_{i+1}(k)| &\leq \gamma |e_i(k)|, \quad 0 < \gamma < 1, \quad i = 1, \dots, L. \end{aligned} \right\} \quad (8.7)$$

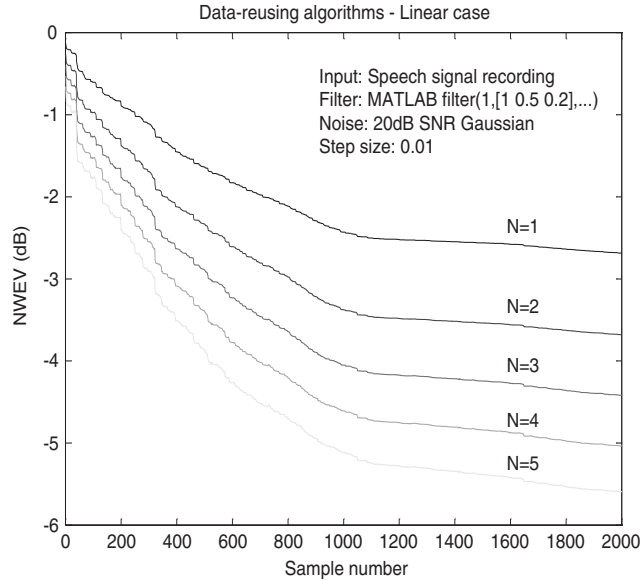


Figure 8.1 Convergence curves for a repeatedly applied data-reusing algorithm

From (8.7), $\mathbf{w}(k+1)$ is associated with the index $(L+1)$, i.e. $\mathbf{w}(k+1) = \mathbf{w}_{L+1}(k)$, whereas for $L=1$, the problem reduces to the standard *a priori* algorithm, i.e. $\mathbf{w}_1(k) = \mathbf{w}(k)$, $\mathbf{w}_2(k) = \mathbf{w}(k+1)$. Convergence curves for such a reiterated LMS algorithm for a data-reusing FIR filter applied to echo cancellation are shown in Figure 8.1. The averaged squared prediction error becomes smaller with the number of iterations, N . For $N \rightarrow \infty$, the prediction error becomes the one of the NLMS¹ algorithm. A geometrical perspective of the procedure (8.7) is given in Appendix F and Figures F.2 and F.3. This provides advantageous stabilising features as compared to standard algorithms. This is further elaborated in Section F.2.2 of Appendix F. In practice, however, the advantage of the *a posteriori* algorithms is not always significant, and depends on the physics of the problem and the chosen filter.

8.2.2 Note on the Computational Complexity

It has been shown that the computational complexity of the *a priori* RTRL algorithm is $\mathcal{O}(N^4)$ (Haykin 1994; Williams and Zipser 1995), with N denoting the number of neurons in the RNN. If, in order to improve the performance, the number of neurons in the network is increased from N to $(N+1)$, the time required for the new adaptation process to finish can be dramatically increased. To depict that problem, the relative change in the computational load when the number of neurons increases, i.e. the ratio $(N+1)^4/N^4$, is shown in Figure 8.2. In other words, that means that the *a posteriori*

¹ In fact, for the linear case, the NLMS algorithm is approached by repeating this kind of data-reusing for an infinite number of times (Nitzberg 1985; Roy and Shynk 1989; Schnaufer and Jenkins 1993). For further details, see Appendix F.

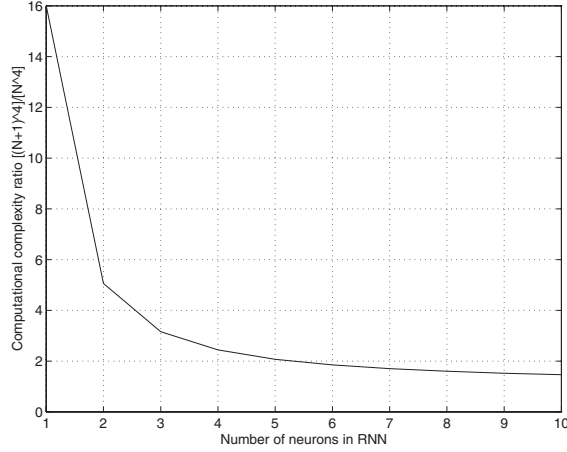


Figure 8.2 Ratio of the increase of computational burden with N

procedure applied to the network with N neurons should have the computational load \mathcal{C}_L given by

$$\mathcal{C}_L(N^4) \leq \mathcal{C}_L \text{ a posteriori} < \mathcal{C}_L((N+1)^4). \quad (8.8)$$

8.2.3 Chapter Summary

A detailed account of various data-reusing techniques for nonlinear adaptive filters realised as neural networks is provided. The relationships between the *a priori* and *a posteriori* errors are derived and the corresponding bounds on learning rates are analysed. This class of algorithms performs better than standard algorithms, does not introduce a significant additional computational burden, and for a class of data-reusing algorithms, when iterated for an infinite number of times, converges to a class of normalised algorithms.

8.3 A Class of Simple *A Posteriori* Algorithms

Consider a simple computational model of a feedforward neural adaptive filter shown in Figure 6.4. The aim is to preserve

$$|\bar{e}(k)| \leq \gamma |e(k)|, \quad 0 \leq \gamma < 1 \quad (8.9)$$

at each iteration, for both feedforward and recurrent neural networks acting as a nonlinear predictor. The problem of obtaining the *a posteriori* error can be represented in the gradient descent setting as (Mandic and Chambers 2000e)

$$\left. \begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) - \eta \nabla_{\mathbf{w}} E(k), \\ \bar{e}(k) &= d(k) - \Phi(\mathbf{x}^T(k) \mathbf{w}(k+1)), \\ \text{subject to } |\bar{e}(k)| &\leq \gamma |e(k)|, \quad 0 < \gamma < 1. \end{aligned} \right\} \quad (8.10)$$

From (8.10), the actual learning is performed in the standard manner, i.e. *a priori* using $e(k)$, whereas an improved *a posteriori* error $\bar{e}(k)$ is calculated at every discrete time interval using the updated weight vector $\mathbf{w}(k+1)$. The gradient descent algorithm for this computational model, with the cost function in the form of $E(k) = \frac{1}{2}e^2(k)$, is given by

$$\left. \begin{aligned} e(k) &= d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)), \\ \mathbf{w}(k+1) &= \mathbf{w}(k) + \eta(k)e(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}(k))\mathbf{x}(k), \\ \bar{e}(k) &= d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)). \end{aligned} \right\} \quad (8.11)$$

This case represents a generalisation of the LMS algorithm for FIR adaptive linear filters. Let us express the *a posteriori* error term from above as

$$\bar{e}(k) = d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)) - [\Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k))]. \quad (8.12)$$

Using the CMT, for a contractive, monotonically increasing Φ and positive $e(k)$ and $\bar{e}(k)$, we have

$$\Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)) = \alpha(k)\mathbf{x}^T(k)\Delta\mathbf{w}(k), \quad (8.13)$$

where $\alpha(k) = \Phi'(\xi) < 1$, $\xi \in (\mathbf{x}^T(k)\mathbf{w}(k), \mathbf{x}^T(k)\mathbf{w}(k+1))$. Using (8.11)–(8.13) yields

$$\bar{e}(k) = [1 - \eta(k)\alpha(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2]e(k), \quad (8.14)$$

where $\Phi'(k) = \Phi'(\mathbf{x}^T(k)\mathbf{w}(k))$. The learning rate

$$\eta(k) = \frac{1}{\alpha(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2},$$

which minimises (8.14), is approximately that of a normalised nonlinear gradient descent algorithm (9.15), given in Chapter 9.

To obtain the bounds of such an *a posteriori* error, premultiplying the weight update equation in (8.11) by $\mathbf{x}^T(k)$ and applying the nonlinear activation function Φ on either side yields (Mandic and Chambers 2000e)

$$\Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) = \Phi(\mathbf{x}^T(k)\mathbf{w}(k) + \eta(k)e(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2). \quad (8.15)$$

Further analysis depends on the function Φ , which can exhibit either contractive or expansive behaviour. For simplicity, let us consider a class of contractive functions Φ , which satisfy²

$$\Phi(a+b) \leq \Phi(a) + \Phi(b). \quad (8.16)$$

With $a = \mathbf{x}^T(k)\mathbf{w}(k)$ and $b = \eta(k)e(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2$, applying (8.16) to (8.15) and subtracting $d(k)$ from both sides of the resulting equation, due to contractivity of Φ , we obtain

$$\bar{e}(k) \geq e(k) - \Phi(\eta(k)e(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2). \quad (8.17)$$

² This is the case, for instance, for many sigmoid functions. For many other functions this is satisfied in a certain range of interest. For instance, for $a = 0$, positive b and a saturating, monotonically increasing, positive sigmoid, $\Phi(a+b) < \Phi(b) < b$. The condition $\Phi(a+b) \leq \Phi(a) + \Phi(b)$ is satisfied for the logistic function on all of its range and for the positive range of the tanh activation function. For many other functions, $|\Phi(a+b)| \leq |\Phi(a) + \Phi(b)|$.

For Φ a contraction, $|\Phi(\xi)| < |\xi|$, $\forall \xi \in \mathbb{R}$, and (8.17) finally becomes

$$\bar{e}(k) > [1 - \eta(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2]e(k), \quad (8.18)$$

which is the lower bound for the *a posteriori* error for a contractive nonlinear activation function. In this case, the range allowed for the learning rate $\eta(k)$ in (8.18) with constraint (8.9) is³

$$0 < \eta(k) < \frac{1}{\Phi'(k)\|\mathbf{x}(k)\|_2^2}. \quad (8.19)$$

For Φ a linear function,

$$0 < \eta(k) < \frac{1}{\|\mathbf{x}(k)\|_2^2}, \quad (8.20)$$

which boils down to the learning rate of the NLMS algorithm. Therefore, the *a posteriori* algorithm in this context introduces a kind of normalisation of the corresponding learning algorithm.

8.3.1 The Case of a Recurrent Neural Filter

In this case, the gradient updating equation regarding a recurrent perceptron can be symbolically expressed as (Haykin 1994) (see Appendix D)

$$\frac{\partial y(k)}{\partial \mathbf{w}(k)} = \mathbf{\Pi}(k+1) = \Phi'(\mathbf{u}^T(k)\mathbf{w}(k))[\mathbf{u}(k) + \mathbf{w}_a(k)\mathbf{\Pi}(k)], \quad (8.21)$$

where the vector $\mathbf{\Pi}$ denotes the set of corresponding gradients of the output neuron and the vector $\mathbf{u}(k)$ encompasses both the external and feedback inputs to the recurrent perceptron. The correction to the weight vector at the time instant k becomes

$$\Delta \mathbf{w}(k) = \eta(k)e(k)\mathbf{\Pi}(k). \quad (8.22)$$

Following the same principle as for feedforward networks, the lower bound for the *a posteriori* error algorithm in single-node recurrent neural networks with a contractive activation function is obtained as

$$\bar{e}(k) > [1 - \eta(k)\mathbf{u}^T(k)\mathbf{\Pi}(k)]e(k), \quad (8.23)$$

whereas the corresponding range allowed for the learning rate $\eta(k)$ is given by

$$0 < \eta(k) < \frac{1}{|\mathbf{u}^T(k)\mathbf{\Pi}(k)|}. \quad (8.24)$$

³ Condition (8.18) is satisfied for any $\eta > 0$. However, we want to preserve $|\bar{e}(k)| < |e(k)|$ (8.10), with the constraint that both $\bar{e}(k)$ and $e(k)$ have the same sign, and hence the learning rate η has to satisfy (8.19).

8.3.2 The Case of a General Recurrent Neural Network

For recurrent neural networks of the Williams–Zipser type (Williams and Zipser 1989a), with N neurons, one of which is the output neuron, the weight matrix update for an RTRL training algorithm can be expressed as

$$\Delta \mathbf{W}(k) = \eta(k)e(k) \frac{\partial y_1(k)}{\partial \mathbf{W}(k)} = \eta(k)e(k) \mathbf{II}_1(k), \quad (8.25)$$

where $\mathbf{W}(k)$ represents the weight matrix and

$$\mathbf{II}_1(k) = \frac{\partial y_1(k)}{\partial \mathbf{W}(k)}$$

is the matrix of gradients at the output neuron

$$\pi_{n,l}^1(k) = \frac{\partial y_1(k)}{\partial w_{n,l}},$$

where the index n runs along the N neurons in the network and the index l runs along the inputs to the network. This equation is similar to the one for a recurrent perceptron, with the only difference being that weight matrix \mathbf{W} replaces weight vector \mathbf{w} and gradient matrix $\mathbf{II} = [\mathbf{II}_1, \dots, \mathbf{II}_N]$ replaces gradient vector \mathbf{II} . Notice that in order to update matrix \mathbf{II}_1 , a modified version of (8.21) has to update gradient matrices $\mathbf{II}_i, i = 2, \dots, N$. More details about this procedure can be found in Williams and Zipser (1989a) and Haykin (1994).

The lower bound for the *a posteriori* error obtained by an *a priori* learning – *a posteriori* error RTRL algorithm (8.25) with constraint (8.9), and a contractive nonlinear activation function Φ – is therefore

$$\bar{e}(k) > [1 - \eta(k) \mathbf{u}^T(k) \mathbf{II}_1(k)] e(k), \quad (8.26)$$

whereas the range of allowable learning rates $\eta(k)$ is

$$0 < \eta(k) < \frac{1}{|\mathbf{u}^T(k) \mathbf{II}_1(k)|}. \quad (8.27)$$

8.3.3 Example for the Logistic Activation Function

It is shown in Chapter 7 that the condition for the logistic activation function to be a contraction is $\beta < 4$. As such a function is monotone and ascending, the bound on its first derivative is $\Phi'(\xi) \leq \beta/4, \forall \xi \in \mathbb{R}$. That being the case, the bounds on the *a posteriori* error and learning rate for the feedforward case become, respectively,

$$\bar{e}(k) > \frac{1}{4} [4 - \eta(k) \beta \|\mathbf{x}(k)\|_2^2] e(k) \quad (8.28)$$

and

$$0 < \eta(k) < \frac{4}{\beta \|\mathbf{x}(k)\|_2^2}. \quad (8.29)$$

Similar conditions can be derived for the recurrent case. Further relationships between η , β and \mathbf{w} are given in Chapter 12.

8.4 An Iterated Data-Reusing Learning Algorithm

This class of algorithms employs L reuses of the weight update per sample and is a nonlinear version of algorithm (8.7). A data-reusing gradient descent algorithm for a nonlinear FIR filter is given by (Douglas and Rupp 1997; Mandic and Chambers 1998c)

$$\left. \begin{aligned} e_i(k) &= d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}_i(k)), & i = 1, \dots, L, \\ \mathbf{w}_{i+1}(k) &= \mathbf{w}_i(k) + \eta(k)e_i(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_i(k))\mathbf{x}(k), \\ \text{subject to } |e_{i+1}(k)| &\leq \gamma|e_i(k)|, & 0 < \gamma < 1, i = 1, \dots, L, \end{aligned} \right\} \quad (8.30)$$

where $\mathbf{w}_i(k)$ is the weight vector at the i th iteration of (8.30), $\mathbf{x}(k)$ is the input vector, $d(k)$ is some teaching signal and $e_i(k)$ is the prediction error from the i th iteration of (8.30). For $L = 1$, the problem reduces to the standard *a priori* algorithm, whereas $\mathbf{w}(k+1)$ is associated with the index $(L+1)$, i.e.

$$\left. \begin{aligned} \mathbf{w}_1(k) &= \mathbf{w}(k), \\ \mathbf{w}_{L+1}(k) &= \mathbf{w}(k+1). \end{aligned} \right\} \quad (8.31)$$

Starting from the last iteration in (8.30), i.e. for $i = L$, we obtain

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}_{L+1}(k) = \mathbf{w}_L(k) + \eta(k)e_L(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_L(k))\mathbf{x}(k) \\ &= \mathbf{w}_{L-1}(k) + \eta(k)e_{L-1}(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_{L-1}(k))\mathbf{x}(k) \\ &\quad + \eta(k)e_L(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_L(k))\mathbf{x}(k) \\ &= \mathbf{w}(k) + \sum_{i=1}^L \eta(k)e_i(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_i(k))\mathbf{x}(k). \end{aligned} \quad (8.32)$$

Consider the expression for the instantaneous error from the $(i+1)$ th iteration at the output neuron

$$\begin{aligned} e_{i+1}(k) &= d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}_{i+1}(k)) \\ &= [d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}_i(k))] - [\Phi(\mathbf{x}^T(k)\mathbf{w}_{i+1}(k)) - \Phi(\mathbf{x}^T(k)\mathbf{w}_i(k))]. \end{aligned} \quad (8.33)$$

The second term on the right-hand side of (8.33) depends on the function Φ , which can exhibit either contractive or expansive behaviour (Appendix G). For a contractive Φ , assuming positive quantities, $\exists \alpha(k) = \Phi'(\xi)$, $\xi \in (\mathbf{x}^T(k)\mathbf{w}_i(k), \mathbf{x}^T(k)\mathbf{w}_{i+1}(k))$ such that the right-hand term in square brackets from (8.33) can be replaced by $\alpha(k)\mathbf{x}^T(k)\Delta\mathbf{w}_i(k)$, which yields

$$e_{i+1}(k) = e_i(k)[1 - \eta(k)\alpha(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_i(k))\|\mathbf{x}(k)\|_2^2]. \quad (8.34)$$

To calculate the bound on such an error, premultiplying the first equation in (8.30) by $\mathbf{x}^T(k)$ and applying the nonlinear activation function Φ on either side yields

$$\Phi(\mathbf{x}^T(k)\mathbf{w}_{i+1}(k)) = \Phi(\mathbf{x}^T(k)\mathbf{w}_i(k) + \eta(k)e_i(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_i(k))\mathbf{x}(k))\|_2^2. \quad (8.35)$$

Further analysis depends on whether Φ is a contraction or an expansion. It is convenient to assume that $e_i(k)$, $i = 1, \dots, L$, have the same sign during iterations (Appendix F, Figure F.3). From (8.15)–(8.18), we have

$$e_{i+1}(k) > [1 - \eta(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_i(k))\|\mathbf{x}(k)\|_2^2]e_i(k) \quad (8.36)$$

from iteration to iteration of (8.30). Assume that

$$\Phi'(k) \approx \Phi'(\mathbf{x}^T(k)\mathbf{w}_1(k)) \approx \dots \approx \Phi'(\mathbf{x}^T(k)\mathbf{w}_L(k)),$$

then after L iterations⁴ of (8.36), we have

$$e(k+1) > [1 - \eta(k)\Phi'(k)\|\mathbf{x}(k)\|_2^{2L}]e(k). \quad (8.37)$$

The term in the square brackets from above has its modulus less than unity. In that case, the whole procedure is a fixed point iteration, whose convergence is given in Appendix G.

From (8.37) and the condition $|\bar{e}(k)| < |e(k)|$, the range allowed for the learning rate $\eta(k)$ in the data-reusing adaptation (8.30) is

$$0 < \eta(k) < \frac{1}{\Phi'(k)\|\mathbf{x}(k)\|_2^2}. \quad (8.38)$$

8.4.1 The Case of a Recurrent Predictor

The correction to the weight vector of the j th neuron, at the time instant k becomes

$$\Delta\mathbf{w}_j(k) = \eta(k)e(k)\mathbf{II}_1^{(j)}(k), \quad (8.39)$$

where $\mathbf{II}_1^{(j)}(k)$ represents the j th row of the gradient matrix $\mathbf{II}_1(k)$. From the above analysis

$$0 < \eta(k) < \max_j \frac{1}{|\mathbf{u}^T(k)\mathbf{II}_1^{(j)}(k)|}. \quad (8.40)$$

8.5 Convergence of the *A Posteriori* Approach

In the case of nonlinear adaptive filters, there is generally no Wiener solution, and hence the convergence is mainly considered through Lyapunov stability (DeRusso *et al.* 1998; Zurada and Shen 1990), or through contraction mapping (Mandic and Chambers 1999b). Here, due to the assumption that for this class of data-reusing algorithms, the *a priori* and the *a posteriori* errors have the same sign through the data-reusing fixed point iteration, and $|\bar{e}(k)| < |e(k)|$, convergence of the *a posteriori* (data-reusing) error algorithm is defined by convergence of the underlying *a priori* error learning algorithm, which is detailed in Chapter 10. The limit behaviour of the above class of algorithms can be achieved for the infinite number of data-reuse iterations, i.e. when

⁴ The term in the square brackets from (8.37) is strictly less than unity and becomes smaller with L . Also, $e(k) = e_1(k)$, $e(k+1) = e_{L+1}(k)$. In fact, the relation (8.36) represents a fixed point iteration, which, due to CMT, converges for $|1 - \eta(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}_i(k))\|\mathbf{x}(k)\|_2^2| < 1$.

$L \rightarrow \infty$. In that case, for instance, $e_i(k) > [1 - \eta(k)\Phi'(k)\|\mathbf{x}(k)\|_2^2]^{i-1}e(k)$, which from (8.36) forms a geometric series, which converges to a normalised nonlinear gradient descent algorithm (Figure F.3), and consequently the ratio $e_{i+1}(k)/e_i(k) \rightarrow 0$.

8.6 A Posteriori Error Gradient Descent Algorithm

The *a posteriori* outputs (8.3) can be used to form an updated *a posteriori* information vector

$$\bar{\mathbf{u}}(k) = [x(k-1), \dots, x(k-M), 1, \bar{y}(k-1), \dots, \bar{y}(k-N)]^T, \quad (8.41)$$

which can replace the *a priori* information vector (8.2) in the output (8.3) and weight update calculations (6.43)–(6.45). An alternate representation of such an algorithm is the so-called *a posteriori* error gradient descent algorithm (Ljung and Soderstrom 1983; Treichler 1987), which is the topic of this section. Since the updated weight vector $\mathbf{w}(k+1)$ is available before the new input vector $\mathbf{x}(k+1)$ arrives, an *a posteriori* error gradient can be expressed as (Douglas and Rupp 1997; Ljung and Soderstrom 1983; Treichler 1987)

$$\bar{\nabla}_{\mathbf{w}}(\frac{1}{2}\bar{e}^2(k)) = \frac{\partial(\frac{1}{2}\bar{e}^2(k))}{\partial\mathbf{w}(k+1)}. \quad (8.42)$$

Using the above expression and, for simplicity, constraining the *a posteriori* information vector $\bar{\mathbf{u}}(k)$ to the case of a nonlinear dynamical neuron without feedback yields (Ljung and Soderstrom 1983; Treichler 1987)

$$\frac{\partial(\frac{1}{2}\bar{e}^2(k))}{\partial\mathbf{w}(k+1)} = -\Phi'(\mathbf{x}^T(k)\mathbf{w}(k+1))\bar{e}(k)\mathbf{x}(k). \quad (8.43)$$

The *a posteriori* error can be now expressed as (Mandic and Chambers 1998b,c)

$$\begin{aligned} \bar{e}(k) &= d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) \\ &= d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)) + \Phi(\mathbf{x}^T(k)\mathbf{w}(k)) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) \\ &= e(k) - [\Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k))], \end{aligned} \quad (8.44)$$

which contains terms with the time index $(k+1)$. Let us therefore express the term⁵

$$\Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) = \Phi(\mathbf{x}^T(k)\mathbf{w}(k) + \mathbf{x}^T(k)\Delta\mathbf{w}(k)) \quad (8.45)$$

via its first-order Taylor expansion about the point $\mathbf{x}^T(k)\mathbf{w}(k)$ as

$$\begin{aligned} \Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) &\approx \Phi(\mathbf{x}^T(k)\mathbf{w}(k)) + \frac{\partial\Phi(\mathbf{x}^T(k)\mathbf{w}(k))}{\partial\mathbf{w}(k)}\Delta\mathbf{w}(k) \\ &= \Phi(\mathbf{x}^T(k)\mathbf{w}(k)) + \eta\bar{e}(k)\Phi'^2(k)\mathbf{x}^T(k)\mathbf{x}(k), \end{aligned} \quad (8.46)$$

⁵ Notice that using Lipschitz continuity of Φ , the modulus of the term on the right-hand side of (8.44), i.e. $[\Phi(\mathbf{x}^T(k)\mathbf{w}(k+1)) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k))]$ is bounded from above by

$$|\eta\bar{e}(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}(k+1))\mathbf{x}^T(k)\mathbf{x}(k)|.$$

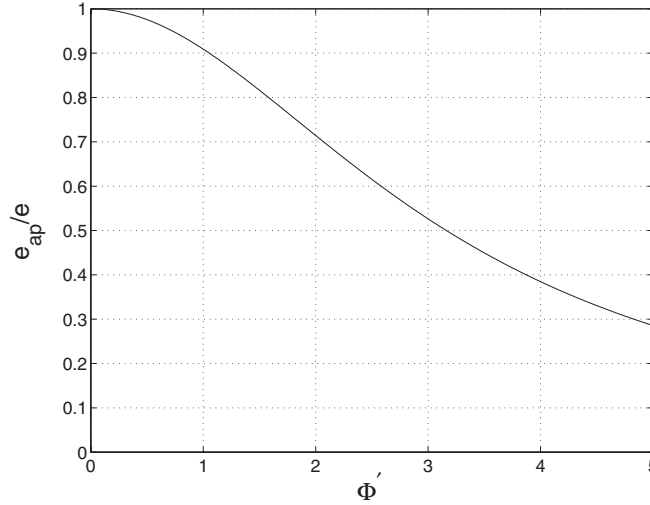


Figure 8.3 Ratio between the *a posteriori* and *a priori* errors for various slopes of the activation function

where $\Phi'(k) = \Phi'(\mathbf{x}^T(k)\mathbf{w}(k)) \approx \Phi'(\mathbf{x}^T(k)\mathbf{w}(k+1))$. Now, combining (8.44) and (8.46) yields the *a posteriori* error

$$\bar{e}(k) = \frac{e(k)}{1 + \eta\Phi'^2(k)\mathbf{x}^T(k)\mathbf{x}(k)} = \frac{e(k)}{1 + \eta\Phi'^2(k)\|\mathbf{x}(k)\|_2^2}. \quad (8.47)$$

The *a posteriori* error $\bar{e}(k)$ (8.47) is smaller in magnitude than the corresponding *a priori* error $e(k)$, since the denominator of (8.47) is strictly greater than unity. For expansive activation functions, $\Phi' > 1$ and the effect described by (8.47) is even more emphasised. However, the *a priori* error in this case can be quite large in magnitude. The effective learning rate for this algorithm becomes⁶

$$\eta(k) = \frac{1}{1 + \eta\Phi'^2(k)\|\mathbf{x}(k)\|_2^2}, \quad (8.48)$$

which does not exhibit the problem of unboundedness of η for small $\|\mathbf{x}(k)\|_2^2$, as experienced with the class of normalised algorithms. The ratio between \bar{e} and e as defined by (8.47) is shown in Figure 8.3. For the simulation, the parameters were $\eta = 0.1$ and $\|\mathbf{x}\| = 1$. The *a posteriori* error is clearly smaller than the *a priori* error for any Φ' of interest. For expansive functions, the ratio is smaller than for contractive functions, which means that data-reusing has the effect of stabilisation of the algorithm in this case.

⁶ For a linear Φ , using the matrix inversion lemma, the learning rate (8.48) is equivalent to the learning rate of a general NLMS algorithm $\mu_0(k)/\|\mathbf{x}(k)\|_2^2$ (Douglas and Rupp 1997).

8.6.1 A Posteriori Error Gradient Algorithm for Recurrent Neural Networks

Recall that when using the *a posteriori* information vector (8.41), the output of a recurrent perceptron becomes

$$\bar{y}(k) = \Phi\left(\sum_{j=1}^M w_j(k+1)x(k-j) + w_{M+1}(k+1) + \sum_{m=1}^N w_{m+M+1}(k+1)\bar{y}(k-m)\right), \quad (8.49)$$

or compactly as

$$\bar{y}(k) = \Phi(\bar{\mathbf{u}}^T(k)\mathbf{w}(k+1)). \quad (8.50)$$

Following the approach from Treichler (1987) and the above analysis, the *a posteriori* error gradient adaptation regarding the recurrent perceptron can be symbolically expressed as

$$\bar{\mathbf{\Pi}}(k+1) = \Phi'(\bar{\mathbf{u}}^T(k)\mathbf{w}(k+1))[\bar{\mathbf{u}}(k) + \mathbf{w}(k+1)\bar{\mathbf{\Pi}}(k)]. \quad (8.51)$$

Using the *a posteriori* error gradient technique from Ljung and Soderstrom (1983) and Treichler (1987), the weight update of this algorithm becomes

$$\eta\bar{e}(k)\bar{\mathbf{\Pi}}(k), \quad (8.52)$$

where $\bar{e}(k) = d(k) - \Phi(\bar{\mathbf{u}}^T(k)\mathbf{w}(k+1))$. Hence (Mandic and Chambers 1998c, 1999d),

$$\bar{e}(k) = \frac{d(k) - \Phi(\bar{\mathbf{u}}^T(k)\mathbf{w}(k))}{1 + \eta\|\bar{\mathbf{\Pi}}(k)\|_2^2}. \quad (8.53)$$

The denominator in (8.53) is strictly greater than unity, which makes the *a posteriori* error $\bar{e}(k)$ smaller in magnitude than the *a priori* error (Mandic and Chambers 1998a). The analysis for a full recurrent neural network is straightforward.

8.7 Experimental Results

The simulations on two speech signals, denoted by s1 and s2, which come from two different individuals, were undertaken in order to support the derived algorithms. The amplitudes of the signals were adjusted to lie within the range of the function Φ , i.e. within (0,1) for the logistic function. The measure that was used to assess the performance of the predictors was the forward prediction gain R_p given by

$$R_p \triangleq 10 \log_{10} \left(\frac{\hat{\sigma}_s^2}{\hat{\sigma}_e^2} \right) \text{ dB}, \quad (8.54)$$

where $\hat{\sigma}_s^2$ denotes the estimated variance of the speech signal $\{s(k)\}$, whereas $\hat{\sigma}_e^2$ denotes the estimated variance of the forward prediction error signal $\{e(k)\}$. In the experiments, the initialisation procedure used the same strategy, namely epochwise, with 200 epochs ran over 300 samples, as described in Mandic *et al.* (1998) and Baltersee and Chambers (1998). The network chosen for the analysis was with $N = 2$ neurons and one external input signal to the network. Such a network was tested on both the *a priori* and *a posteriori* algorithms and the simulation results are shown

Table 8.1 Comparison between prediction gains R_p between an *a priori* scheme and various *a posteriori* schemes

Prediction gain	s1	s2
R_p (dB) <i>a priori</i> scheme $\mathbf{W}(k)$, feedback nonupdated, $N = 2$	7.78	8.48
R_p (dB) <i>a posteriori</i> scheme $\mathbf{W}(k+1)$, feedback nonupdated, $N = 2$	7.84	8.55
R_p (dB) <i>a posteriori</i> scheme $\mathbf{W}(k+1)$, feedback updated, <i>a posteriori</i> $\bar{\mathbf{I}}$, $N = 2$	9.23	8.78
R_p (dB) <i>a priori</i> scheme $\mathbf{W}(k)$, feedback nonupdated, $N = 3$	7.79	8.49

in Table 8.1. In order to show the merit of the *a posteriori* approach on the network with $N = 2$ neurons, the results achieved were further compared to the performance obtained using the *a priori* network with $N = 3$ neurons. The results show that the performance of an RNN predictor improves with the amount of *a posteriori* information involved. The best results were achieved for the scheme with the *a posteriori* gradients, which were used in the recursive weights adaptation algorithm, and the corresponding feedback values were updated, i.e. the *a posteriori* information vector was used. The improvement in the performance in that case, as compared with the *a priori* scheme with $N = 3$, was, for example, 1.44 dB for the signal s1.

In the second experiment, an *a posteriori* algorithm with an updated *a posteriori* information vector was applied to modular nested recurrent architectures (PRNN) as described in Mandic *et al.* (1998) and Baltersee and Chambers (1998). The experiments were undertaken on speech. For modular networks with typically $M = 5$ modules and $N = 2$ neurons per recurrent module, there was not much improvement in prediction gain between the *a posteriori* gradient formulation and the standard algorithm. However, for an ERLS learning algorithm (Appendix D), there was an improvement of several percent in the prediction gain when using the algorithm with an updated *a posteriori* information vector (8.41).

8.8 Summary

Relationships between the *a priori* and *a posteriori* prediction error, learning rate and slope of the nonlinear activation function of a nonlinear adaptive filter realised by a neural network have been derived. This has been undertaken for learning algorithms based upon gradient descent for both the feedforward and recurrent case. A general data-reusing (*a posteriori* adaptation) adaptive algorithm for nonlinear adaptive filters realised as neural networks was then analysed. These algorithms use L weight updates per fixed external input vector and teaching signal. Therefore, their performance is in between the standard gradient descent and normalised algorithm. Relationships between the errors from consecutive iterations

of the algorithm are derived based upon the corresponding *a priori* prediction error and gradients of the nonlinear activation function of a neuron, as well as the \mathcal{L}_2 norm of the input data vector and the learning rate η . However, in practice, the benefits of data-reusing techniques may not be significant. Relationships between η , β and $\|\mathbf{x}\|_2^2$ deserve more attention and will be addressed in Chapter 12.

9

A Class of Normalised Algorithms for Online Training of Recurrent Neural Networks

9.1 Perspective

A normalised version of the real-time recurrent learning (RTRL) algorithm is introduced. This has been achieved via local linearisation of the RTRL around the current point in the state space of the network. Such an algorithm provides an adaptive learning rate normalised by the \mathcal{L}_2 norm of the gradient vector at the output neuron. The analysis is general and also covers simpler cases of feedforward networks and linear FIR filters.

9.2 Introduction

Gradient-descent-based algorithms for training neural networks, such as the backpropagation, backpropagation through time, recurrent backpropagation (RBP) and real-time recurrent learning (RTRL) algorithm, typically suffer from slow convergence when dealing with statistically nonstationary inputs. In the area of linear adaptive filters, similar problems with the LMS algorithm have been addressed by utilising normalised algorithms, such as NLMS. We therefore introduce a normalised RTRL-based learning algorithm with the idea to impose similar stabilisation and convergence effects on training of RNNs, as normalisation imposes on the LMS algorithm.

In the area of linear FIR adaptive filters, it is shown (Soria-Olivas *et al.* 1998) that a normalised gradient-descent-based learning algorithm can be derived starting from the Taylor series expansion of the instantaneous output error of an adaptive FIR filter, given by

$$e(k+1) = e(k) + \sum_{i=1}^N \frac{\partial e(k)}{\partial w_i(k)} \Delta w_i(k) + \frac{1}{2!} \sum_{i=1}^N \sum_{j=1}^N \frac{\partial^2 e(k)}{\partial w_i(k) \partial w_j(k)} \Delta w_i(k) \Delta w_j(k) + \dots \quad (9.1)$$

From the mathematical description of LMS¹ from Chapter 2, we have

$$\frac{\partial e(k)}{\partial w_i(k)} = -x(k-i+1), \quad i = 1, 2, \dots, N, \quad (9.2)$$

and

$$\Delta w_i(k) = \mu(k)e(k)x(k-i+1), \quad i = 1, 2, \dots, N. \quad (9.3)$$

Due to the linearity of the FIR filter, the second- and higher-order partial derivatives in (9.1) vanish.

Combining (9.1)–(9.3) yields

$$e(k+1) = e(k) - \mu(k)e(k)\|\mathbf{x}(k)\|_2^2 \quad (9.4)$$

for which the nontrivial solution gives the learning rate of a normalised LMS algorithm

$$\mu_{\text{NLMS}}(k) = \frac{1}{\|\mathbf{x}(k)\|_2^2}. \quad (9.5)$$

The stability analysis of adaptive algorithms can be undertaken using contractive operators and fixed point iteration. For the contractive operator \mathbf{T} , it follows that

$$\|\mathbf{T}\mathbf{z}_1 - \mathbf{T}\mathbf{z}_2\| \leq \gamma\|\mathbf{z}_1 - \mathbf{z}_2\|, \quad 0 \leq \gamma < 1, \quad \mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^N. \quad (9.6)$$

The convergence analysis of LMS, for instance, can be undertaken starting from the misalignment² vector $\mathbf{v}(k) = \mathbf{w}(k) - \tilde{\mathbf{w}}(k)$ by setting $\mathbf{z}_1 = \mathbf{v}(k+1)$, $\mathbf{z}_2 = \mathbf{v}(0)$ and $\mathbf{T} = [\mathbf{I} - \mu(k)\mathbf{x}(k)\mathbf{x}^T(k)]$ (Gholkar 1990). Detailed convergence analysis for a class of gradient-based learning algorithms for recurrent neural networks is given in Chapter 10.

9.3 Overview

A class of normalised gradient-based algorithms is derived starting from the LMS algorithm for linear adaptive filters through to a normalised algorithm for training recurrent neural networks. For each case the adaptive learning rate has been derived. Stability of such algorithms is addressed in Chapter 10. The normalised algorithms are shown to outperform standard algorithms with fixed learning rate.

¹ The two core equations for adaptation of the LMS algorithm are

$$\begin{aligned} e(k) &= d(k) - \mathbf{x}^T(k)\mathbf{w}(k), \\ \mathbf{w}(k+1) &= \mathbf{w}(k) + \mu(k)e(k)\mathbf{x}(k). \end{aligned}$$

² The misalignment vector is defined as $\mathbf{v}(k) = \mathbf{w}(k) - \tilde{\mathbf{w}}(k)$, where $\tilde{\mathbf{w}}(k)$ is the set of optimal weights of the system.

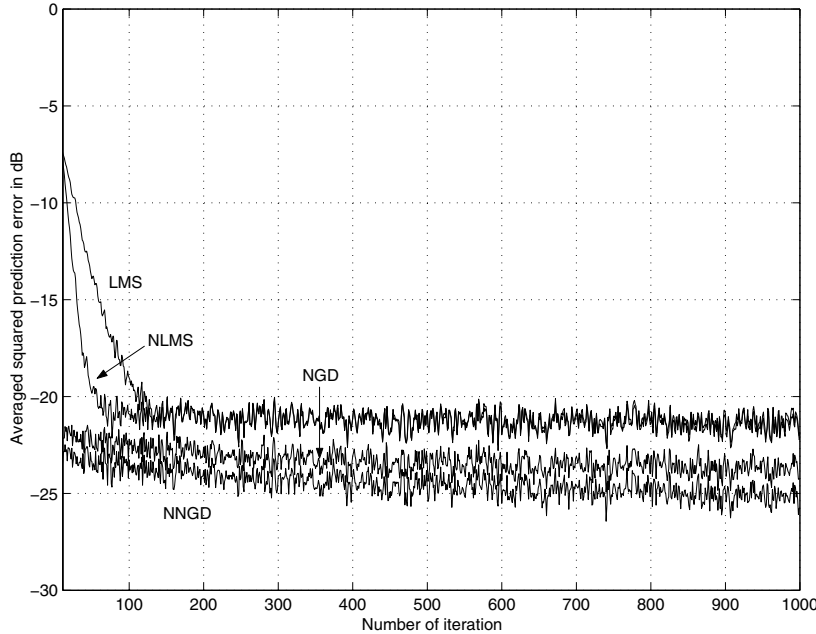


Figure 9.1 Comparison of convergence of the averaged squared prediction error with the LMS, NLMS, NGD and NNGD algorithms, with logistic activation function, for a coloured input

9.4 Derivation of the Normalised Adaptive Learning Rate for a Simple Feedforward Nonlinear Filter

The equations that define the adaptation for a neural adaptive filter with one neuron (Figure 2.6), trained by a nonlinear gradient descent (NGD) algorithm, are

$$e(k) = d(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)), \tag{9.7}$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}(k))e(k)\mathbf{x}(k), \tag{9.8}$$

where $e(k)$ is the instantaneous error at the output neuron, $d(k)$ is some training (desired) signal, $\mathbf{x}(k) = [x_1(k), \dots, x_N(k)]^T$ is the input vector, $\mathbf{w}(k) = [w_1(k), \dots, w_N(k)]^T$ is the weight vector, $\Phi(\cdot)$ is a nonlinear activation function of a neuron and $(\cdot)^T$ denotes the vector transpose. The learning rate η is supposed to be a small positive real number. Following the approach from Mandic (2000a), if the output error (9.7) is expanded using a Taylor series expansion, we have

$$e(k+1) = e(k) + \sum_{i=1}^N \frac{\partial e(k)}{\partial w_i(k)} \Delta w_i(k) + \frac{1}{2!} \sum_{i=1}^N \sum_{j=1}^N \frac{\partial^2 e(k)}{\partial w_i(k) \partial w_j(k)} \Delta w_i(k) \Delta w_j(k) + \dots \tag{9.9}$$

From (9.7) and (9.8), the elements of (9.9) are

$$\frac{\partial e(k)}{\partial w_i(k)} = -\Phi'(\mathbf{x}^T(k)\mathbf{w}(k))x_i(k), \quad i = 1, 2, \dots, N, \tag{9.10}$$

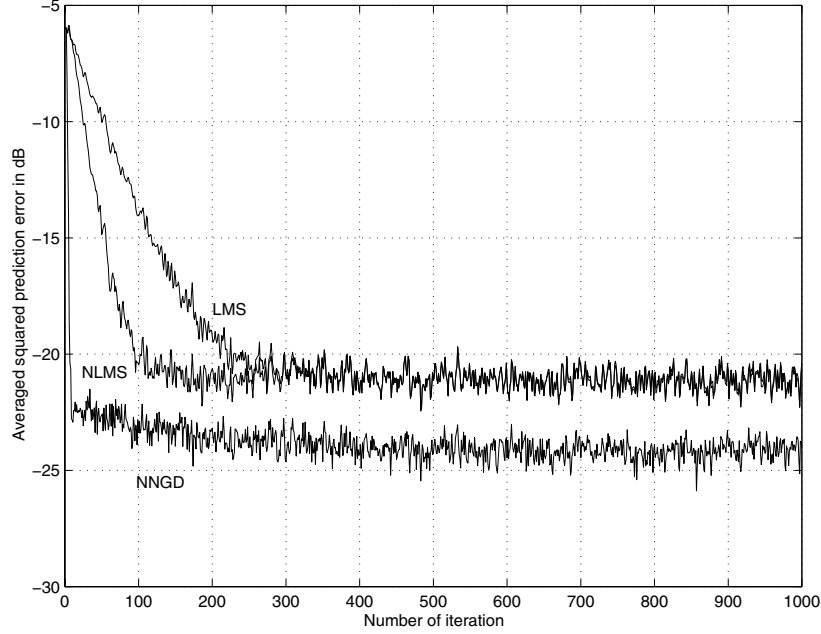


Figure 9.2 Comparison of convergence of the averaged squared prediction error of the LMS, NLMS and NNGD algorithms for a coloured input and tanh activation function with $\beta = 1$

and

$$\Delta w_i(k) = w_i(k+1) - w_i(k) = \eta(k)\Phi'(\mathbf{x}^T(k)\mathbf{w}(k))e(k)x_i(k), \quad i = 1, 2, \dots, N. \quad (9.11)$$

The second partial derivatives are

$$\frac{\partial^2 e(k)}{\partial w_i(k)\partial w_j(k)} = -\Phi''(\mathbf{x}^T(k)\mathbf{w}(k))x_i(k)x_j(k), \quad i, j = 1, 2, \dots, N. \quad (9.12)$$

Let us denote $\text{net}(k) = \mathbf{x}^T(k)\mathbf{w}(k)$. Combining (9.9)–(9.12) yields

$$\begin{aligned} e(k+1) &= e(k) - \eta(k)[\Phi'(\text{net}(k))]^2 e(k) \sum_{i=1}^N x_i^2(k) \\ &\quad - \frac{1}{2!}\eta^2(k)e^2(k)[\Phi'(\text{net}(k))]^2\Phi''(\text{net}(k)) \sum_{i=1}^N \sum_{j=1}^N x_i^2(k)x_j^2(k) + \dots \end{aligned} \quad (9.13)$$

A truncated Taylor series expansion of (9.13) gives

$$e(k+1) = e(k)[1 - \eta(k)[\Phi'(\text{net}(k))]^2\|\mathbf{x}(k)\|_2^2]. \quad (9.14)$$

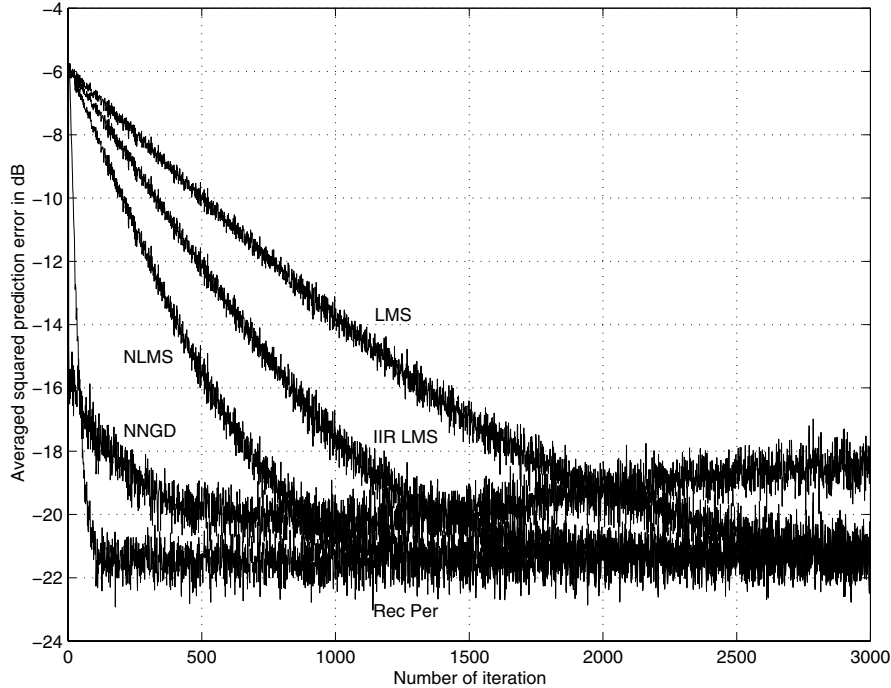


Figure 9.3 Convergence comparison of averaged squared prediction error for feedforward and recurrent structures, tanh activation function with $\beta = 4$ and coloured input

The aim is for the error $e(k+1)$ in (9.14) to vanish, which is the case for the nontrivial solution

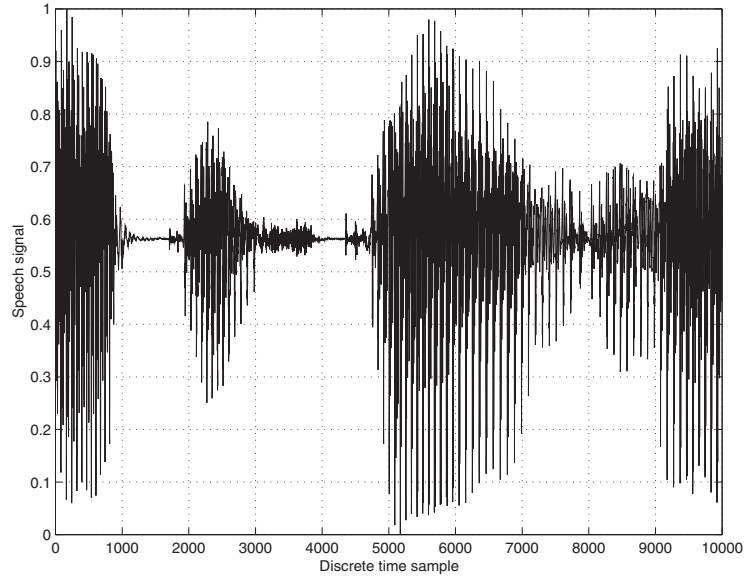
$$\eta_{\text{OPT}}(k) = \frac{1}{[\Phi'(\text{net}(k))]^2 \|\mathbf{x}(k)\|_2^2}, \tag{9.15}$$

which is the step size of a normalised gradient descent (NNGD) algorithm for a non-linear FIR filter. Taking into account the bounds³ on the values of higher derivatives of Φ , for a contractive activation function we may adjust the derived learning rate with a positive constant C , as

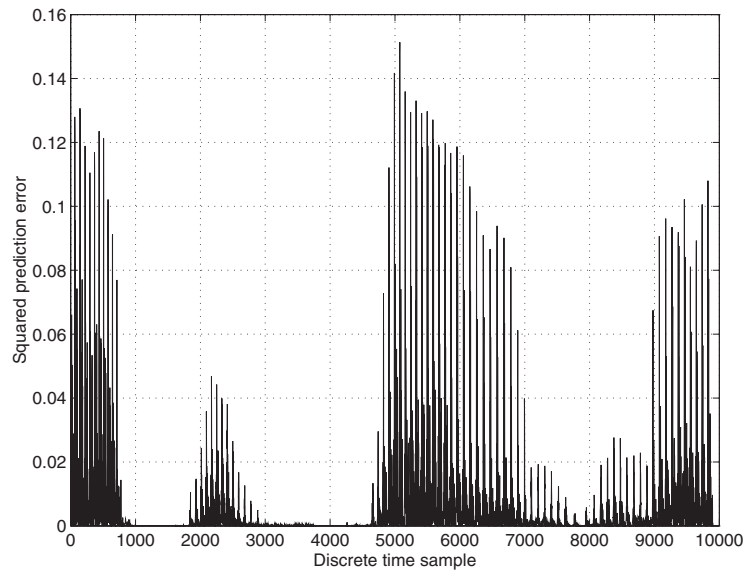
$$\eta_{\text{OPT}}(k) = \frac{1}{C + [\Phi'(\text{net}(k))]^2 \|\mathbf{x}(k)\|_2^2}. \tag{9.16}$$

The magnitude of the learning rate varies in time with the tap input power and the first derivative of the activation function, which provides a normalisation of the algorithm. Further discussion on the size and role of constant C in (9.16) can be found in Mandic and Krmar (2001) and Krmar and Mandic (2001). The adaptive learning rate from (9.15) degenerates into the learning rate of the NLMS algorithm for a linear activation function. A normalised backpropagation algorithm for a general feedforward neural network is given in Mandic and Chambers (2000f). Although the

³ For the logistic function, for instance, the second-order term in the Taylor series expansion is positive.

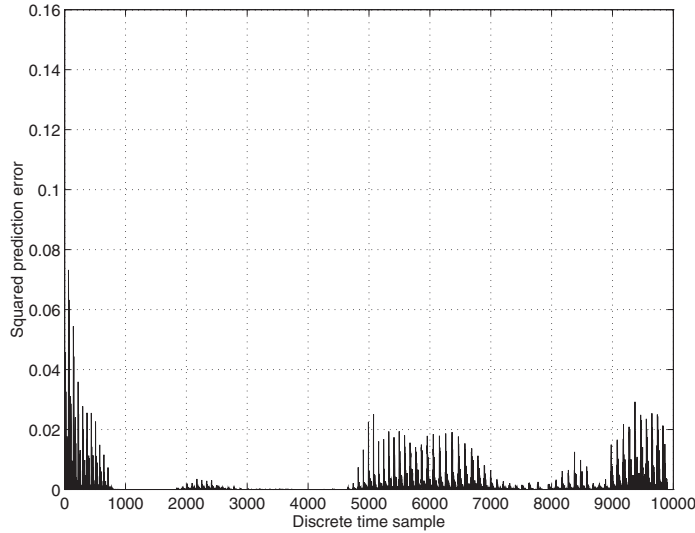


(a) The input speech signal



(b) Standard RTRL algorithm

Figure 9.4 Squared instantaneous prediction error for the RTRL and NRTRL algorithms with speech inputs



(c) Normalised RTRL algorithm

Figure 9.4 *Cont.*

derivation of the normalised algorithm is simple, it assumes statistical independence between the weights, input vector, teaching signal and learning rate, which is often not the case in practical applications. Therefore, the optimal learning rate for practical applications should be chosen to be smaller than the one derived above. This is one of the reasons why there is a need to add a positive constant C to the denominator of (9.15).

In Mandic (2000a), a simulation was undertaken on speech, a nonlinear and nonstationary signal, for a nonlinear FIR filter with tap length $N = 10$, with $\eta = 0.3$, $C = 1$ and $\beta = 4$. The quantitative performance measure was the standard prediction gain, a logarithmic ratio between the expected signal and error variances $R_p = 10 \log(\hat{\sigma}_s^2/\hat{\sigma}_e^2)$. For this setting, the prediction gain for the LMS was 7.24 dB, 8.26 dB for the NLMS, 7.67 dB for a nonlinear GD and 9.28 dB for the NNGD algorithm, confirming the analysis from the previous section.

We next compare the performances of FIR filters trained by LMS and NLMS, IIR filters trained by LMS, nonlinear FIR filters trained by NGD and NNGD and a NARMA recurrent perceptron trained by the RTRL. The order of FIR filters was $N = 10$. The input was a white noise sequence passed through an AR channel given by

$$y(k) = 1.79y(k - 1) - 1.85y(k - 2) + 1.27y(k - 3) - 0.41y(k - 4) + \nu(k), \quad (9.17)$$

where $\nu(k)$ denotes the white input noise. The resulting input signal was rescaled so as to fit within the range of the logistic and tanh activation function. A Monte Carlo simulation with 200 trials was undertaken for all the experiments.

Figure 9.1 shows a comparison between convergence curves for the LMS, NLMS,⁴ NGD (a standard nonlinear gradient descent) and NNGD algorithms for a coloured input from AR channel (9.17). The slope of the logistic function was $\beta = 4$, which partly coincides with the linear curve $y = x$. The NNGD algorithm for a feedforward dynamical neuron clearly outperforms the other employed algorithms. The NGD algorithm also outperformed the LMS and NLMS algorithms. Figure 9.2 shows the convergence curves for a tanh activation function and the input from the same AR channel. The NNGD algorithm has consistently improved convergence performance over the LMS and NLMS algorithms.

Convergence curves for LMS, NLMS, NNGD, IIR LMS and a NARMA(6,1) recurrent perceptron for a correlated input (AR channel) and tanh activation function with $\beta = 4$ are shown in Figure 9.3. A NARMA recurrent perceptron outperformed all the other algorithms in simulations. This does not mean, however, that recurrent structures perform best in all practical applications.

9.5 A Normalised Algorithm for Online Adaptation of Recurrent Neural Networks

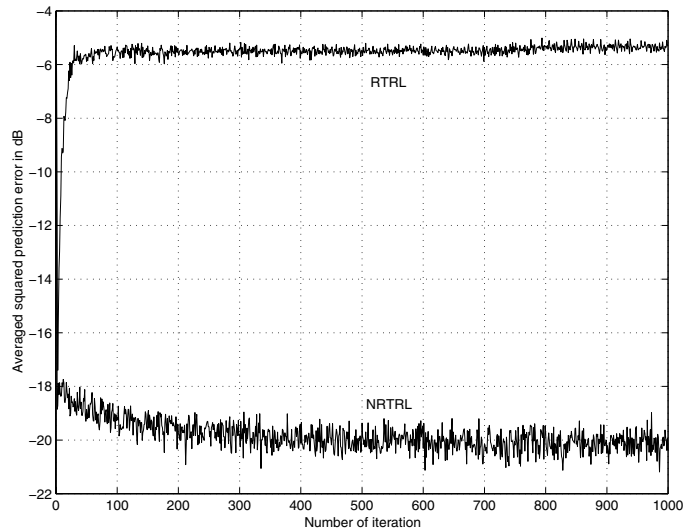
An output error of a fully connected recurrent neural network can be expanded via a Taylor series expansion as (Mandic and Chambers 2000b)

$$\begin{aligned}
 e(k+1) &= e(k) + \sum_{i=1}^N \sum_{j=1}^{M+N+1} \frac{\partial e(k)}{\partial w_{i,j}(k)} \Delta w_{i,j}(k) \\
 &+ \frac{1}{2!} \sum_{i=1}^N \sum_{m=1}^{M+N+1} \sum_{j=1}^N \sum_{n=1}^{M+N+1} \frac{\partial^2 e(k)}{\partial w_{i,m}(k) \partial w_{j,n}(k)} \Delta w_{i,m}(k) \Delta w_{j,n}(k) + \dots,
 \end{aligned}
 \tag{9.18}$$

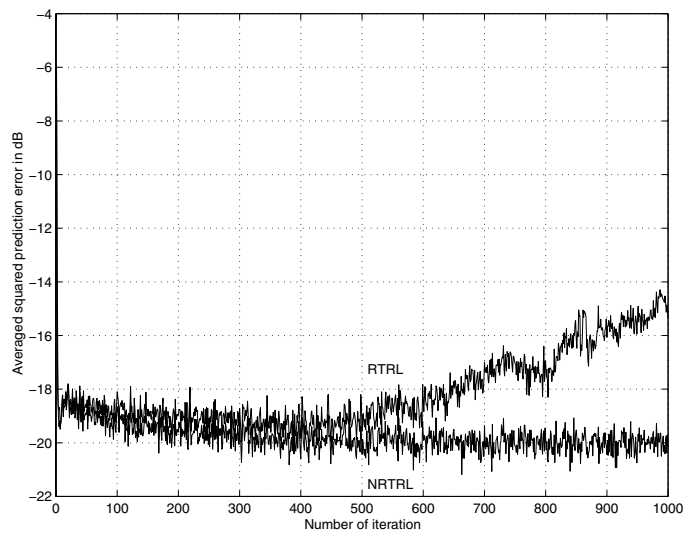
where M is the order of the input signal tap delay line and N is the number of neurons. This is a complicated expression and only the first two terms of (9.18) will be considered. Due to the internal feedback in RNNs, the partial derivatives $\partial e(k)/\partial w_{i,j}(k)$ are not straightforward to calculate (Appendix D). From (9.18), using an approach similar to the one explained for a simple feedforward neural filter and neglecting the higher-order terms in the Taylor series expansion gives

$$\begin{aligned}
 e(k+1) &= e(k) - \eta(k)e(k) \sum_{i=1}^N \sum_{j=1}^{M+N+1} \left[\frac{\partial y_1(k)}{\partial w_{i,j}(k)} \right]^2 \\
 &= e(k) - \eta(k)e(k) \sum_{i=1}^N \|\mathbf{\Pi}_1^{(i)}(k)\|_2^2,
 \end{aligned}
 \tag{9.19}$$

⁴ For numerical stability, the learning rate for NLMS was chosen as $\mu(k) = \mu_0/(\epsilon + \|\mathbf{x}\|_2^2)$, where $\mu_0 < 1$ is a positive constant and ϵ is some small positive constant that prevents divergence for small $\|\mathbf{x}\|_2$. This explains the better performance of NNGD over NLMS for an input coming from a linear AR channel.

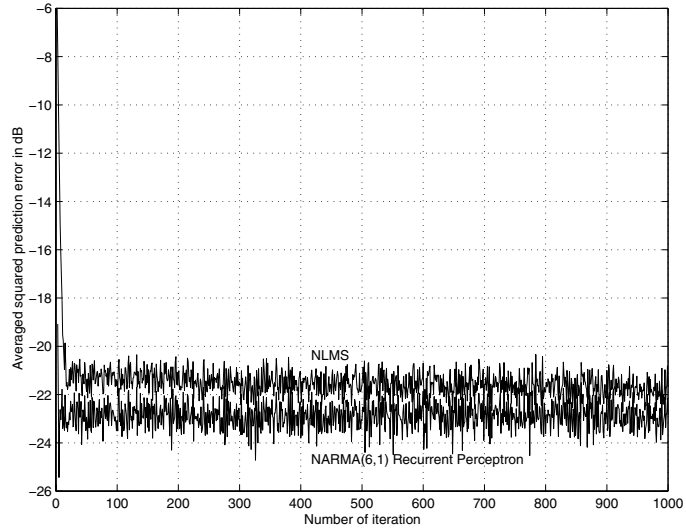


(a) Convergence comparison between RTRL and NRTRL

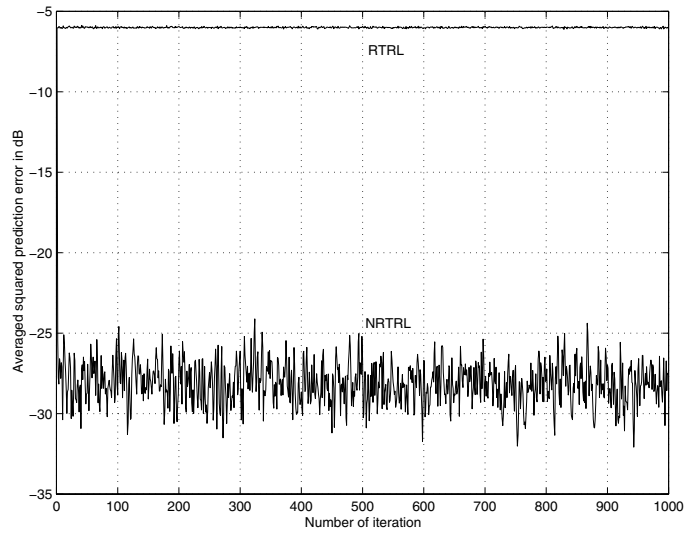


(b) Convergence comparison between RTRL and NRTRL when RTRL fails

Figure 9.5 Convergence comparison of averaged squared prediction error for a RTRL and NRTRL trained recurrent structure, tanh activation function with $\beta = 2$ and coloured input



(a) Convergence curves for NLMS for $N = 10$ and RTRL for a NARMA(4,1) recurrent perceptron for a nonlinear input (9.22), *logistic* activation function with $\beta = 4$



(b) Convergence curves for RTRL and NRTRL, for a NARMA(10,2) recurrent perceptron, *tanh* activation function with $\beta = 8$ for a nonlinear input (9.23)

Figure 9.6 Convergence of RTRL and NRTRL for nonlinear inputs

where $\mathbf{II}_1^{(i)}$ denotes the gradients at the output neuron y_1 with respect to the weights from the i th neuron. Hence, the optimal value of learning rate $\eta_{\text{OPT}}(k)$ for an RTRL trained RNN is

$$\eta_{\text{OPT}}(k) = \frac{1}{\sum_{i=1}^N \|\mathbf{II}_1^{(i)}(k)\|_2^2}. \tag{9.20}$$

The normalisation factor is the tap input power to an RNN multiplied by the derivative of the nonlinear activation function and augmented by the product of gradients and feedback weights. Hence, we will refer to the result from (9.20) as the normalised real-time recurrent learning (NRTRL) algorithm. For a normalised algorithm for a recurrent perceptron, we have

$$\eta_{\text{OPT}}(k) = \frac{1}{\|\mathbf{II}(k)\|_2^2}. \tag{9.21}$$

Due to the derivation of η_{OPT} from a truncated Taylor series expansion, a positive constant C should be added to the term in the denominator of (9.20) and (9.21).

Figure 9.4 shows the comparison of instantaneous squared prediction errors between the RTRL and NRTRL for a nonstationary (speech) signal. The NRTRL algorithm from Figure 9.4(c), clearly achieves significantly better performance than the RTRL algorithm (Figure 9.4(b)). To quantify this, if the measure of performance is the standard prediction gain, the NRTRL achieved approximately 7 dB better performance than the RTRL algorithm. Convergence comparison between the RTRL and NRTRL algorithms for the cases where both algorithms converge (Figure 9.5(a)) and when RTRL diverges (Figure 9.5(b)) is shown in Figure 9.5. A small constant was added to the denominator of the optimal learning rate η_{OPT} . The input was a coloured signal from an AR channel and the slope of the tanh activation function was $\beta = 2$ (notice that the contractivity might have been violated). In both cases depicted in Figure 9.5, the NRTRL comprehensively outperformed the RTRL algorithm. In Figure 9.6, a comparison between convergence curves for benchmark nonlinear inputs defined as (Narendra and Parthasarathy 1990)

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)x(k-1)[y(k-2)-1] + x(k)}{1 + y^2(k-1) + y^2(k-2)}, \tag{9.22}$$

$$y(k+1) = \frac{y(k)}{1 + y^2(k)} + x^3(k), \tag{9.23}$$

is given. In Figure 9.6(a), a NARMA(4,1) recurrent perceptron trained by RTRL outperformed a FIR filter with $N = 10$ trained by NLMS for input (9.22).

In Figure 9.6(b), comparison between convergence curves for RTRL and NRTRL on a benchmark nonlinear input (9.23) is given. The employed tanh activation function was expansive with $\beta = 8$ and the simulations were undertaken for a NARMA(10,2) recurrent perceptron. The NRTRL outperformed RTRL for this case.

Simulations show that the performance of the NRTRL is highly dependent on the choice of the constant C in the denominator of the optimal learning rate. Dependent on the choice of C , the NRTRL can have worse, similar or better performance than RTRL. However, in most practical cases, $C < 1$ is a sufficiently good range for the NRTRL to outperform the RTRL. To further depict the dependence of performance on the

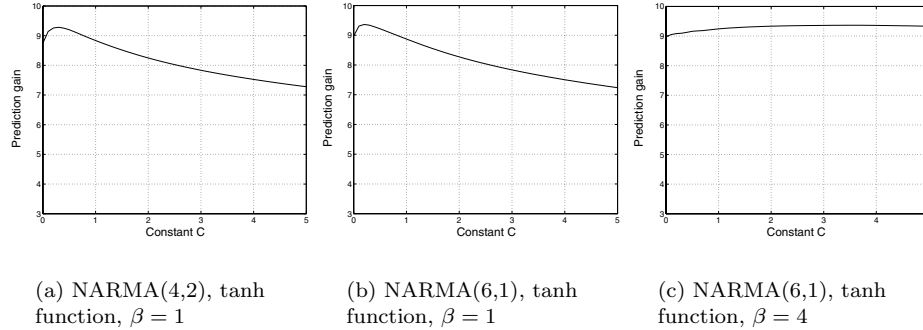


Figure 9.7 Prediction gain versus the Taylor series remainder C for a speech signal and NARMA recurrent perceptrons

value of C , three experiments were undertaken on a real speech signal. The prediction gain was calculated for various values of parameter C . The filter used was a NARMA recurrent perceptron. In Figure 9.7(a), prediction gain for a NARMA(4,2) perceptron with a tanh activation function with $\beta = 1$ had its maximum for $C = 0.3$. The experiment was repeated for a NARMA(6,1) recurrent perceptron, and the maximum of the prediction gain was obtained for $C = 0.22$, which is shown in Figure 9.7(b). Finally, for the same network, an expansive tanh activation function was used, with $\beta = 4$. As expected, in this case, the best performance was achieved for $C > 1$, which is shown in Figure 9.7(c).

9.6 Summary

An optimal adaptive learning rate has been derived for the RTRL algorithm for continually running fully connected recurrent neural networks. The learning rate is optimal in the sense that it minimises the instantaneous squared prediction error at the output neuron for every time instant while the network is running. This algorithm normalises the learning rate of the RTRL and is hence referred to as the normalised RTRL (NRTRL) algorithm. The NRTRL is stabilised by the \mathcal{L}_2 norm of the input data vector and local gradients at the output neuron of the network. The additional computational complexity involved is not significant, when compared to the entire computational complexity of the RTRL algorithm. Simulations show that normalised algorithms outperform the standard algorithms in both the feedforward and recurrent case.

10

Convergence of Online Learning Algorithms in Neural Networks

10.1 Perspective

An analysis of convergence of real-time algorithms for online learning in recurrent neural networks is presented. For convenience, the analysis is focused on the real-time recurrent learning (RTRL) algorithm for a recurrent perceptron. Using the assumption of contractivity of the activation function of a neuron and relaxing the rigid assumptions of the fixed optimal weights of the system, the analysis presented is general and is applicable to a wide range of existing algorithms. It is shown that some of the results obtained for stochastic gradient algorithms for linear systems can be considered as a bound for stability of RNN-based algorithms, as long as the contractivity condition holds.

10.2 Introduction

The following criteria (Bershad *et al.* 1990) are most commonly used to assess the performance of adaptive algorithms.

1. Convergence (consistency of the statistics).
2. Transient behaviour (how quickly the algorithm reacts to changes in the statistics of the input).
3. Convergence rate (how quickly the algorithm approaches the optimal solution), which can be linear, quadratic or superlinear.

The standard approach for the analysis of convergence of learning algorithms for linear adaptive filters is to look at convergence of the mean weight error vector, convergence in the mean square and at the steady-state misadjustment (Gholkar 1990; Haykin 1996a; Kuan and Hornik 1991; Widrow and Stearns 1985). The analysis of convergence of steepest-descent-based algorithms has been ongoing ever since their

introduction (Guo and Ljung 1995; Ljung 1984; Slock 1993; Tarrab and Feuer 1988). Some of the recent results consider the exact expectation analysis of the LMS algorithm for linear adaptive filters (Douglas and Pan 1995) and the analysis of LMS with Gaussian inputs (Bershad 1986). For neural networks as nonlinear adaptive filters, the analysis is far more difficult, and researchers have often resorted to numerical experiments (Ahmad *et al.* 1990). Convergence of neural networks has been considered in Shynk and Roy (1990), Bershad *et al.* (1993a) and Bershad *et al.* (1993b), where the authors used the Gaussian model for input data and a Rosenblatt perceptron learning algorithm. These analyses, however, were undertaken for a hard limiter nonlinearity, which is not convenient for nonlinear adaptive filters. Convergence of RTRL was addressed in Mandic and Chambers (2000b) and Chambers *et al.* (2000).

An error equation for online training of a recurrent perceptron can be expressed as

$$e(k) = s(k) - \Phi(\mathbf{u}^T(k)\mathbf{w}(k)), \quad (10.1)$$

where $s(k)$ is the teaching (desired) signal, $\mathbf{w}(k) = [w_1(k), \dots, w_N(k)]^T$ is the weight vector and $\mathbf{u}(k) = [u_1(k), \dots, u_N(k)]^T$ is an input vector. A weight update equation for a general class of stochastic gradient-based nonlinear neural algorithms can be expressed as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)F(\mathbf{u}(k))g(\mathbf{u}(k), \mathbf{w}(k)), \quad (10.2)$$

where $\eta(k)$ is the learning rate, $F: \mathbb{R}^N \rightarrow \mathbb{R}^N$ usually consists of N copies of the scalar function f and $g(\cdot)$ is a scalar function related to the error $e(k)$. The function F is related to data nonlinearities, which have an influence on the convergence of the algorithm. The function g is related to error nonlinearities, and it affects the cost function to be minimised. Error nonlinearities are mostly chosen to be sign-preserving (Sethares 1992).

Let us assume additive noise $q(k) \sim \mathcal{N}(0, \sigma_q^2)$ in the output of the system, which can be expressed as

$$s(k) = \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k)) + q(k), \quad (10.3)$$

where $\tilde{\mathbf{w}}(k)$ are optimal filter weights and $q(k)$ is an i.i.d. sequence. The error equation (10.1) now becomes

$$e(k) = \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k)) - \Phi(\mathbf{u}^T(k)\mathbf{w}(k)) + q(k). \quad (10.4)$$

To examine the stability of algorithm (10.2), researchers often resort to linearisation. For RTRL, F is an identity matrix and g is some nonlinear, sign-preserving function of the output error. A further assumption is that the learning rate η is sufficiently small to allow the algorithm to be linearised around its current point in the state space. From Lyapunov stability theory, the system

$$\mathbf{z}(k+1) = F(k, \mathbf{z}(k)) \quad (10.5)$$

can be analysed via its linearised version

$$\mathbf{z}(k+1) = \mathbf{A}(k)\mathbf{z}(k), \quad (10.6)$$

where \mathbf{A} is the Jacobian of F . This is the Lyapunov indirect method and assumes that $\mathbf{A}(k)$ is bounded in the neighbourhood of the current point in the state space

and that

$$\lim_{\|\mathbf{z}\| \rightarrow 0} \max_k \frac{\|F(k, \mathbf{z}) - \mathbf{A}(k)\mathbf{z}\|}{\|\mathbf{z}\|} = 0, \quad (10.7)$$

which guarantees that time variation in the nonlinear terms of the Taylor series expansion of (10.5) does not become arbitrarily large in time (Chambers *et al.* 2000). Results on Lyapunov stability for a class of nonlinear systems can be found in Wang and Michel (1994) and Tanaka (1996).

Averaging methods for the analysis of stability and convergence of adaptive algorithms, for instance, use a linearised version of the system matrix of (10.2)

$$\mathbf{v}(k) = [\mathbf{I} - \eta \mathbf{u}(k) \mathbf{u}^T(k)] \tilde{\mathbf{w}}(k), \quad (10.8)$$

which is then replaced by the ensemble average (Anderson *et al.* 1986; Kushner 1984; Solo and Kong 1994)

$$E[\mathbf{I} - \eta \mathbf{u}(k) \mathbf{u}^T(k)] = \mathbf{I} - \eta \mathbf{R}_{\mathbf{u}, \mathbf{u}}, \quad (10.9)$$

where $\mathbf{v}(k)$ is the misalignment vector which will be defined later and $\mathbf{R}_{\mathbf{u}, \mathbf{u}}$ is the autocorrelation matrix of the tap-input vector $\mathbf{u}(k)$.

It is also often assumed that the filter coefficients are statistically independent of the input data currently in the filter memory, which is convenient, but essentially incorrect. This assumption is one of the *independence assumptions*, which are (Haykin 1996a)

1. the sequence of tap input vectors are statistically independent;
2. the tap input vector is statistically independent of all the previous samples of the desired response;
3. the desired response is statistically independent of all the previous samples of the desired response; and
4. the tap input vector and the desired response consist of *mutually Gaussian-distributed* random variables.

The weight error vector hence depends on the previous sample input vectors, the previous samples of the desired response and the initial value of the tap weight vector. Convergence analysis of stochastic gradient algorithms is still ongoing, mainly to relax the independence assumptions (Douglas and Pan 1995; Guo *et al.* 1997; Solo and Kong 1994).

The following are the most frequently used convergence criteria in the analysis of adaptive algorithms:

1. convergence of the weight fluctuation in the mean $\|E[\mathbf{v}(k)]\| \rightarrow 0$, as $k \rightarrow \infty$, where $\mathbf{v}(k) = \mathbf{w}(k) - \tilde{\mathbf{w}}(k)$;
2. mean squared error convergence calculated from $E[\mathbf{v}(k) \mathbf{v}^T(k)]$; and
3. steady-state mean squared error, which is obtained from mean squared error convergence (misadjustment).

To allow for time-varying input signal statistics, in the following analysis we use a fairly general condition that the optimal filter weights $\tilde{\mathbf{w}}(k)$ are governed by the modified first-order Markov model as (Bershad *et al.* 1990),

$$\tilde{\mathbf{w}}(k+1) = \lambda \tilde{\mathbf{w}}(k) + \sqrt{1 - \lambda^2} \mathbf{n}(k), \quad (10.10)$$

where $\lambda \in [0, 1]$ is the parameter which defines the time variation of $\tilde{\mathbf{w}}(k)$ and $\mathbf{n}(k)$ is an i.i.d. Gaussian noise vector. A zero-mean initialisation of model (10.10) is assumed ($E[\tilde{\mathbf{w}}(k)] = \mathbf{0}$). This model covers most of the learning algorithms employed, be they linear or nonlinear. For instance, the momentum algorithm models the weight update as an AR process. In addition, learning algorithms based upon the Kalman filter model weight fluctuations as a white noise sequence (random walk), which is in fact a first-order Markov process (Appendix D). The standard case of a single optimal solution to the stochastic gradient optimisation process (non time-varying) can be obtained by setting $\lambda = 1$.

10.3 Overview

Based upon the stability results introduced in Chapter 7, the analysis of convergence for stochastic gradient algorithms for nonlinear adaptive filters is provided. The analysis is mathematically strict and covers most of the previously introduced algorithms. This approach can be extended to more complicated architectures and learning algorithms.

10.4 Convergence Analysis of Online Gradient Descent Algorithms for Recurrent Neural Adaptive Filters

The problem of optimal nonlinear gradient-descent-based training can be presented in a similar fashion to the linear case (Douglas 1994), as

$$\text{minimise} \quad \|\mathbf{w}(k+1) - \mathbf{w}(k)\| \quad (10.11)$$

$$\text{subject to} \quad s(k) - \Phi(\mathbf{u}^T(k)\mathbf{w}(k+1)) = 0, \quad (10.12)$$

where $\|\cdot\|$ denotes some norm (most commonly the 2-norm). The equation that defines the adaptation of a recurrent neural network is

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta(k) \nabla_{\mathbf{w}(k)} E(k), \quad (10.13)$$

where $E(k) = \frac{1}{2}e^2(k)$ is the cost function to be minimised. The correction to the weight vector for a recurrent perceptron at time instant k becomes (Williams and Zipser 1989a)

$$\Delta \mathbf{w}(k) = \eta(k) e(k) \mathbf{\Pi}(k), \quad (10.14)$$

where

$$\mathbf{\Pi}(k) = \left[\frac{\partial y(k)}{\partial w_1(k)}, \dots, \frac{\partial y(k)}{\partial w_N(k)} \right]^T$$

represents the gradient vector at the output of the neuron. Consider the weight update equation for a general RTRL trained RNN

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)e(k)\mathbf{II}(k). \quad (10.15)$$

Following the approach from Chambers *et al.* (2000) and using (10.4) and (10.15), we have

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)q(k)\mathbf{II}(k) + \eta(k)\Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k))\mathbf{II}(k) - \eta(k)\Phi(\mathbf{u}^T(k)\mathbf{w}(k))\mathbf{II}(k). \quad (10.16)$$

The misalignment vector \mathbf{v} can be expressed as

$$\mathbf{v}(k) = \mathbf{w}(k) - \tilde{\mathbf{w}}(k). \quad (10.17)$$

Let us now subtract $\tilde{\mathbf{w}}(k+1)$ from both sides of (10.16), which yields

$$\begin{aligned} \mathbf{v}(k+1) &= \mathbf{w}(k) - \tilde{\mathbf{w}}(k+1) + \eta(k)q(k)\mathbf{II}(k) \\ &\quad - \eta(k)[\Phi(\mathbf{u}^T(k)\mathbf{w}(k)) - \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k))]\mathbf{II}(k). \end{aligned}$$

Using (10.10), we have

$$\begin{aligned} \mathbf{v}(k+1) &= \mathbf{w}(k) - \tilde{\mathbf{w}}(k) + \tilde{\mathbf{w}}(k) - \lambda\tilde{\mathbf{w}}(k) - \sqrt{1 - \lambda^2}\mathbf{n}(k) + \eta(k)q(k)\mathbf{II}(k) \\ &\quad - \eta(k)[\Phi(\mathbf{u}^T(k)\mathbf{w}(k)) - \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k))]\mathbf{II}(k). \end{aligned} \quad (10.18)$$

It then follows that $\mathbf{v}(k+1)$ becomes

$$\begin{aligned} \mathbf{v}(k+1) &= \mathbf{v}(k) + \eta(k)q(k)\mathbf{II}(k) - \eta(k)[\Phi(\mathbf{u}^T(k)\mathbf{w}(k)) - \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k))]\mathbf{II}(k) \\ &\quad + (1 - \lambda)\tilde{\mathbf{w}}(k) - \sqrt{1 - \lambda^2}\mathbf{n}(k). \end{aligned} \quad (10.19)$$

For $\Phi(k)$ a sign-preserving¹ contraction mapping (as in the case of the logistic function), the term in the square brackets from (10.19) is bounded from above by $\Theta|\mathbf{u}^T(k)\mathbf{v}(k)|$, $0 < \Theta < 1$ (Mandic and Chambers 2000e). Further analysis towards the weight convergence becomes rather involved because of the nature of $\mathbf{II}(k)$. Let us denote $\mathbf{u}^T(k)\mathbf{w}(k) = \text{net}(k)$. Since the gradient vector \mathbf{II} is a vector of partial derivatives of the output $y(k)$,

$$\mathbf{II}(k) = \frac{\partial y(k)}{\partial \mathbf{w}(k)} = \Phi'(\text{net}(k))[\mathbf{u}(k) + \mathbf{w}_a(k)\mathbf{II}_a(k)], \quad (10.20)$$

where the subscript ‘ a ’ denotes the elements which are due to the feedback of the system, we restrict ourselves to an approximation,

$$\mathbf{II}(k) \longrightarrow \Phi'(\text{net}(k))\mathbf{u}(k).$$

¹ For the sake of simplicity, we assume Φ sign preserving, i.e. for positive a, b , $b > a$, $\Phi(b) - \Phi(a) < b - a$. For other contractive activation functions, $|\Phi(a) - \Phi(b)| < |a - b|$, and norms of the corresponding expressions from the further analysis should be taken into account. The activation functions most commonly used in neural networks are sigmoidal, monotonically increasing, contractive, with a positive first derivative, so that this assumption holds.

This should not affect the generality of the result, since it is possible to return to the \mathbf{II} terms after the convergence results are obtained. In some cases, due to the problem of vanishing gradient, this approximation is quite satisfactory (Bengio *et al.* 1994). In fact, after approximating \mathbf{II} , the structure degenerates into a single-layer, single neuron feedforward neural network (Mandic and Chambers 2000f). For Φ a monotonic ascending contractive activation function, $\exists \alpha(k) \in (0, \Theta]$, such that the term $[\Phi(\mathbf{u}^T(k)\mathbf{w}(k)) - \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k))]$ from (10.19) can be replaced² by $\alpha(k)\mathbf{u}^T(k)\mathbf{v}(k)$. Now, analysing (10.19) with the newly introduced parameter $\alpha(k)$, we have

$$\begin{aligned} \mathbf{v}(k+1) = & \mathbf{v}(k) + \eta(k)q(k)\Phi'(\text{net}(k))\mathbf{u}(k) - \alpha(k)\eta(k)\mathbf{u}^T(k)\mathbf{v}(k)\Phi'(\text{net}(k))\mathbf{u}(k) \\ & + (1-\lambda)\tilde{\mathbf{w}}(k) - \sqrt{1-\lambda^2}\mathbf{n}(k). \end{aligned} \quad (10.21)$$

For a contractive activation function $0 < \Phi'(\text{net}(k)) < 1$ (Mandic and Chambers 1999b) and can be replaced³ by $\gamma(k)$. Equation (10.21) now becomes

$$\begin{aligned} \mathbf{v}(k+1) = & \mathbf{v}(k) + \gamma(k)\eta(k)q(k)\mathbf{u}(k) - \alpha(k)\gamma(k)\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)\mathbf{v}(k) \\ & + (1-\lambda)\tilde{\mathbf{w}}(k) - \sqrt{1-\lambda^2}\mathbf{n}(k). \end{aligned} \quad (10.22)$$

After including the zero-mean assumption for the driving noise, $\mathbf{n}(k)$ and the mutual statistical independence assumption between $\eta(k)$, $\mathbf{u}(k)$, $\mathbf{n}(k)$, $\tilde{\mathbf{w}}(k)$, $\alpha(k)$, $\gamma(k)$ and $\mathbf{v}(k)$, we have

$$E[\mathbf{v}(k+1)] = E[\mathbf{I} - \alpha\gamma\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)]E[\mathbf{v}(k)], \quad (10.23)$$

where $\gamma = E[\gamma(k)]$ and $\alpha = E[\alpha(k)]$, which are also in the range $(0, 1)$. For convergence,

$$0 < \|E[\mathbf{I} - \alpha\gamma\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)]\| < 1$$

as both α and γ are positive scalars for monotonic ascending contractive activation functions. For stability of the algorithm, the limits on $\eta(k)$ are thus⁴

$$0 < \eta(k) < E\left[\frac{2}{\alpha\gamma\mathbf{u}^T(k)\mathbf{u}(k)}\right]. \quad (10.24)$$

Equation (10.24) tells us that the stability limit for the NLMS algorithm is the bound for the simplified recurrent perceptron algorithm. By continuity, the NLMS algorithm for IIR adaptive filters is the bound for the stability analysis of a single-neuron RTRL algorithm. The mean square and steady-state convergence analysis follow the same form and are presented below.

² In fact, by the CMT, $\exists \xi \in (\mathbf{u}^T(k)\mathbf{w}(k), \mathbf{u}^T(k)\tilde{\mathbf{w}}(k))$ such that

$$|\Phi(\mathbf{u}^T(k)\mathbf{w}(k)) - \Phi(\mathbf{u}^T(k)\tilde{\mathbf{w}}(k))| = |\Phi'(\xi)| |\mathbf{u}^T(k)\mathbf{w}(k) - \mathbf{u}^T(k)\tilde{\mathbf{w}}(k)| = |\Phi'(\xi)| |\mathbf{u}^T(k)\mathbf{v}(k)|.$$

Hence, for a sigmoidal monotonic ascending, contractive Φ (logistic, tanh), the first derivative is strictly positive and $\alpha(k) = \Phi'(\xi)$. Assume positive $a, b, b > a$, then $\Phi(b) - \Phi(a) = \alpha(k)(b - a)$.

³ From (10.20), there is a finite $\gamma(k)$ such that $\|\mathbf{II}(k)\| = \gamma(k)\|\mathbf{u}(k)\|$. For simplicity, we approximate $\mathbf{II}(k)$ as above and use $\gamma(k)$ as defined by the CMT. The derived results, however, are valid for any finite $\gamma(k)$, i.e. are directly applicable for both the recurrent and feedforward architectures.

⁴ Using the independence assumption, $E[\mathbf{u}(k)\mathbf{u}^T(k)]$ is a diagonal matrix and its norm can be replaced by $E[\mathbf{u}^T(k)\mathbf{u}(k)]$.

10.5 Mean-Squared and Steady-State Mean-Squared Error Convergence

To investigate the mean squared convergence properties of stochastic gradient descent algorithms for recurrent neural networks, we need to analyse $\mathbf{R}_{\mathbf{v},\mathbf{v}}(k+1)$ which is defined as $\mathbf{R}_{\mathbf{v},\mathbf{v}}(k+1) = E[\mathbf{v}(k+1)\mathbf{v}^T(k+1)]$. From (10.22), cross-multiplying and applying the expectation operator to both sides and using the definition of $\mathbf{R}_{\mathbf{v},\mathbf{v}}(k+1)$, α and γ and the previous assumptions, we obtain⁵

$$\begin{aligned} \mathbf{R}_{\mathbf{v},\mathbf{v}}(k+1) &= \mathbf{R}_{\mathbf{v},\mathbf{v}}(k) - \alpha\gamma E[\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)]\mathbf{R}_{\mathbf{v},\mathbf{v}}(k) - \mathbf{R}_{\mathbf{v},\mathbf{v}}(k)E[\mathbf{u}(k)\mathbf{u}^T(k)\eta(k)]\gamma\alpha \\ &\quad + \alpha^2\gamma^2 E[\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)\mathbf{v}(k)\mathbf{v}^T(k)\mathbf{u}(k)\mathbf{u}^T(k)\eta(k)] \\ &\quad + \gamma^2 E[\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)\eta(k)]\sigma_q^2 \\ &\quad + (1-\lambda)^2 E[\tilde{\mathbf{w}}(k)\tilde{\mathbf{w}}^T(k)] + (1-\lambda^2)E[\mathbf{n}(k)\mathbf{n}^T(k)], \end{aligned} \quad (10.25)$$

where σ_q^2 is the variance of the noise signal $q(k)$. The expectation terms are now evaluated using $\eta = E[\eta(k)]$ and σ_u^2 as the variance of the i.i.d. input signal $\mathbf{u}(k)$, which implies

$$E[\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)]\mathbf{R}_{\mathbf{v},\mathbf{v}}(k) = \mathbf{R}_{\mathbf{v},\mathbf{v}}(k)E[\mathbf{u}(k)\mathbf{u}^T(k)\eta(k)] = \eta\sigma_u^2\mathbf{R}_{\mathbf{v},\mathbf{v}}(k), \quad (10.26)$$

$$E[\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)\eta(k)] = \eta^2\sigma_u^2\mathbf{I} \quad (10.27)$$

and by the fourth-order standard factorisation property of zero mean Gaussian variables⁶ (Papoulis 1984)

$$E[\eta(k)\mathbf{u}(k)\mathbf{u}^T(k)\mathbf{v}(k)\mathbf{v}^T(k)\mathbf{u}(k)\mathbf{u}^T(k)\eta(k)] = \eta^2\sigma_u^4[2\mathbf{R}_{\mathbf{v},\mathbf{v}}(k) + \mathbf{I}\text{tr}\{\mathbf{R}_{\mathbf{v},\mathbf{v}}(k)\}]. \quad (10.28)$$

⁵ For small quantities $E[x^2(k)] \approx (E[x(k)])^2$, so that $E[\alpha^2(k)] \approx \alpha^2$, $E[\gamma^2(k)] \approx \gamma^2$ and $E[\eta^2(k)] \approx \eta^2$. Experiments show that this is a realistic assumption for the range of allowed $\alpha(k)$, $\gamma(k)$ and $\eta(k)$. Moreover, if η is fixed, $\eta(k) = \eta$ and $E[\eta^2] = \eta^2$.

⁶ $E[\mathbf{x}_n\mathbf{x}_n^T\mathbf{x}_n\mathbf{x}_n^T]_{kl} = E[x(n-k)\sum_{i=1}^N x^2(n-i)x(n-l)]$, which by the standard factorisation property of real, zero mean Gaussian variables becomes

$$\begin{aligned} E[\mathbf{x}_1\mathbf{x}_2^T\mathbf{x}_3\mathbf{x}_4^T]_{kl} &= E[\mathbf{x}_1\mathbf{x}_2]E[\mathbf{x}_3\mathbf{x}_4] + E[\mathbf{x}_1\mathbf{x}_3]E[\mathbf{x}_2\mathbf{x}_4] + E[\mathbf{x}_1\mathbf{x}_4]E[\mathbf{x}_2\mathbf{x}_3] \\ &= 2\sum_{i=1}^N E[x(n-k)x(n-i)]E[x(n-l)x(n-i)] \\ &\quad + E[x(n-k)x(n-l)]\sum_{i=1}^N E[x^2(n-i)], \end{aligned}$$

which, in turn, implies

$$E[\mathbf{x}_n\mathbf{x}_n^T\mathbf{x}_n\mathbf{x}_n^T] = 2\mathbf{R}^2 + \mathbf{R}\text{tr}\{\mathbf{R}\},$$

where $\text{tr}\{\cdot\}$ is the trace operator. Now for i.i.d. Gaussian input signals \mathbf{x}_n , we have

$$E[x(n-i)x(n-j)] = \begin{cases} 0, & \text{if } i \neq j, \\ \sigma_x^2, & \text{if } i = j, \end{cases}$$

so that

$$E[\mathbf{x}_n\mathbf{x}_n^T\mathbf{x}_n\mathbf{x}_n^T]_{kl} = \begin{cases} 0, & \text{if } l \neq k, \\ (N+2)\sigma_x^4, & \text{if } l = k, \end{cases} \quad \text{and} \quad E[\mathbf{x}_n\mathbf{x}_n^T\mathbf{x}_n\mathbf{x}_n^T] = (N+2)\sigma_x^4\mathbf{I},$$

as required.

The first-order Markov model (10.10) used as the time-varying optimal weight system implies⁷ that

$$E[\tilde{\mathbf{w}}(k)\tilde{\mathbf{w}}^T(k)] = \sigma_n^2 \mathbf{I}, \quad (10.29)$$

$$E[\mathbf{n}(k)\mathbf{n}^T(k)] = \sigma_n^2 \mathbf{I}, \quad (10.30)$$

where σ_n^2 is the variance of the signal $\mathbf{n}(k)$. Combining (10.25)–(10.30), we have

$$\begin{aligned} \mathbf{R}_{v,v}(k+1) = \mathbf{R}_{v,v}(k) - 2\alpha\gamma\eta\sigma_u^2\mathbf{R}_{v,v}(k) + \alpha^2\gamma^2\eta^2\sigma_u^4[2\mathbf{R}_{v,v}(k) + \mathbf{I}\operatorname{tr}\{\mathbf{R}_{v,v}(k)\}] \\ + \gamma^2\eta^2\sigma_u^2\sigma_q^2\mathbf{I} + 2(1-\lambda)\sigma_n^2\mathbf{I}. \end{aligned} \quad (10.31)$$

The mean squared misalignment ξ , which is a commonly used quantity in the assessment of the performance of an algorithm, can be now defined as

$$\xi(k+1) = E[\mathbf{v}^T(k+1)\mathbf{v}(k+1)], \quad (10.32)$$

which can be obtained from $\mathbf{R}_{v,v}(k+1)$ by taking its trace. Thus, we have

$$\begin{aligned} \xi(k+1) = [1 - 2\alpha\gamma\eta\sigma_u^2 + \alpha^2\gamma^2\eta^2\sigma_u^4(N+2)]\xi(k) \\ + \gamma^2\eta^2\sigma_u^2\sigma_q^2N + 2(1-\lambda)N\sigma_n^2, \end{aligned} \quad (10.33)$$

where N is the length of vector $\mathbf{u}(k)$.

10.5.1 Convergence in the Mean Square

In order to guarantee convergence of the mean-square error (MSE), which is given under the above assumptions as

$$\text{MSE}(k) = \sigma_u^2\xi(k),$$

the update of the MSE has to be governed by a contraction mapping, i.e. from (10.33)

$$0 < |\alpha\gamma\eta\sigma_u^2[2 - \alpha\gamma\eta\sigma_u^2(N+2)]| < 2.$$

For convergence, the bounds on the learning rate η become⁸

$$0 < \eta < \frac{2}{\alpha\gamma\sigma_u^2(N+2)}. \quad (10.34)$$

The derived result is the upper bound for the learning rate which preserves the mean square convergence of the RTRL algorithm for a recurrent perceptron. Depending on the choice of γ , this is directly applicable for learning algorithms for both feedforward and recurrent neural networks. For a highly contractive Φ , α is small and η can be larger. For a linear activation function, $\alpha = \gamma = 1$, and the result (10.34) degenerates into the result for the LMS for linear FIR filters.

⁷ Vectors $\tilde{\mathbf{w}}$ and \mathbf{n} are drawn from the same statistical distribution $\mathcal{N}(\mathbf{0}, \sigma_n^2\mathbf{I})$.

⁸ Compare (10.34) with (10.24). From (10.24), for an i.i.d. input,

$$E\left[\frac{2}{\alpha\gamma\mathbf{u}^T(k)\mathbf{u}(k)}\right] \approx \frac{2}{\alpha\gamma N\sigma_u^2},$$

which means that the MSE stability condition (10.34) is more stringent than the mean weight error stability condition (10.24).

10.5.2 Steady-State Mean-Squared Error

Let us first derive the steady-state misalignment. Normally, this is obtained by setting $\xi = \xi(k) = \xi(k+1)$ in (10.33) and solving for ξ , and thus

$$\begin{aligned}\xi &= \frac{\gamma^2 \eta^2 \sigma_u^2 \sigma_q^2 N + 2(1-\lambda)N\sigma_n^2}{\alpha\gamma\eta\sigma_u^2[2 - \alpha\gamma\eta\sigma_u^2(N+2)]} \\ &= \frac{\gamma\eta\sigma_q^2 N}{\alpha[2 - \alpha\gamma\eta\sigma_u^2(N+2)]} + \frac{2(1-\lambda)N\sigma_n^2}{\alpha\gamma\eta\sigma_u^2[2 - \alpha\gamma\eta\sigma_u^2(N+2)]}.\end{aligned}\quad (10.35)$$

The steady-state MSE is then

$$\text{MSE} = \sigma_u^2 \xi. \quad (10.36)$$

The results for systems with a single fixed optimal weight solution can be obtained from the above by setting $\lambda = 1$.

10.6 Summary

Techniques for convergence analysis for an online stochastic gradient descent algorithm for neural adaptive filters have been provided. These are based upon the previously addressed contraction mapping properties of nonlinear neurons. The analysis has been undertaken for a general case of time-varying behaviour of the optimal weight vector. The learning algorithms for linear filters have been shown to be the bounds for the algorithms employed for neural networks. The analysis is applicable to both recurrent and feedforward architectures and can be straightforwardly extended to more complicated structures and learning algorithms.

11

Some Practical Considerations of Predictability and Learning Algorithms for Various Signals

11.1 Perspective

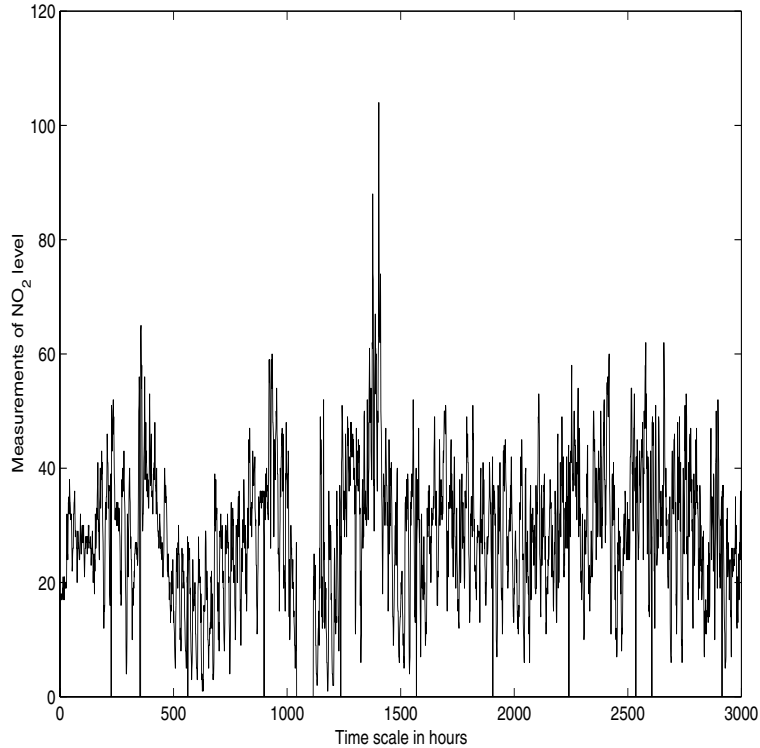
In this chapter, predictability, detecting nonlinearity and performance with respect to the prediction horizon are considered. Methods for detecting nonlinearity of signals are first discussed. Then, different algorithms are compared for the prediction of nonlinear and nonstationary signals, such as real NO₂ air pollutant and heart rate variability signals, together with a synthetic chaotic signal. Finally, bifurcations and attractors generated by a recurrent perceptron are analysed to demonstrate the ability of recurrent neural networks to model complex physical phenomena.

11.2 Introduction

When modelling a signal, an initial linear analysis is first performed on the signal, as linear models are relatively quick and easy to implement. The performance of these models can then determine whether more flexible nonlinear models are necessary to capture the underlying structure of the signal. One such standard model of linear time series, the auto-regressive integrated moving average, or ARIMA(p, d, q) model popularised by Box and Jenkins (1976), assumes that the time series x_k is generated by a succession of ‘random shocks’ ϵ_k , drawn from a distribution with zero mean and variance σ_ϵ^2 . If x_k is non-stationary, then successive differencing of x_k via the differencing operator, $\nabla x_k = x_k - x_{k-1}$ can provide a stationary process. A stationary process $z_k = \nabla^d x_k$ can be modelled as an autoregressive moving average

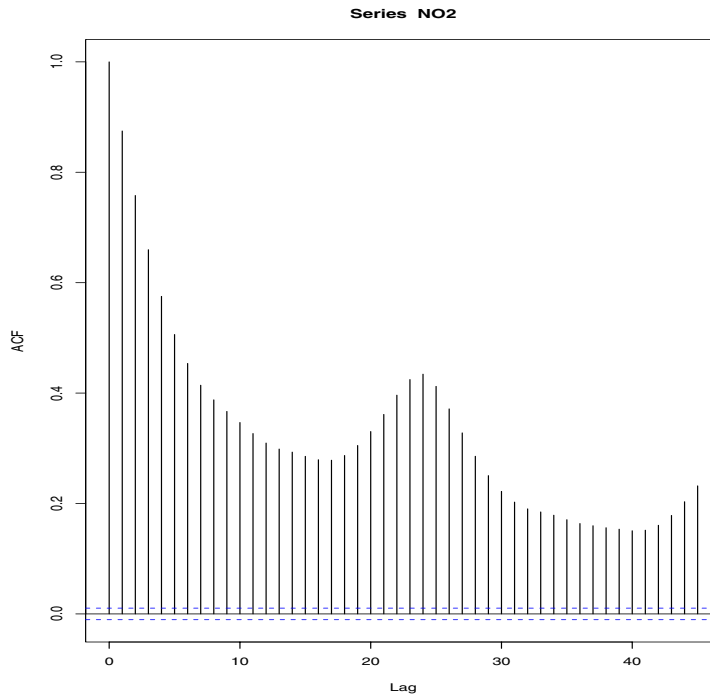
$$z_k = \sum_{i=1}^p a_i z_{k-i} + \sum_{i=1}^q b_i \epsilon_{k-i} + \epsilon_k. \quad (11.1)$$

Of particular interest are pure autoregressive (AR) models, which have an easily understood relationship to the nonlinearity detection technique of DVS (deterministic

(a) The raw NO₂ time series**Figure 11.1** The NO₂ time series and its autocorrelation function

versus stochastic) plots. Also, an ARMA(p, q) process can be accurately represented as a pure AR(p') process, where $p' \gg p + d$ (Brockwell and Davis 1991). Penalised likelihood methods such as AIC or BIC (Box and Jenkins 1976) exist for choosing the order of the autoregressive model to be fitted to the data; or the point where the *autocorrelation function* (ACF) essentially vanishes for all subsequent lags can also be used. The autocorrelation function for a wide-sense stationary time series x_k at lag h gives the correlation between x_k and x_{k+h} ; clearly, a non-zero value for the ACF at a lag h suggests that for modelling purposes at least the previous h lags should be used ($p \geq h$).

For instance, Figure 11.1 shows a raw NO₂ signal and its autocorrelation function (ACF) for lags of up to 40; the ACF does not vanish with lag and hence a high-order AR model is necessary to model the signal. Note the peak in the ACF at a lag of 24 hours and the rise to a smaller peak at a lag of 48 hours. This is evidence of *seasonal* behaviour, that is, the measurement at a given time of day is likely to be related to the measurement taken at the same time on a different day. The issue of seasonal time series is dealt with in Appendix J.

(b) The ACF of the NO₂ series**Figure 11.1** *Cont.*

11.2.1 Detecting Nonlinearity in Signals

Before deciding whether to use a linear or nonlinear model of a process, it is important to check whether the signal itself is linear or nonlinear. Various techniques exist for detecting nonlinearity in time series. Detecting nonlinearity is important because the existence of nonlinear structure in the series opens the possibility of highly accurate short-term predictions. This is not true for series which are largely stochastic in nature. Following the approach from Theiler *et al.* (1993), to gauge the efficacy of the techniques for detecting nonlinearity, a surrogate dataset is simulated from a high-order autoregressive model fit to the original series. Two main methods to achieve this exist, the first involves fitting a finite-order ARMA(p, q) model (we use a high-order AR(p) model to fit the data). The model coefficients are then used to generate the surrogate series, with the surrogate residuals ϵ_k taken as random permutations of the residuals from the original series. The second method involves taking a Fourier transform of the series. The phases at each frequency are replaced randomly from the uniform $(0, 2\pi)$ distribution while keeping the magnitude of each frequency the same as for the original series. The surrogate series is then obtained by taking the inverse Fourier transform. This series will have approximately the same autocor-

relation function as the original series, with the approximation becoming exact in the limit as $N \rightarrow \infty$. A discussion of the respective merits of the two methods of generating surrogate data is given in Theiler *et al.* (1993), the method used here is the former. Evidence of nonlinearity from any method of detection is negated if the method gives a similar result when applied to the surrogate series, which is known to be linear (Theiler *et al.* 1993).

11.3 Overview

This chapter deals with some practical issues when performing prediction of nonlinear and nonstationary signals. Techniques for detecting nonlinearity and chaotic behaviour of signals are first introduced and a detailed analysis is provided for the NO₂ air pollutant measurements taken at hourly intervals from the Leeds meteo station, UK. Various linear and nonlinear algorithms are compared for prediction of air pollutants, heart rate variability and chaotic signals. The chapter concludes with an insight into the capability of recurrent neural networks to generate and model complex nonlinear behaviour such as chaos.

11.4 Measuring the Quality of Prediction and Detecting Nonlinearity within a Signal

Existence and/or discovery of an attractor in the phase space demonstrates whether the system is deterministic, purely stochastic or contains elements of both. To reconstruct the attractor examine plots in the m -dimensional space of $[x_k, x_{k-\tau}, \dots, x_{k-(m-1)\tau}]^T$. It is critically important for the dimension of the space, m , in which the attractor resides, to be large enough to ‘untangle’ the attractor. This is known as the *embedding dimension* (Takens 1981). The value of τ , the *lag time* or *lag spacing*, is also important, particularly with noise present. The first inflection point of the autocorrelation function is a possible starting value for τ (Beule *et al.* 1999). Alternatively, if the series is known to be sampled coarsely, the value of τ can be taken as unity (Casdagli and Weigend 1993). A famous example of an attractor is given by the *Lorenz equations* (Lorenz 1963)

$$\left. \begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= rx - y - xz, \\ \dot{z} &= xy - bz, \end{aligned} \right\} \quad (11.2)$$

where σ , r and $b > 0$ are parameters of the system of equations. In Lorenz (1963) these equations were studied for the case $\sigma = 10$, $b = \frac{8}{3}$ and $r = 28$. A Lorenz attractor is shown in Figure 11.13(a). The discovery of an attractor for an air pollution time series would demonstrate chaotic behaviour; unfortunately, the presence of noise makes such a discovery unlikely. More robust techniques are necessary to detect the existence of deterministic structure in the presence of substantial noise.

11.4.1 Deterministic Versus Stochastic Plots

Deterministic versus stochastic (DVS) plots (Casdagli and Weigend 1993) display the (robust) prediction error $E(n)$ for local linear models against the number of nearest neighbours, n , used to fit the model, for a range of embedding dimensions m . The data are separated into a *test* set and a *training* set, where the test set is the last M elements of the series. For each element in the test set x_k , its corresponding delay vector in m -dimensional space

$$\mathbf{x}(k) = [x_{k-\tau}, x_{k-2\tau}, \dots, x_{k-m\tau}]^T \quad (11.3)$$

is constructed. This delay vector is then examined against the set of all the delay vectors constructed from the training set. From this set the n nearest neighbours are defined to be the n delay vectors $\mathbf{x}(k')$ which have the shortest Euclidean distance to $\mathbf{x}(k)$. These n nearest neighbours $\mathbf{x}(k')$ along with their corresponding target values $x_{k'}$ are used as the variables to fit a simple linear model. This model is then given $\mathbf{x}(k)$ as an input which provides a prediction \hat{x}_k for the target value x_k , with a robust prediction error of

$$|x_k - \hat{x}_k|. \quad (11.4)$$

This procedure is repeated for all the test set, enabling calculation of the mean robust prediction error,

$$E(n) = \frac{1}{M} \sum_{x_k \in T} |x_k - \hat{x}_k|, \quad (11.5)$$

where T is the test set. If the optimal number of nearest neighbours n , taken to be the value giving the lowest prediction error $E(n)$, is at, or close to, the maximum possible n , then globally linear models perform best and there is no indication of nonlinearity in the signal. As this global linear model uses all possible length m vectors of the series, it is equivalent to an AR model of order m when $\tau = 1$. Small optimal n suggests local linear models perform best, indicating nonlinearity and/or chaotic behaviour.

11.4.2 Variance Analysis of Delay Vectors

Closely related to DVS plots is the nonlinearity detection technique introduced in Khalaf and Nakayama (1998). The general idea is not to fit models, linear or otherwise, using the nearest neighbours of a delay vector, but rather to examine the variability of the set of targets corresponding to groups of close (in the Euclidean distance sense) delay vectors. For each observation x_k , $k \geq m + 1$ construct the group, Ω_k , of nearest neighbour delay vectors given by

$$\Omega_k = \{\mathbf{x}(k') : k' \neq k \text{ \& } d_{kk'} \leq \alpha A_x\}, \quad (11.6)$$

where $\mathbf{x}(k') = \{x_{k'-1}, x_{k'-2}, \dots, x_{k'-m}\}$, $d_{kk'} = \|\mathbf{x}(k') - \mathbf{x}(k)\|$ is the Euclidean distance, $0 < \alpha \leq 1$,

$$A_x = \frac{1}{N - m} \sum_{k=m+1}^N |x_k|$$

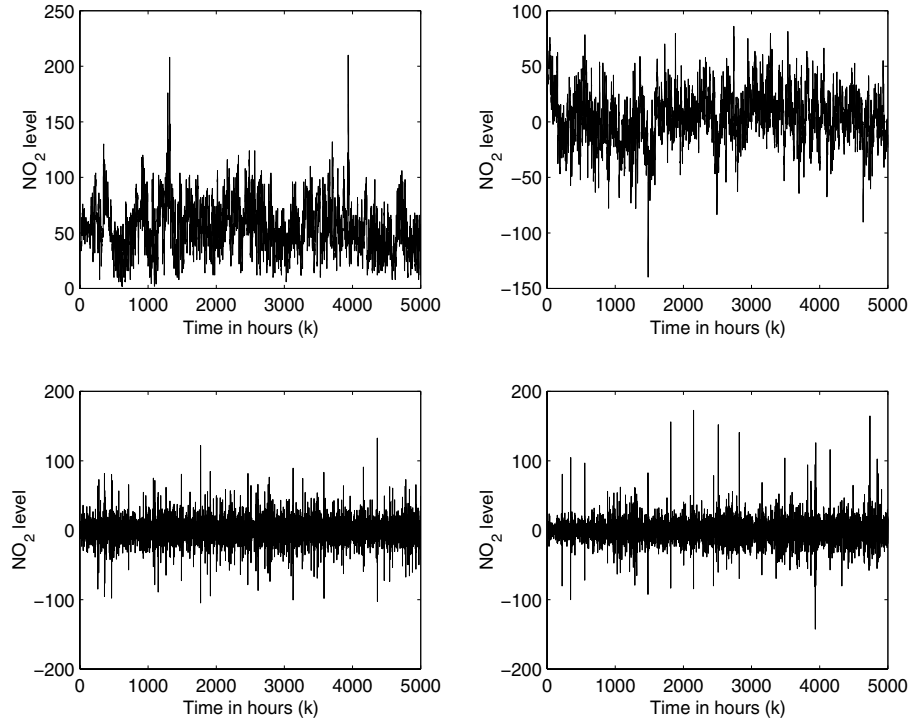


Figure 11.2 Time series plots for NO_2 . Clockwise, starting from top left: raw, simulated, simulated deseasonalised, deseasonalised

and N is the length of the time series. If the series is linear, then the similar patterns $\mathbf{x}(k')$ belonging to a group Ω_k will map onto similar $x_{k'}$ s. For nonlinear series, the patterns $\mathbf{x}(k')$ will not map onto similar $x_{k'}$ s. This is measured by the variance σ^2 of each group Ω_k

$$\sigma_k^2 = \frac{1}{|\Omega_k|} \sum_k (x_{k'} - \mu_k)^2, \quad \mathbf{x}(k') \in \Omega_k.$$

The measure of nonlinearity is taken to be the mean of σ_k^2 over all the Ω_k , denoted $\overline{\sigma_N^2}$, normalised by dividing through by σ_x^2 , the variance of the entire time series

$$\overline{\sigma^2} = \frac{\overline{\sigma_N^2}}{\sigma_x^2}.$$

The larger the value of $\overline{\sigma^2}$ the greater the suggestion of nonlinearity (Khalaf and Nakayama 1998). A comparison with surrogate data is especially important with this method to get evidence of nonlinearity.

11.4.3 Dynamical Properties of NO_2 Air Pollutant Time Series

The four time series generated from the NO_2 dataset are given in Figure 11.2, with the deseasonalised series on the bottom and the simulated series on the right. The

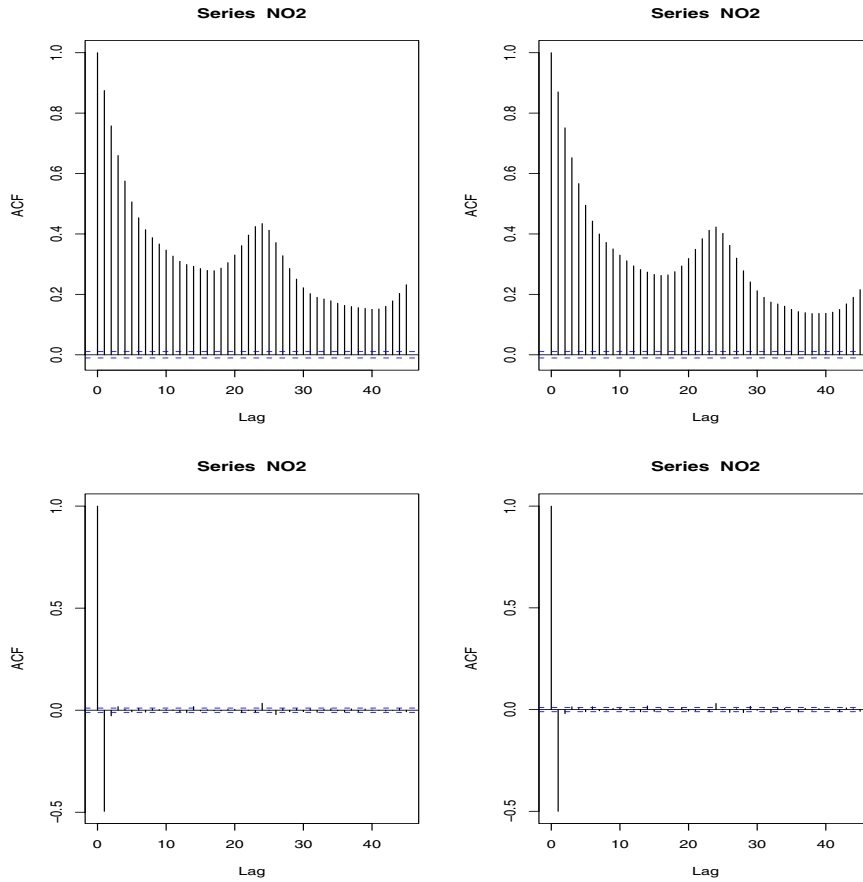


Figure 11.3 ACF plots for NO₂. Clockwise, starting from top left: raw, simulated, simulated deseasonalised, deseasonalised

sine wave structure can clearly be seen in the raw (unaltered) time series (top left), evidence confirming the relationship between NO₂ and temperature. Also note that once an air pollutant series has been simulated or deseasonalised, the condition that no readings can be below zero no longer holds. The respective ACF plots for the NO₂ series are given in Figure 11.3. The raw and simulated ACFs (top) are virtually identical – as should be the case, since the simulated time series is based on a linear AR(45) fit to the raw data, the correlations for the first 45 lags should be the same. Since generating the deseasonalised data involves application of the backshift operator, the autocorrelations are much reduced, although a ‘mini-peak’ can still be seen at a lag of 24 hours.

Nonlinearity detection in NO₂ signal

Figure 11.4 shows the two-dimensional attractor reconstruction for the NO₂ time series after it has been passed through a linear filter to remove some of the noise

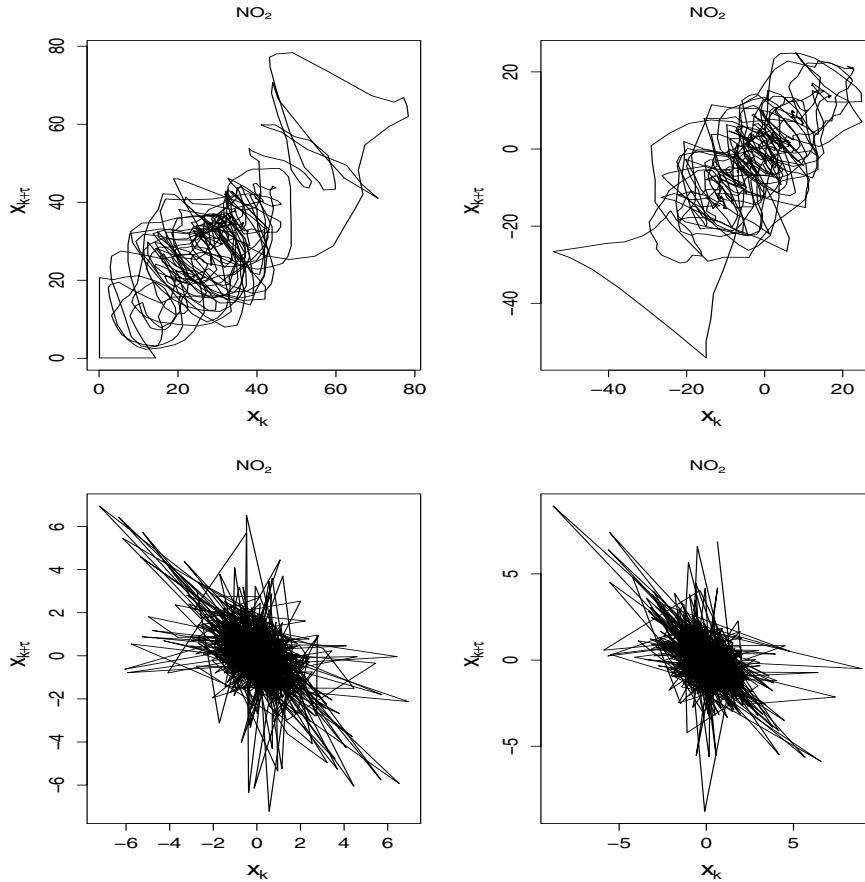


Figure 11.4 Attractor reconstruction plots for NO_2 . Clockwise, starting from top left: raw, simulated, simulated deseasonalised and deseasonalised

present. This graph shows little regularity and there is little to distinguish between the raw and the simulated plots. If an attractor does exist, then it is in a higher-dimensional space or is swamped by the random noise. The DVS plots for NO_2 are given in Figure 11.5, the DVS analysis of a related air pollutant can be found in Foxall *et al.* (2001). The optimal n (that is, the value of n corresponding to the minimum of $E(n)$), is clearly less than the maximum of n for the raw data for each of the embedding dimensions (m) examined. However, the difference is not great and the minimum occurs quite close to the maximum n , so this only provides weak evidence for nonlinearity. The DVS plot for the simulated series obtains the optimal error measure at the maximum n , as is expected. The deseasonalised DVS plots follow the same pattern, except that the evidence for nonlinearity is weaker, and the best embedding dimension now is $m = 6$ rather than $m = 2$. Figure 11.6 shows the results from analysing the variance of the delay vectors for the NO_2 series. The top two plots show lesser variances for the raw series, strongly suggesting nonlinearity. However, for

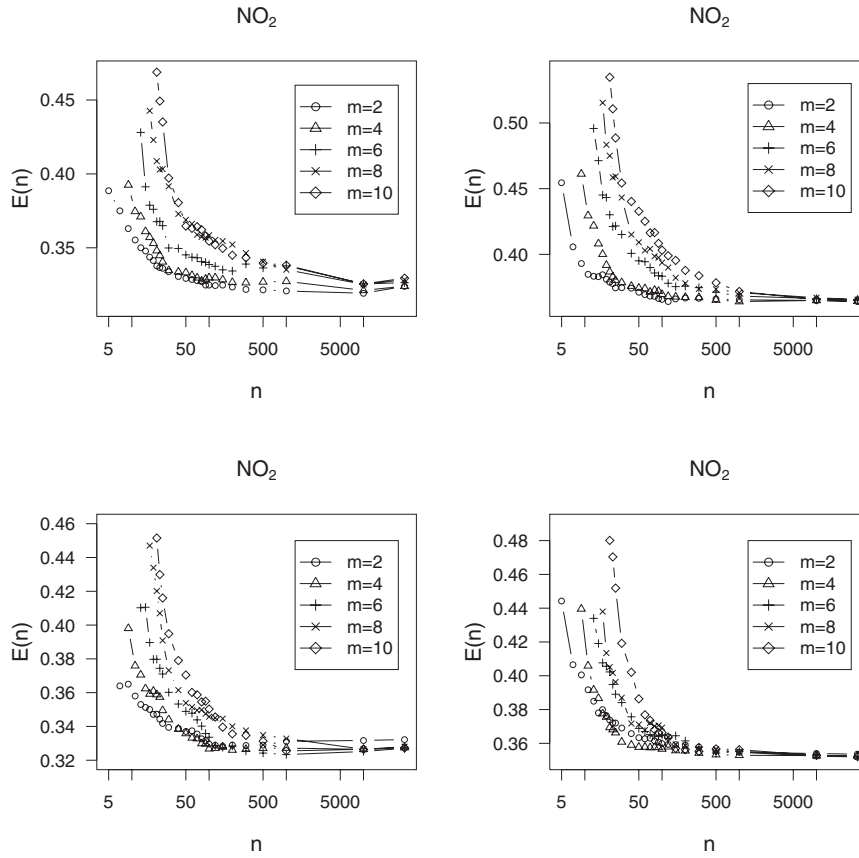


Figure 11.5 DVS plots for NO₂. Clockwise, starting from top left: raw, simulated, simulated deseasonalised and deseasonalised

Table 11.1 Performance of gradient descent algorithms in prediction of the NO₂ time series

	NGD	NNGD	Recurrent perceptron	NLMS
Predicted gain (dB)	5.78	5.81	6.04	4.75

the deseasonalised series (bottom) the variances are roughly equal, and indeed greater for higher embedding dimensions, suggesting that evidence for nonlinearity originated from the seasonality of the data.

To support the analysis, experiments on prediction of this signal were performed. The air pollution data represent hourly measurements of the concentration of nitrogen dioxide (NO₂), over the period 1994–1997, provided by the Leeds meteo station.

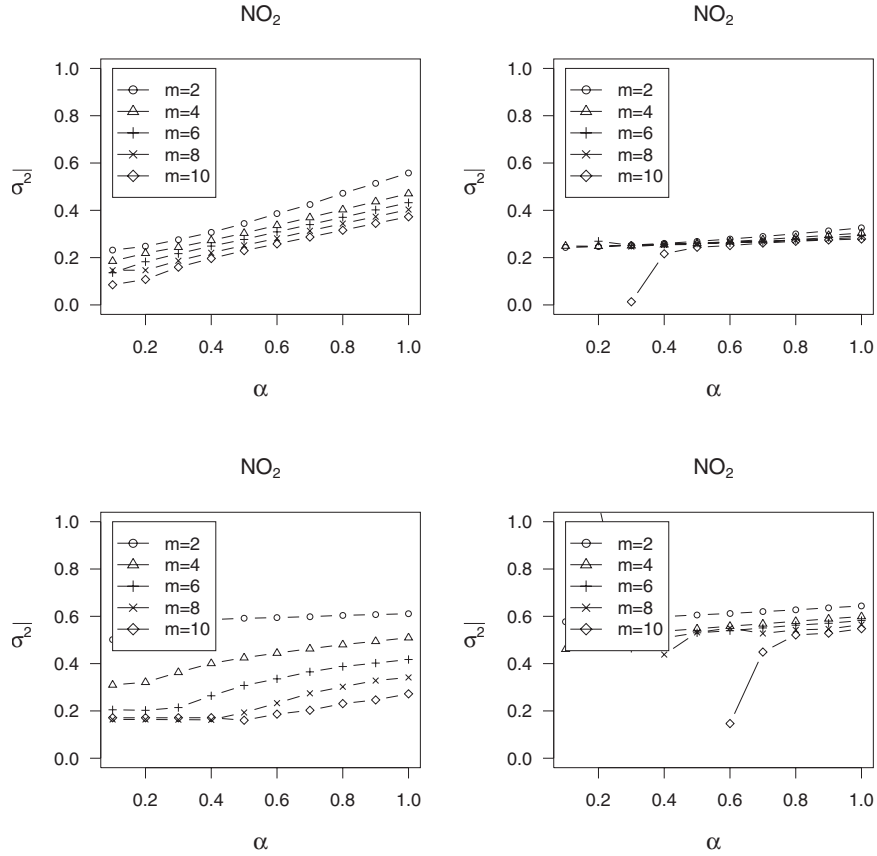


Figure 11.6 Delay vector variance plots for NO₂. Clockwise, starting from top left: raw, simulated, simulated deseasonalised and deseasonalised

In the experiments the logistic function was chosen as the nonlinear activation function of a dynamical neuron (Figure 2.6). The quantitative performance measure was the standard prediction gain, a logarithmic ratio between the expected signal and error variances $R_p = 10 \log(\hat{\sigma}_s^2 / \hat{\sigma}_e^2)$. The slope of the nonlinear activation function of the neuron β was set to be $\beta = 4$. The learning rate parameter η in the NGD algorithm was set to be $\eta = 0.3$ and the constant C in the NNGD algorithm was set to be $C = 0.1$. The order of the feedforward filter N was set to be $N = 10$. For simplicity, a NARMA(3,1) recurrent perceptron was used as a recurrent network. The summary of the performed experiments is given in Table 11.1. From Table 11.1, the nonlinear algorithms perform better than the linear one, confirming the analysis which detected nonlinearity in the signal. To further support the analysis given in the DVS plots, Figure 11.7(a) shows prediction gains versus number of taps for linear and nonlinear feedforward filters trained by the NGD, NNGD and NLMS algorithms, whereas Figure 11.7(b) shows prediction performance of a recurrent perceptron (Fox-

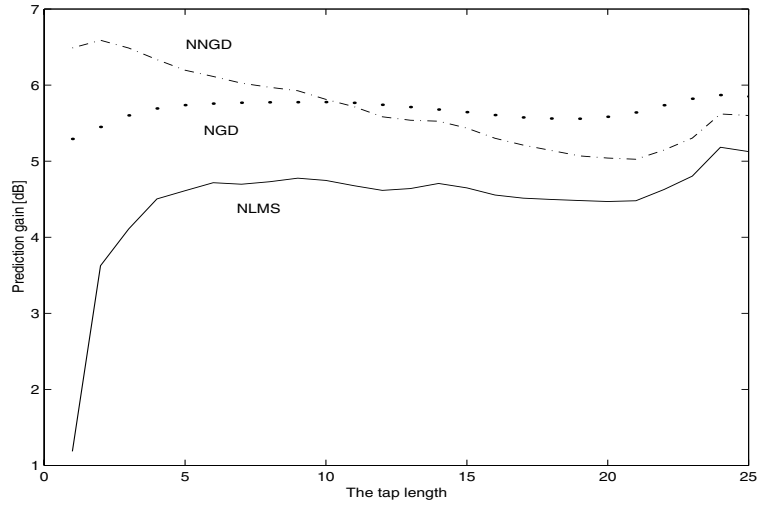
all *et al.* 2001). Both the nonlinear filters trained by the NGD and NNGD algorithms outperformed the linear filter trained by the NLMS algorithm. For the tap length up to $N = 10$, the NNGD was outperforming the NGD; the worse performance of the NNGD over the NGD for $N > 10$ can be explained by the insufficient approximation of the remainder of the Taylor series expansion within the derivation of the algorithm for large N . The recurrent structure achieved better performance for a smaller number of tap inputs than the standard feedforward structures.

11.5 Experiments on Heart Rate Variability

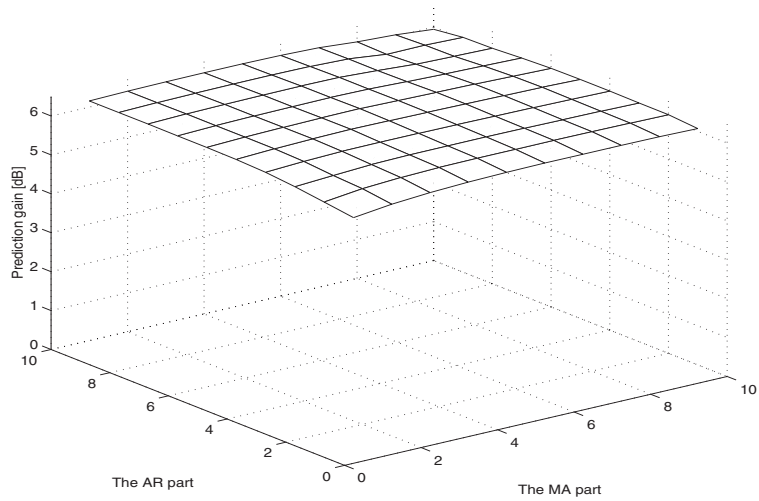
Information about heart rate variability (HRV) is extracted from the electrocardiogram (ECG). There are different approaches to the assessment of HRV from the measured data, but most of them rely upon the so-called R–R intervals, i.e. distance in time between two successive R waves in the HRV signal. Here, we use the R–R intervals that originate from ECG obtained from two patients. The first patient (A) was male, aged over 60, with a normal sinus rhythm, while patient (B) was also male, aged over 60, who suffered a myocardial infarction. In order to examine predictability of HRV signals, we use various gradient-descent-based neural adaptive filters.

11.5.1 Experimental Results

Figure 11.8(a) shows the HRV for patient A, while Figure 11.8(b) shows HRV for patient B. Prediction was performed using a logistic activation function Φ of a dynamical neuron with $N = 10$. The quantitative performance measure was the standard prediction gain $R_p = 10 \log(\hat{\sigma}_s^2/\hat{\sigma}_e^2)$. The slope of the nonlinear activation function of the neuron β was set to be $\beta = 4$. Due to the saturation type logistic nonlinearity, input data were prescaled to fit within the range of the neuron activation function. Both the standard NGD and the data-reuse modifications of the NGD algorithm were used. The number of data-reuse iterations L was set to be $L = 10$. The performance comparison between the NGD algorithm and a data-reusing NGD algorithm is shown in Figure 11.9. The plots show the prediction gain versus the tap length and the prediction horizon (number of steps ahead in prediction). In all the cases from Figure 11.9, the data-reusing algorithms outperformed the standard algorithms for short-term prediction. The standard algorithms showed better prediction results for long-term prediction. As expected, the performance deteriorates with the order of prediction ahead. In the next experiment we compare the performance of a recurrent perceptron trained with the fixed learning rate $\eta = 0.3$ and a recurrent perceptron trained by the NRTRL algorithm on prediction of the HRV signal. In the experiment the MA and the AR part of the recurrent perceptron vary from 1 to 15, while prediction horizon varies from 1 to 10. The results of the experiment are shown in Figures 11.10 and 11.11. From Figure 11.10, for a relatively large input line and feedback tap delay lines, there is a saturation in performance. This confirms that the recurrent structure was able to capture the dynamics of the HRV signal. The prediction performance deteriorates with the prediction step, and due to the recurrent nature of the filter, the performance is not good for a NARMA recurrent perceptron with



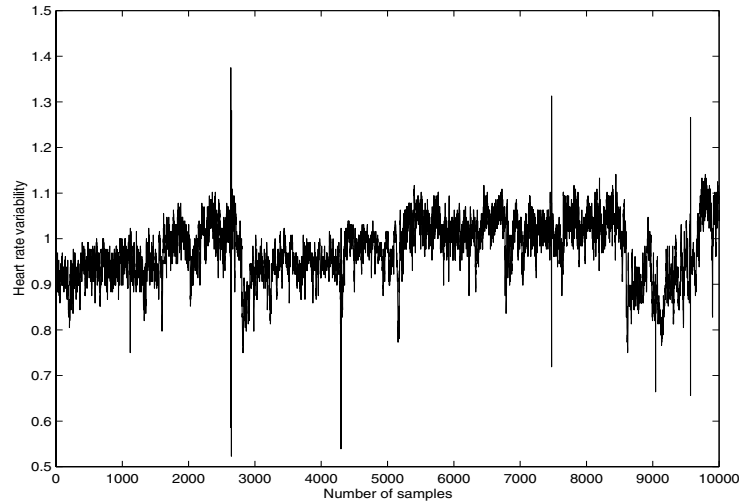
(a) Performance of the NGD, NNGD and NLMS algorithms in the prediction of NO_2 time series



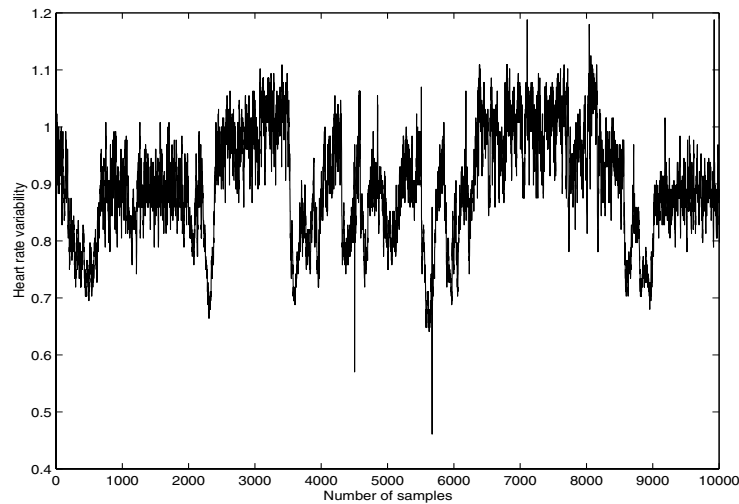
(b) Performance of the recurrent perceptron in the prediction of NO_2 time series

Figure 11.7 Performance comparison of various structures for prediction of NO_2 series

a small order of the AR and MA part. Figure 11.11 shows the results of an experiment similar to the previous one, with the exception that the employed algorithm was the NRTRL algorithm. The $\text{NARMA}(p, q)$ recurrent perceptron trained with this



(a) HRV signal for patient A

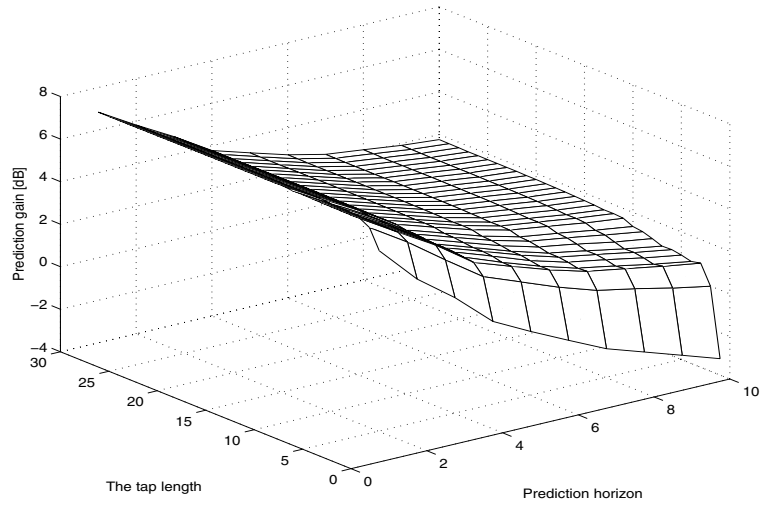


(b) HRV signal for patient B

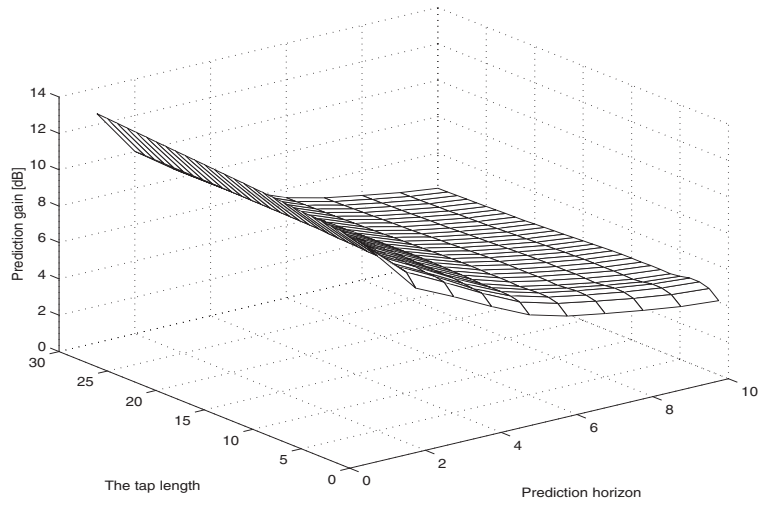
Figure 11.8 Heart rate variability signals for patients A and B

algorithm persistently outperformed the standard recurrent perceptron trained by the RTRL.

Figure 11.12 shows performance of the recurrent perceptron with fixed η in prediction of HRV time series (patient B), for different prediction horizons. Similar arguments as for patient A are applicable.

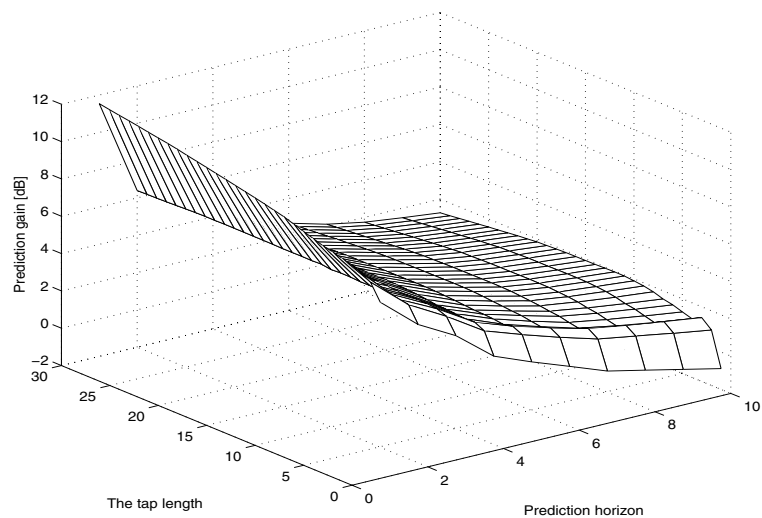


(a) Performance of the NGD algorithm in prediction of HRV time series, patient A

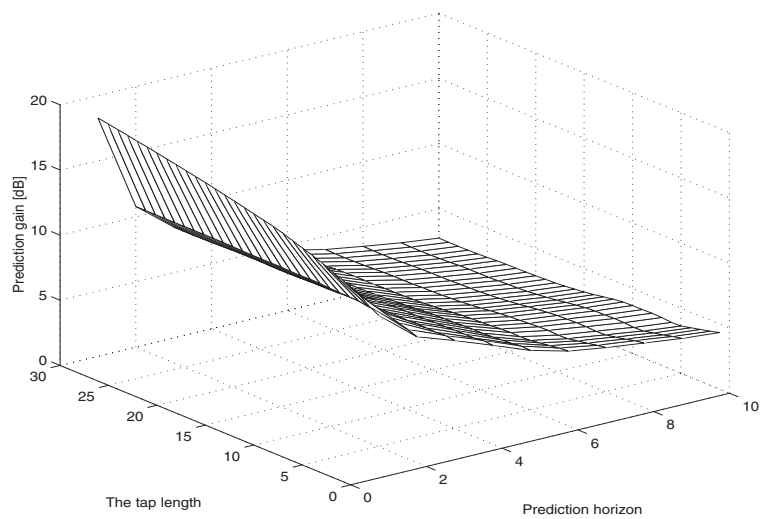


(b) Performance of the NGD algorithm in prediction of HRV time series, patient B

Figure 11.9 Performance comparison between standard and data-reusing algorithms for prediction of HRV signals

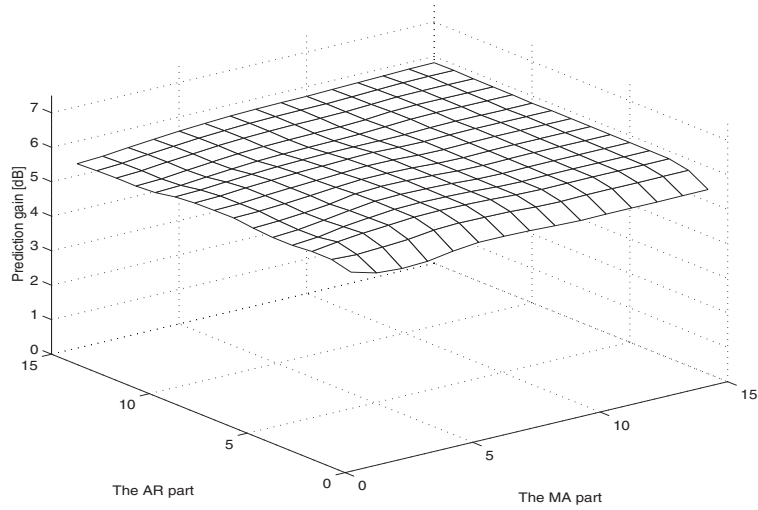


(c) Performance of the data-reusing NGD algorithm in prediction of HRV time series, patient A, $L = 10$

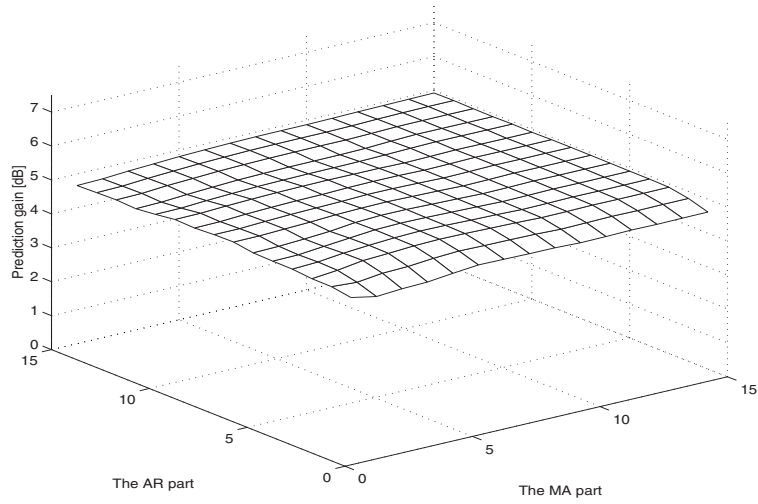


(d) Performance of the data-reusing NGD algorithm in prediction of HRV time series, patient B, $L = 10$

Figure 11.9 *Cont.*

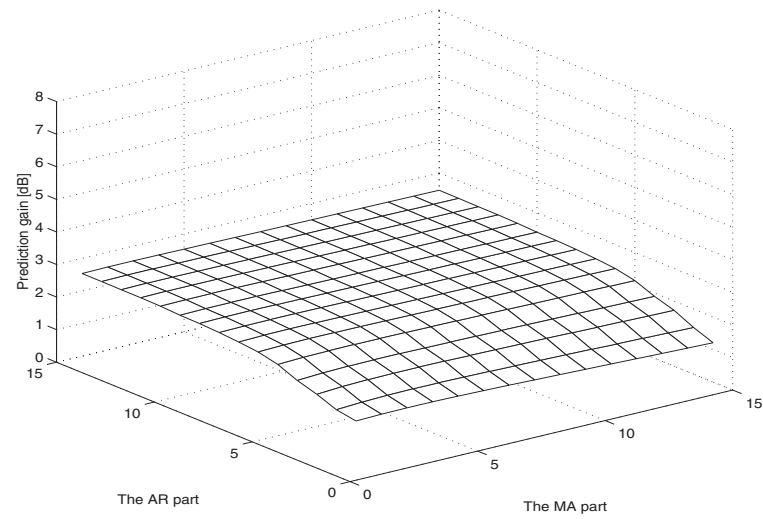


(a) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient A, prediction horizon is 1

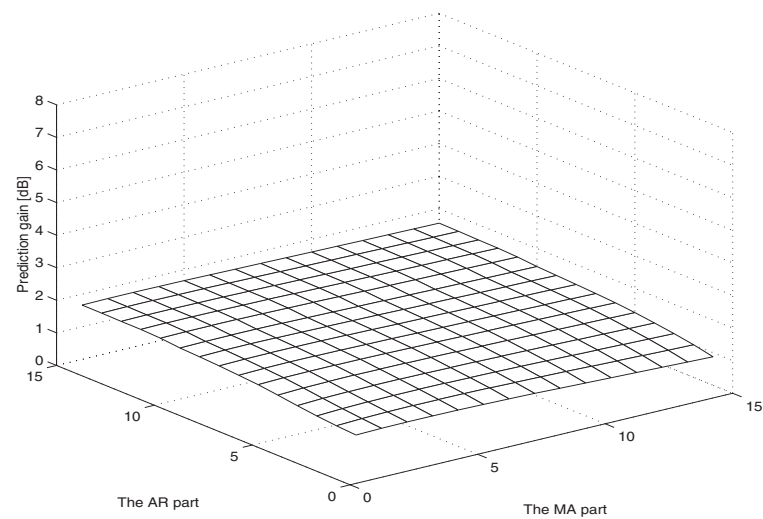


(b) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient A, prediction horizon is 2

Figure 11.10 Performance of a NARMA recurrent perceptron on prediction of HRV signals for different prediction horizons

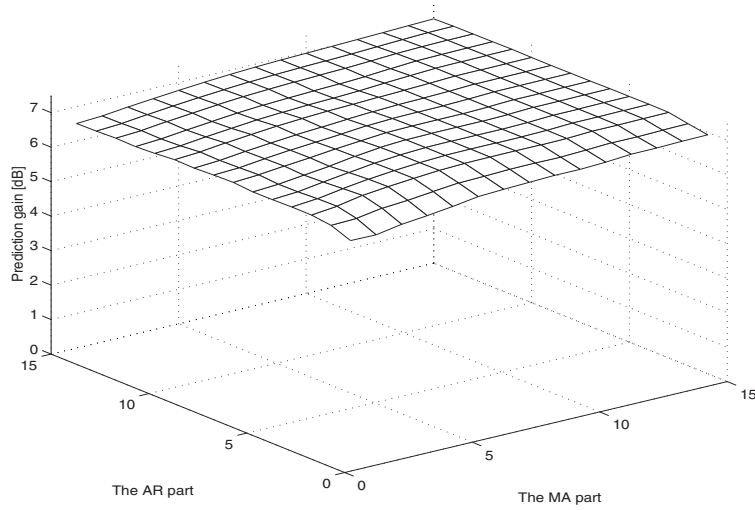


(c) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient A, prediction horizon is 5

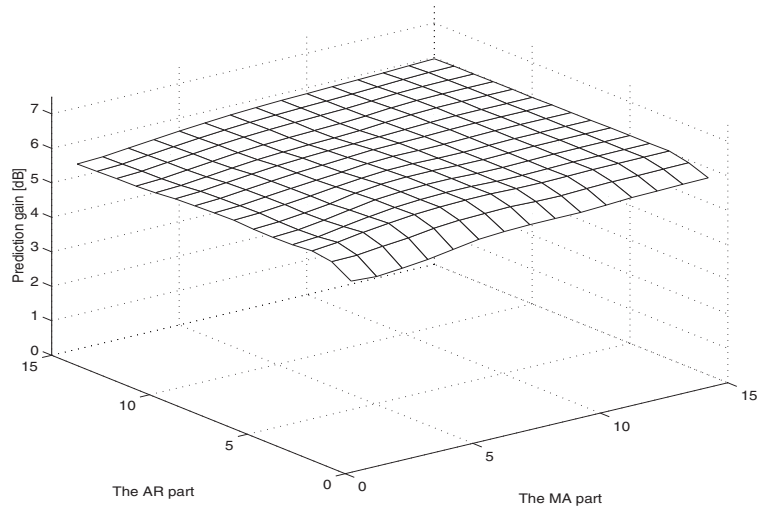


(d) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient A, prediction horizon is 10

Figure 11.10 *Cont.*

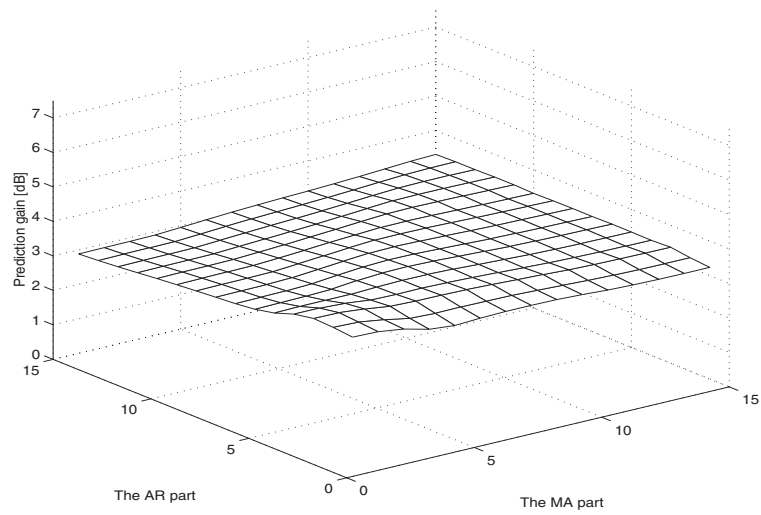


(a) Performance of the recurrent perceptron trained with the NRTRL algorithm in prediction of HRV time series, patient A, prediction horizon is 1

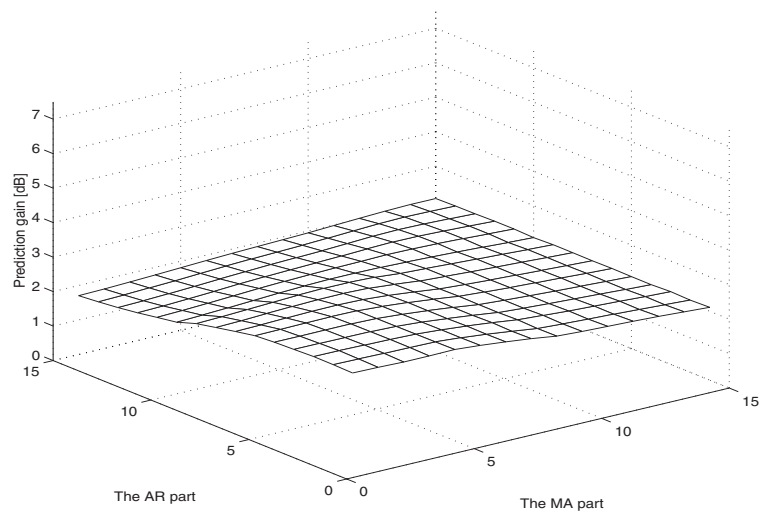


(b) Performance of the recurrent perceptron trained with the NRTRL algorithm in prediction of HRV time series, patient A, prediction horizon is 2

Figure 11.11 Performance of the NRTRL algorithms on prediction of HRV, for different prediction horizons

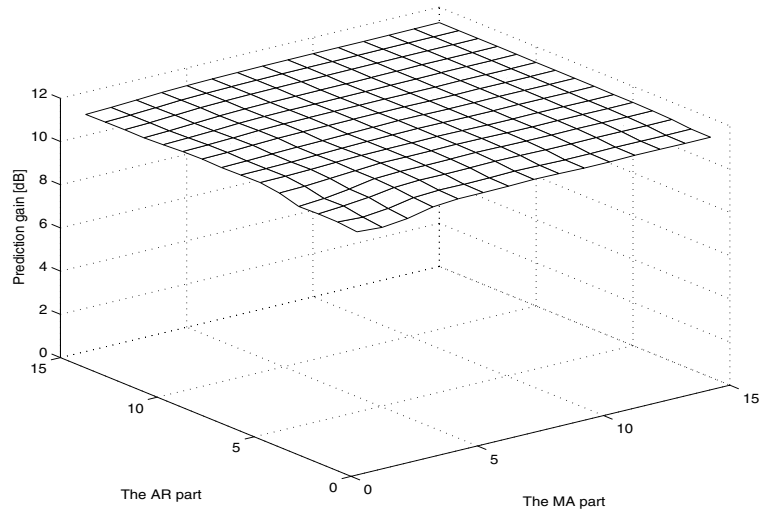


(c) Performance of the recurrent perceptron trained with the NRTRL algorithm in prediction of HRV time series, patient A, prediction horizon is 5

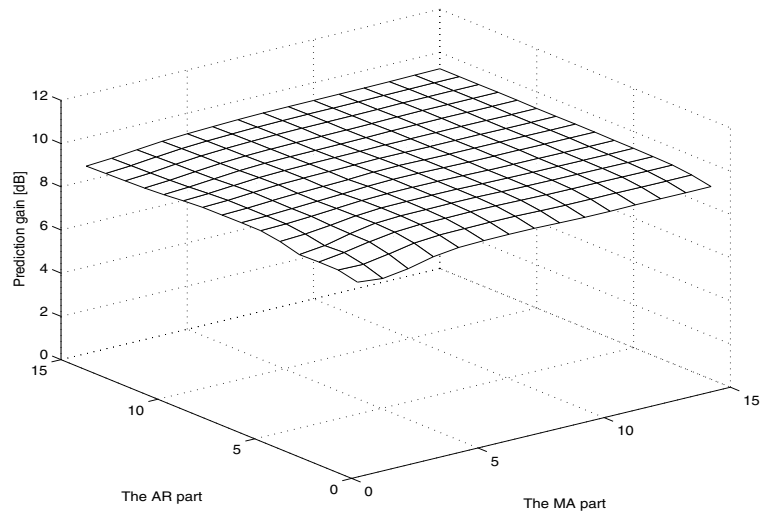


(d) Performance of the recurrent perceptron trained with the NRTRL algorithm in prediction of HRV time series, patient A, prediction horizon is 10

Figure 11.11 *Cont.*

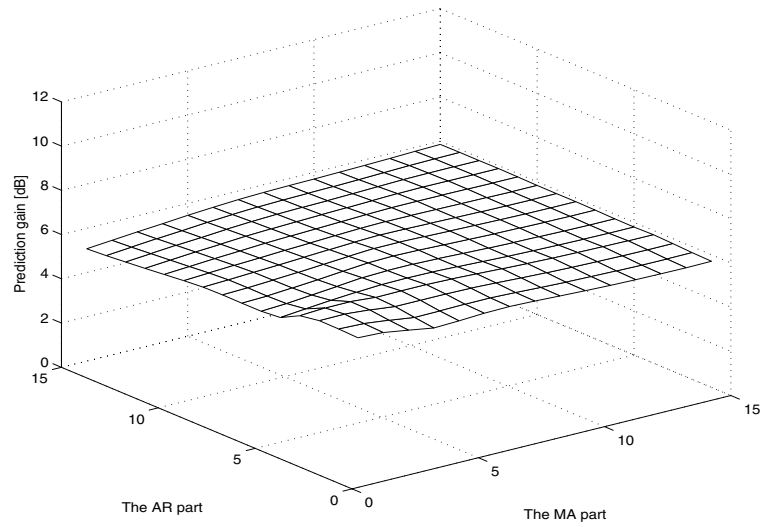


(a) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient B, prediction horizon is 1

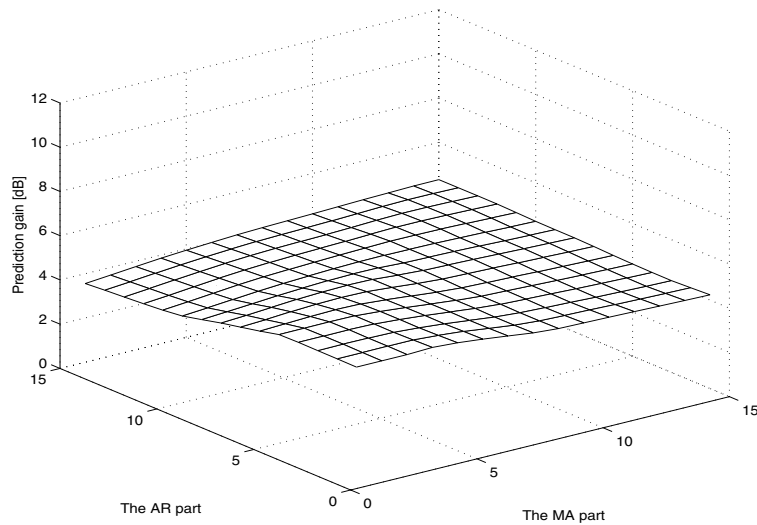


(b) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient B, prediction horizon is 2

Figure 11.12 Performance of a recurrent perceptron for prediction of HRV signals for different prediction horizons

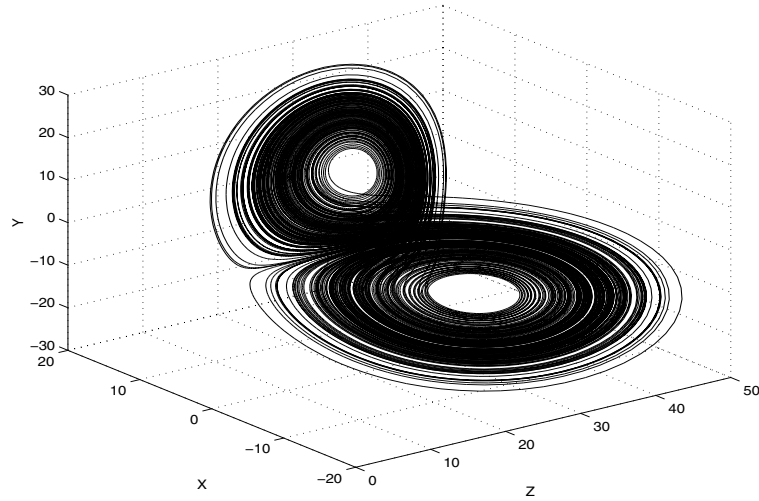


(c) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient B, prediction horizon is 5

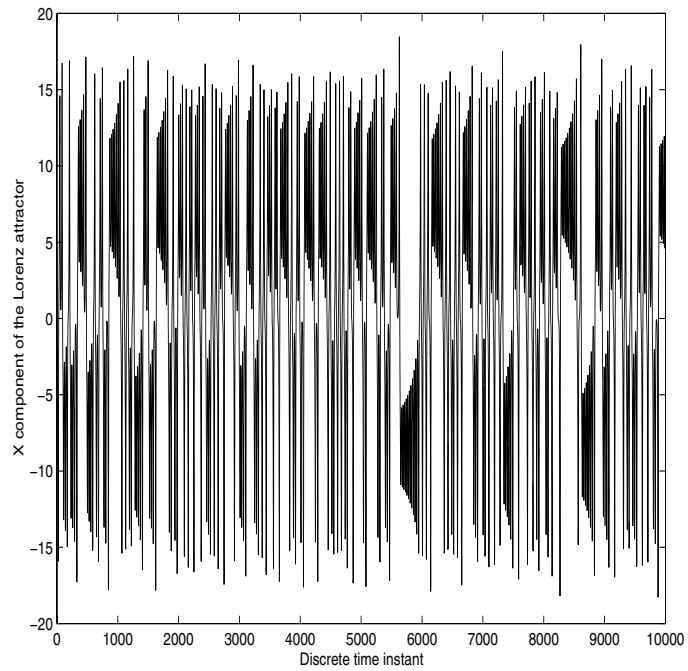


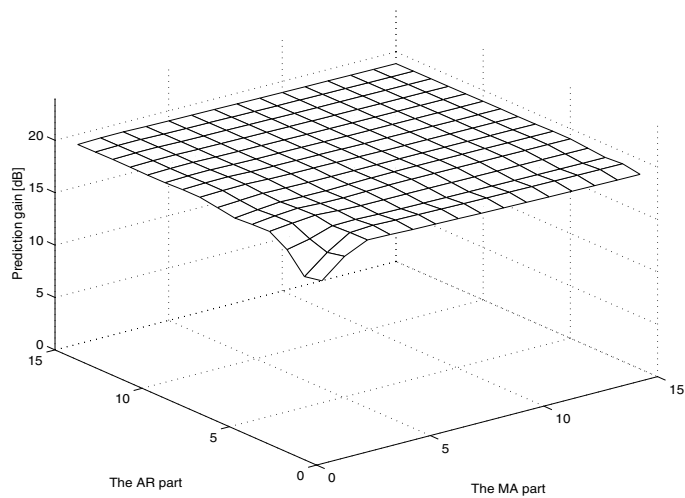
(d) Performance of the recurrent perceptron with fixed learning rate in prediction of HRV time series, patient B, prediction horizon is 10

Figure 11.12 *Cont.*

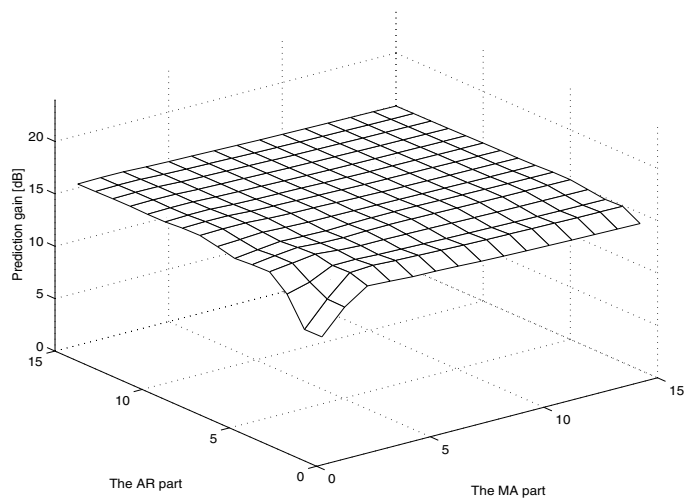


(a) The Lorenz attractor

(b) The x -component of Lorenz attractor**Figure 11.13** The Lorenz attractor and its x -component

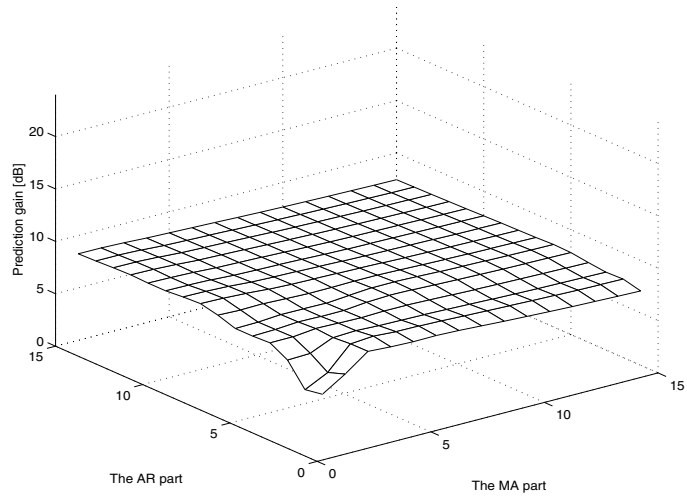


(a) Performance of the recurrent perceptron with fixed learning rate in prediction of the x -component of Lorenz system, prediction horizon is 1

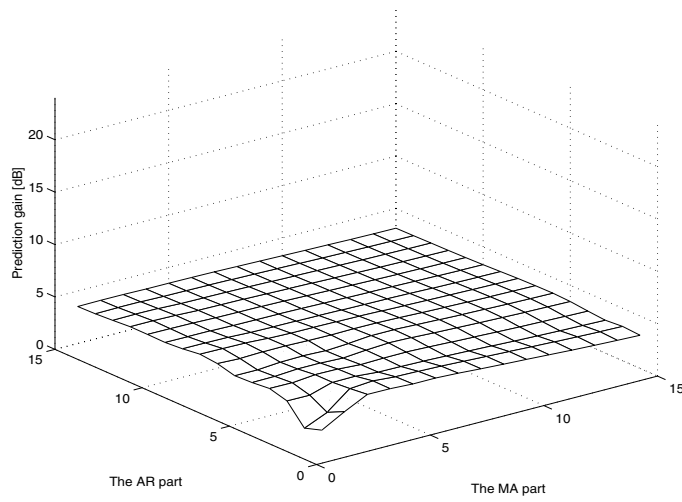


(b) Performance of the recurrent perceptron with fixed learning rate in prediction of the x -component of Lorenz system, prediction horizon is 2

Figure 11.14 Performance of algorithms for prediction of the x -component of the Lorenz system

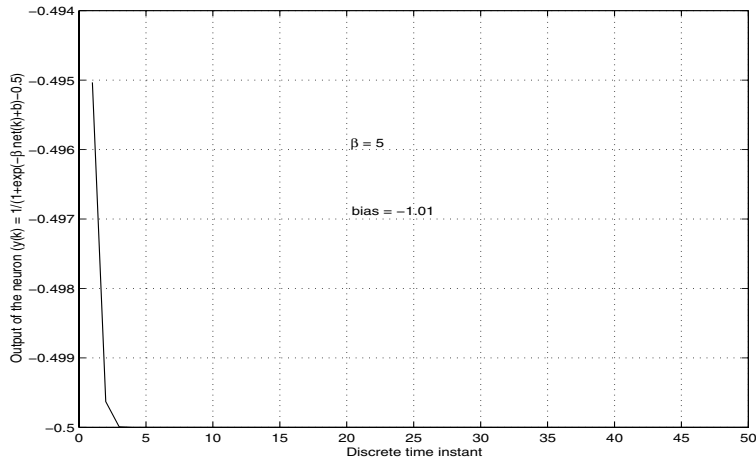


(c) Performance of the recurrent perceptron with fixed learning rate in prediction of the x -component of Lorenz system, prediction horizon is 5

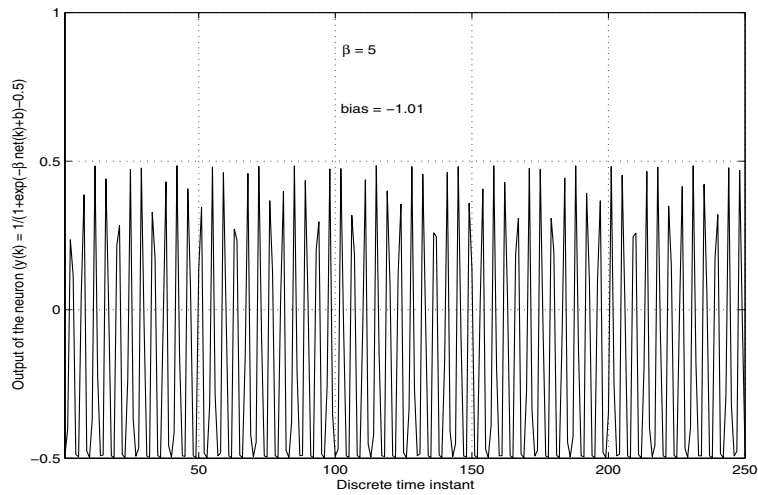


(d) Performance of the recurrent perceptron with fixed learning rate in prediction of the x -component of Lorenz system, prediction horizon is 10

Figure 11.14 *Cont.*



(a) Output of the recurrent perceptron with the logistic activation function, $\mathbf{w} = [1, 1.5, 1.5]$

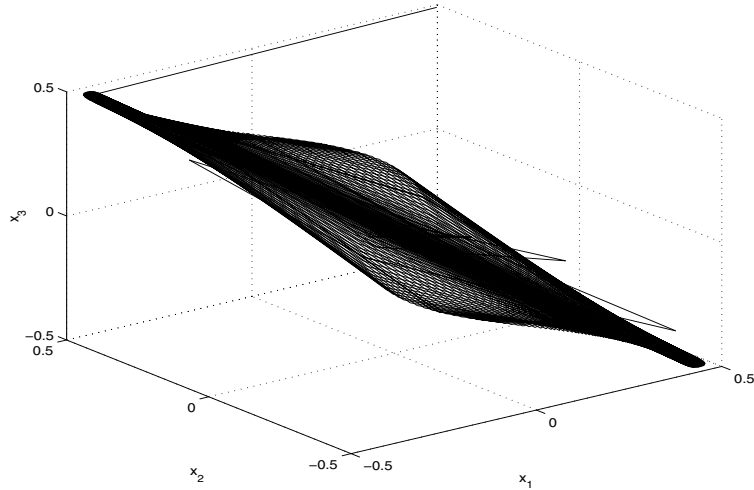


(b) Output of the recurrent perceptron with the logistic activation function, $\mathbf{w} = [-1, -1.5, -1.5]$

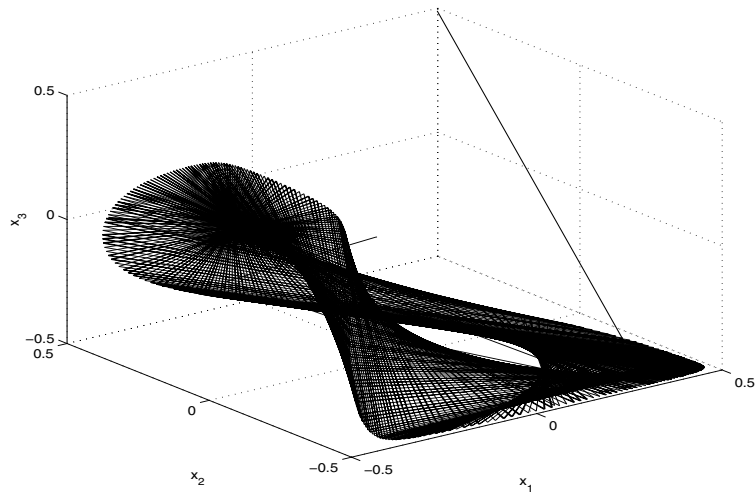
Figure 11.15 Output of a perceptron for different activation functions

11.6 Prediction of the Lorenz Chaotic Series

The Lorenz attractor is a system of three nonlinear coupled differential equations, given by (11.2). In order to perform experiments on the Lorenz system, the set of



(a) Attractor of the recurrent perceptron with the logistic activation function, bias = -0.01



(b) Attractor of the recurrent perceptron with the logistic activation function, bias = -1.01

Figure 11.16 Attractor diagrams for a recurrent perceptron and various biases

nonlinear Equations (11.2) was integrated using the Runge–Kutta method of numerical integration, starting from the initial state $[5, 20, -5]^T$. Thus, we obtained three discrete time series, sampled with sampling frequency $f_s = 40$ Hz. Figure 11.13 shows

the Lorenz attractor and its x -component. Different GD-based neural adaptive filters were employed in prediction of the x -component of the Lorenz system. In applied neural adaptive filters the parameters of the filters were the same as for the previous experiments. Due to the saturation type of logistic nonlinearity, input data were prescaled to fit within the range of a neuron activation function. In the experiment the MA and the AR part of the recurrent perceptron vary from 1 to 15, while the prediction horizon varies from 1 to 10. Figure 11.14 shows the performance of the recurrent perceptron with fixed η in prediction of the x -component of the Lorenz system, for different prediction horizons. Again, the recurrent structure was able to capture the dynamics of the modelled signal. As expected, due to the nature of the signal, the nonlinear structures were able to achieve good short term prediction performance, whereas the long term prediction performance was bad.

11.7 Bifurcations in Recurrent Neural Networks

Presence of feedback in a neural network structure may result in a very complex output behaviour. The output of the RNN is bounded, due to the saturation type of nonlinear activation function of an output neuron. Further, behaviour of an RNN can be described by the set of nonlinear difference equations. Evidence of chaotic motion within the dynamics of an RNN has been given in Dror and Tsodyks (2000) and Chen and Aihara (1997). Even though the recurrent perceptron has a simple structure, for certain values of the network parameters, the dynamics of a recurrent perceptron can exhibit a very complex behaviour. Here, we look at the value of the slope β of a neuron activation function and the value of the bias term b . Figure 11.15(a) shows the output of the recurrent perceptron with the logistic activation function shifted by -0.5 , i.e. $y = (1/(1 + e^{-\beta x})) - 0.5$, with $\beta = 5$ and $q = 3$,¹ weight vector $\mathbf{w} = [1, 1.5, 1.5]^T$ and bias = -1.01 . Figure 11.15(b) shows the output of the recurrent perceptron with the logistic activation function shifted by -0.5 , i.e. $y = (1/(1 + e^{-\beta x})) - 0.5$, with $\beta = 5$, $q = 3$, weight vector $\mathbf{w} = [-1, -1.5, -1.5]^T$ and bias = -1.01 . In both cases, the dynamics of the recurrent system were autonomous, hence performing a kind of fixed point iteration and producing the so-called bifurcation maps. From the figure, the recurrent structures produce complex dynamical behaviour. To further depict the ability of recurrent neural networks to produce complex dynamics, we plot the attractors for the above simulated bifurcation diagrams. Figure 11.16(a) shows the attractor of the recurrent perceptron with the logistic activation function shifted by -0.5 , i.e. $y = (1/(1 + \exp(-\beta x))) - 0.5$, with $\beta = 5$, $q = 3$, weight vector $\mathbf{w} = [-1, -1.5, -1.5]^T$ and bias = -0.01 . Figure 11.16(b) shows the attractor of the recurrent perceptron with the logistic activation function shifted by -0.5 , i.e. $y = (1/(1 + \exp(-\beta x))) - 0.5$, with $\beta = 5$, $q = 3$, weight vector $\mathbf{w} = [-1, -1.5, -1.5]^T$ and bias = -1.01 . The x_1 -, x_2 - and x_3 -axes from Figure 11.16 represent, respectively, the outputs $y(k)$, $y(k - 1)$ and $y(k - 2)$ from a recurrent perceptron. The attractors show regularity and change the shape according to the change in the input parameters. Further reading on chaos in neural networks can be found in Haykin and Principe (1998).

¹ This q is within the NARMA(p, q) model of a recurrent perceptron, i.e. the order of the feedback.

11.8 Summary

To demonstrate some practical issues and difficulties with modelling and prediction of real-time signals, an analysis of nonlinearity, predictability and performance of nonlinear adaptive algorithms has been undertaken. These issues, especially the detection of nonlinearity, are often neglected by practitioners in the area. The experimental results confirm the superiority of nonlinear architectures and algorithms over the linear ones for modelling a nonlinear signal. Finally, the ability of recurrent structures to exhibit complex dynamical behaviour, such as bifurcations and attractors is briefly demonstrated. In the experiments, recurrent perceptrons are used due to their simplicity and due to the fact that to make a fair comparison with the feedforward and linear structures, a simple recurrent structure ought to be used.

12

Exploiting Inherent Relationships Between Parameters in Recurrent Neural Networks

12.1 Perspective

Optimisation of complex neural network parameters is a rather involved task. It becomes particularly difficult for large-scale networks, such as modular networks, and for networks with complex interconnections, such as feedback networks. Therefore, if an inherent relationship between some of the free parameters of a neural network can be found, which holds at every time instant for a dynamical network, it would help to reduce the number of degrees of freedom in the optimisation task of learning in a particular network.

We derive such relationships between the gain β in the nonlinear activation function of a neuron Φ and the learning rate η of the underlying learning algorithm for both the gradient descent and extended Kalman filter trained recurrent neural networks.

The analysis is then extended in the same spirit for modular neural networks. Both the networks with parallel modules and networks with nested (serial) modules are analysed. A detailed analysis is provided for the latter, since the former can be considered a linear combination of modules that consist of feedforward or recurrent neural networks.

For all these cases, the static and dynamic equivalence between an arbitrary neural network described by β , η and $\mathbf{W}(k)$ and a referent network described by $\beta^R = 1$, η^R and $\mathbf{W}^R(k)$ are derived. A deterministic relationship between these parameters is provided, which allows one degree of freedom less in the nonlinear optimisation task of learning in this framework. This is particularly significant for large-scale networks of any type.

12.2 Introduction

When using neural networks, many of their parameters are chosen empirically. Apart from the choice of topology, architecture and interconnection, the parameters that

influence training time and performance of a neural network are the learning rate η , gain of the activation function β and set of initial weights \mathbf{W}_0 . The optimal values for these parameters are not known *a priori* and generally they depend on external quantities, such as the training data. Other parameters that are also important in this context are

- steepness of the sigmoidal activation function, defined by $\gamma\beta$; and
- dimensionality of the input signal to the network and dimensionality and character of the feedback for recurrent networks.

It has been shown (Thimm and Fiesler 1997a,b) that the distribution of the initial weights has almost no influence on the training time or the generalisation performance of a trained neural network. Hence, we concentrate on the relationship between the parameters of a learning algorithm (η) and those of a nonlinear activation function (β).

To improve performance of a gradient descent trained network, Jacobs (1988) proposed that the acceleration of convergence of learning in neural networks be achieved through the learning rate adaptation. His arguments were that

1. every adjustable learning parameter of the cost function should have its own learning rate parameter; and
2. every learning rate parameter should vary from one iteration to the next.

These arguments are intuitively sound. However, if there is a dependence between some of the parameters in the network, this approach would lead to suboptimal learning and oscillations, since coupled parameters would be trained using different learning rates and different speed of learning, which would deteriorate the performance of the network. To circumvent this problem, some heuristics on the values of the parameters have been derived (Haykin 1994). To shed further light onto this problem and offer feasible solutions, we therefore concentrate on finding relationships between coupled parameters in recurrent neural networks. The derived relationships are also valid for feedforward networks, since recurrent networks degenerate into feedforward networks when the feedback is removed.

Let us consider again a common choice for the activation function,

$$\Phi(\gamma, \beta, x) = \frac{\gamma}{1 + e^{-\beta x}}. \quad (12.1)$$

This is a $\Phi : \mathbb{R} \rightarrow (0, \gamma)$ function. The parameter β is called the *gain* and the product $\gamma\beta$ the *steepness* (slope) of the activation function.¹ The reciprocal of gain is also referred to as the *temperature*. The gain γ of a node in a neural network is a constant that amplifies or attenuates the net input to the node. In Kruschke and Movellan (1991), it has been shown that the use of gradient descent to adjust the gain of the node increases learning speed.

Let us consider again the general gradient-descent-based weight adaptation algorithm, given by

$$\mathbf{W}(k) = \mathbf{W}(k-1) - \eta \nabla_{\mathbf{W}} E(k), \quad (12.2)$$

¹ The gain and steepness are identical for activation functions with $\gamma = 1$. Hence, for such networks, we often use the term slope for β .

where $E(k) = \frac{1}{2}e^2(k)$ is a cost function, $\mathbf{W}(k)$ is the weight vector/matrix at the time-instant k and η is a learning rate. The gradient $\nabla_{\mathbf{W}}E(k)$ in (12.2) comprises the first derivative of the nonlinear activation function (12.1), which is a function of β (Narendra and Parthasarathy 1990). For instance, for a simple nonlinear FIR filter shown in Figure 12.1, the weight update is given by

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta\Phi'(\mathbf{x}^T(k)\mathbf{w}(k))e(k)\mathbf{x}(k). \quad (12.3)$$

For a function $\Phi(\beta, x) = \Phi(\beta x)$, which is the case for the logistic, tanh and arctan nonlinear functions,² Equation (12.3) becomes

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta\beta\Phi'(\mathbf{x}^T(k)\mathbf{w}(k))e(k)\mathbf{x}(k). \quad (12.4)$$

From (12.4), if β increases, so too will the step on the error performance surface for a fixed η . It seems, therefore, advisable to keep β constant, say at unity, and to control the features of the learning process by adjusting the learning rate η , thereby having one degree of freedom less, when all of the parameters in the network are adjustable. Such reduction may be very significant for nonlinear optimisation algorithms employed for parameter adaptation in a particular recurrent neural network.

A fairly general gradient algorithm that continuously adjusts parameters η , β and γ can be expressed by

$$\left. \begin{aligned} y(k) &= \Phi(\mathbf{X}(k), \mathbf{W}(k)), \\ e(k) &= s(k) - y(k), \\ \mathbf{W}(k+1) &= \mathbf{W}(k) - \frac{\eta(k)}{2} \frac{\partial e^2(k)}{\partial \mathbf{W}(k)}, \\ \eta(k+1) &= \eta(k) - \frac{\rho}{2} \frac{\partial e^2(k)}{\partial \eta(k)}, \\ \beta(k+1) &= \beta(k) - \frac{\theta}{2} \frac{\partial e^2(k)}{\partial \beta(k)}, \\ \gamma(k+1) &= \gamma(k) - \frac{\zeta}{2} \frac{\partial e^2(k)}{\partial \gamma(k)}, \end{aligned} \right\} \quad (12.5)$$

where ρ is a small positive constant that controls the adaptive behaviour of the step size sequence $\eta(k)$, whereas small positive constants θ and ζ control the adaptation

² For the logistic function

$$\sigma(\beta, x) = \frac{1}{1 + e^{-\beta x}} = \sigma(\beta x),$$

its first derivative becomes

$$\frac{d\sigma(\beta, x)}{dx} = -\frac{\beta e^{-\beta x}}{(1 + e^{-\beta x})^2},$$

whereas for the tanh function

$$\tanh(\beta, x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}} = \tanh(\beta x),$$

we have

$$\frac{d \tanh(\beta x)}{dx} = \beta \frac{d \tanh(\beta x)}{d\beta x}.$$

The same principle is valid for the Gaussian and inverse tangent activation functions.

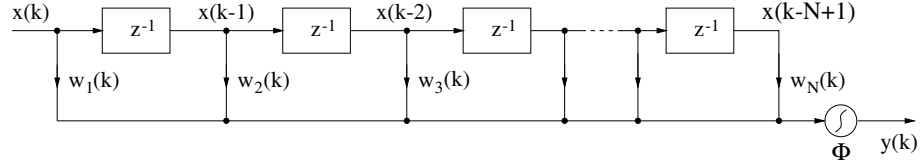


Figure 12.1 A simple nonlinear adaptive filter

of the gain of the activation function β and gain of the node γ , respectively. We will concentrate only on adaptation of β and η .

The selection of learning rate η is critical for the gradient descent algorithms (Mathews and Xie 1993). An η that is small as compared to the reciprocal of the input signal power will ensure small misadjustment in the steady state, but the algorithm will converge slowly. A relatively large η , on the other hand, will provide faster convergence at the cost of worse misadjustment and steady-state characteristics. Therefore, an ideal choice would be an adjustable η which would be relatively large in the beginning of adaptation and become gradually smaller when approaching the global minimum of the error performance surface (optimal values of weights).

We illustrate the above ideas on the example of a simple nonlinear FIR filter, shown in Figure 12.1, for which the output is given by

$$y(k) = \Phi(\mathbf{x}^T(k)\mathbf{w}(k)). \quad (12.6)$$

We can continually adapt the step size using a gradient descent algorithm so as to reduce the squared estimation error at each time instant. Extending the approach from Mathews and Xie (1993) to the nonlinear case, we obtain

$$\left. \begin{aligned} e(k) &= s(k) - \Phi(\mathbf{x}^T(k)\mathbf{w}(k)), \\ \mathbf{w}(k) &= \mathbf{w}(k-1) + \eta(k-1)e(k-1)\Phi'(k-1)\mathbf{x}(k-1), \\ \eta(k) &= \eta(k-1) - \frac{\rho}{2} \frac{\partial}{\partial \eta(k-1)} e^2(k) \\ &= \eta(k-1) - \frac{\rho}{2} \frac{\partial^T e^2(k)}{\partial \mathbf{w}(k)} \frac{\partial \mathbf{w}(k)}{\partial \eta(k-1)} \\ &= \eta(k-1) + \rho e(k)e(k-1)\Phi'(k)\Phi'(k-1)\mathbf{x}^T(k-1)\mathbf{x}(k), \end{aligned} \right\} \quad (12.7)$$

where $\Phi'(k) = \Phi'(\mathbf{x}^T(k)\mathbf{w}(k))$, $\Phi'(k-1) = \Phi'(\mathbf{x}^T(k-1)\mathbf{w}(k-1))$ and ρ is a small positive constant that controls the adaptive behaviour of the step size sequence $\eta(k)$.

If we adapt the step size for each weight individually, we have

$$\eta_i(k) = \eta_i(k-1) + \rho e(k)e(k-1)\Phi'(k)\Phi'(k-1)x_i(k)x_i(k-1), \quad i = 1, \dots, N, \quad (12.8)$$

and

$$w_i(k+1) = w_i(k) + \eta_i(k)e(k)x_i(k), \quad i = 1, \dots, N. \quad (12.9)$$

These expressions become much more complicated for large and recurrent networks.

As an alternative to the continual learning rate adaptation, we might consider continual adaptation of the gain of the activation function $\Phi(\beta x)$. The gradient descent

algorithm that would update the adaptive gain can be expressed as

$$\left. \begin{aligned} e(k) &= s(k) - \Phi(\mathbf{w}^T(k)\mathbf{x}(k)), \\ \mathbf{w}(k) &= \mathbf{w}(k-1) + \eta(k-1)e(k-1)\Phi'(k-1)\mathbf{x}(k-1), \\ \beta(k) &= \beta(k-1) - \frac{\theta}{2} \frac{\partial}{\partial \beta(k-1)} e^2(k) \\ &= \beta(k-1) - \frac{\theta}{2} \frac{\partial^T e^2(k)}{\partial \mathbf{w}(k)} \frac{\partial \mathbf{w}(k)}{\partial \beta(k-1)} \\ &= \beta(k-1) + \theta \eta(k-1)e(k)e(k-1)\Phi'(k)\Phi''_{\beta}(k-1)\mathbf{x}^T(k-1)\mathbf{x}(k). \end{aligned} \right\} \quad (12.10)$$

For the adaptation of $\beta(k)$ there is a need to calculate the second derivative of the activation function, which is rather computationally involved. Such an adaptive gain algorithm was, for instance, analysed in Birkett and Goubran (1997). The proposed function was

$$\sigma(x, a) = \begin{cases} x, & |x| \leq a, \\ \text{sgn}(x) \left[(1-a) \tanh\left(\frac{|x-a|}{1-a}\right) + a \right], & |x| > a, \end{cases} \quad (12.11)$$

where x is the input signal and a defines the adaptive linear region of the sigmoid. This activation function is shown in Figure 12.2. Parameter a is updated according to the stochastic gradient rule. The benefit of this algorithm is that the slope and region of linearity of the activation function can be adjusted. Although this and similar approaches are an alternative to the learning rate adaptation, researchers have not taken into account that parameters β and η might be coupled. If a relationship between them can be derived, then we choose adaptation of the parameter that is less computationally expensive to adapt and less sensitive to adaptation errors. As shown above, adaptation of the gain β is far more computationally expensive than adaptation of η . Hence, there is a need to mathematically express the dependence between the two and reduce the computational load for training neural networks.

Thimm *et al.* (1996) provided the relationship between the gain β of the logistic activation function,

$$\Phi(\beta, x) = \frac{1}{1 + e^{-\beta x}}, \quad (12.12)$$

and the learning rate η for a class of general feedforward neural networks trained by backpropagation. They prove that changing the gain of the activation function is equivalent to simultaneously changing the learning rate and the weights. This simplifies the backpropagation learning rule by eliminating one of its parameters (Thimm *et al.* 1996). This concept has been successfully applied to compensate for the non-standard gain of optical sigmoids for optical neural networks. Relationships between η and β for recurrent and modular networks were derived by Mandic and Chambers (1999a,e).

Basic modular architectures are the parallel and serial architecture. Parallel architectures provide linear combinations of neural network modules, and learning algorithms for them are based upon minimising the linear combination of the output

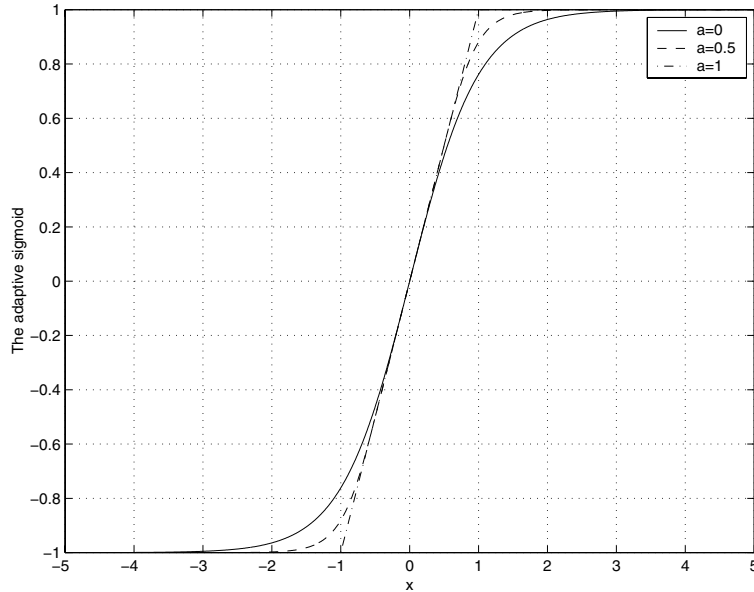


Figure 12.2 An adaptive sigmoid

errors of particular modules. Hence, the algorithms for training such networks are extensions of standard algorithms designed for single modules. Serial (nested) modular architectures are more complicated, an example of which is a pipelined recurrent neural network (PRNN). This is an emerging architecture used in nonlinear time series prediction (Haykin and Li 1995; Mandic and Chambers 1999f). It consists of a number of nested small-scale recurrent neural networks as its modules, which means that a learning algorithm for such a complex network has to perform a nonlinear optimisation task on a number of parameters. We look at relationships between the learning rate and the gain of the activation function for this architecture and for various learning algorithms.

12.3 Overview

A relationship between the learning rate η in the learning algorithm and the gain β in the nonlinear activation function, for a class of recurrent neural networks (RNNs) trained by the real-time recurrent learning (RTRL) algorithm is provided. It is shown that an arbitrary RNN can be obtained via the referent RNN, with some deterministic rules imposed on its weights and the learning rate. Such relationships reduce the number of degrees of freedom when solving the nonlinear optimisation task of finding the optimal RNN parameters. This analysis is further extended for modular neural architectures.

We define the conditions of static and dynamic equivalence between a referent network, with $\beta = 1$, and an arbitrary network with an arbitrary β . Since the dynamic equivalence is dependent on the chosen learning algorithm, the relationships are provided for a variety of both the gradient descent (GD) and the extended recursive

least-squares (ERLS) classes of learning algorithms and a general nonlinear activation function of a neuron.

By continuity, the derived results are also valid for feedforward networks and their linear and nonlinear combinations.

12.4 Static and Dynamic Equivalence of Two Topologically Identical RNNs

As the aim is to eliminate either the gain β or the learning rate η from the paradigm of optimisation of the RNN parameters, it is necessary to derive the relationship between a network with arbitrarily chosen parameters β and η and the referent network, so as the outputs of the networks are identical for every time instant. An obvious choice for the referent network is the network with the gain of the activation function $\beta = 1$. Let us therefore denote all the entries in the referent network, which are different from those of an arbitrary network, with the superscript ‘R’ joined to a particular variable, i.e. $\beta^R = 1$.

For two networks to be equivalent, it is necessary that their outputs are identical and that this is valid for both the trained network and while on the run, i.e. while tracking some dynamical process. We therefore differentiate between the equivalence in the static and dynamic sense. We define the static and dynamic equivalence between two networks below.

Definition 12.4.1. By static equivalence, we consider the equivalence of the outputs of an arbitrary network and the referent network with fixed weights, for a given input vector $\mathbf{u}(k)$, at a fixed time instant k .

Definition 12.4.2. By dynamic equivalence, we consider the equivalence of the outputs between an arbitrary network and the referent network for a given input vector $\mathbf{u}(k)$, with respect to the learning algorithm, while the networks are running.

The static equivalence is considered for already trained networks, whereas both static and dynamic equivalence are considered for networks being adapted on the run. We can think of the static equivalence as an analogue to the forward pass in computation of the outputs of a neural network, whereas the dynamic equivalence can be thought of in terms of backward pass, i.e. weight update process. We next derive the conditions for either case.

12.4.1 Static Equivalence of Two Isomorphic RNNs

In order to establish the static equivalence between an arbitrary and referent RNN, the outputs of their neurons must be the same, i.e.

$$y_n(k) = y_n^R(k) \quad \Leftrightarrow \quad \Phi(\mathbf{u}_n^T(k)\mathbf{w}_n(k)) = \Phi^R(\mathbf{u}_n^T(k)\mathbf{w}_n^R(k)), \quad (12.13)$$

where the index ‘ n ’ runs over all neurons in an RNN, and $\mathbf{w}_n(k)$ and $\mathbf{u}_n(k)$ are, respectively, the set of weights and the set of inputs which belong to the neuron n . For a general nonlinear activation function, we have

$$\Phi(\beta, \mathbf{w}_n, \mathbf{u}_n) = \Phi(1, \mathbf{w}_n^R, \mathbf{u}_n) \quad \Leftrightarrow \quad \beta\mathbf{w}_n = \mathbf{w}_n^R. \quad (12.14)$$

To illustrate this, consider, for instance, the logistic nonlinearity, given by

$$\frac{1}{1 + e^{-\beta \mathbf{u}_n^T \mathbf{w}_n}} = \frac{1}{1 + e^{-\mathbf{u}_n^T \mathbf{w}_n^R}} \Leftrightarrow \beta \mathbf{w}_n = \mathbf{w}_n^R, \quad (12.15)$$

where the time index (k) is neglected, since all the vectors above are constant during the calculation of the output values. As the equality (12.14) should be valid for every neuron in the RNN, it is therefore valid for the complete weight matrix \mathbf{W} of the RNN.

The essence of the above analysis is given in the following lemma, which is independent of the underlying learning algorithm for the RNN, which makes it valid for two isomorphic³ RNNs of any topology and architecture.

Lemma 12.4.3 (see Mandic and Chambers 1999e). *For a recurrent neural network, with weight matrix \mathbf{W} and gain of the activation function β , to be equivalent in the static sense to the referent network, characterised by \mathbf{W}^R and $\beta^R = 1$, with the same topology and architecture (isomorphic), the following condition must be satisfied:*

$$\beta \mathbf{W}(k) = \mathbf{W}^R(k). \quad (12.16)$$

12.4.2 Dynamic Equivalence of Two Isomorphic RNNs

The equivalence of two RNNs, includes both the static equivalence and dynamic equivalence. As in the learning process (12.2), the learning rate η is multiplied by the gradient of the cost function, we shall investigate the role of β in the gradient of the cost function for the RNN. We are interested in a general class of nonlinear activation functions where

$$\frac{\partial \Phi(\beta, x)}{\partial x} = \frac{\partial \Phi(\beta x)}{\partial(\beta x)} \frac{\partial(\beta x)}{\partial x} = \beta \frac{\partial \Phi(\beta x)}{\partial(\beta x)} = \beta \frac{\partial \Phi(1, x)}{\partial x}. \quad (12.17)$$

In our case, it becomes

$$\Phi'(\beta, \mathbf{w}, \mathbf{u}) = \beta \Phi'(1, \mathbf{w}^R, \mathbf{u}). \quad (12.18)$$

Indeed, for a simple logistic function (12.12), we have

$$\Phi'(x) = \frac{\beta e^{-\beta x}}{(1 + e^{-\beta x})^2} = \beta \Phi'(x^R),$$

where $x^R = \beta x$ denotes the argument of the referent logistic function (with $\beta^R = 1$), so that the network considered is equivalent in the static sense to the referent network. The results (12.17) and (12.18) mean that wherever Φ' occurs in the dynamical equation of the RTRL-based learning process, the first derivative (or gradient when applied to all the elements of the weight matrix \mathbf{W}) of the referent function equivalent in the static sense to the one considered, becomes multiplied by the gain β .

The following theorem provides both the static and dynamic interchangeability of the gain in the activation function β and the learning rate η for the RNNs trained by the RTRL algorithm.

³ Isomorphic networks have identical topology, architecture and interconnections.

Theorem 12.4.4 (see Mandic and Chambers 1999e). *For a recurrent neural network, with weight matrix \mathbf{W} , gain of the activation function β and learning rate in the RTRL algorithm η , to be equivalent in the dynamic sense to the referent network, characterised by \mathbf{W}^R , $\beta^R = 1$ and η^R , with the same topology and architecture (isomorphic), the following conditions must hold.*

- (i) *The networks are equivalent in the static sense, i.e.*

$$\mathbf{W}^R(k) = \beta\mathbf{W}(k). \quad (12.19)$$

- (ii) *The learning rate η of the network considered and the learning rate η^R of the referent network are related by*

$$\eta^R = \beta^2\eta. \quad (12.20)$$

Proof. From the equivalence in the static sense, the weight update equation for the referent network can be written as

$$\mathbf{W}^R(k) = \mathbf{W}^R(k-1) + \beta\Delta\mathbf{W}(k), \quad (12.21)$$

which gives

$$\Delta\mathbf{W}^R(k) = \beta\Delta\mathbf{W}(k) = \beta\left(\eta e(k) \frac{\partial y_1(k)}{\partial \mathbf{W}(k)}\right) = \eta\beta e(k) \mathbf{II}_1(k), \quad (12.22)$$

where $\mathbf{II}_1(k)$ is the matrix with elements $\pi_{n,l}^1(k)$.

Now, in order to derive the conditions of dynamical equivalence between an arbitrary and the referent RNN, the relationship between the appropriate matrices $\mathbf{II}_1(k)$ and $\mathbf{II}_1^R(k)$ must be established. That implies that, for all the neurons in the RNN, the matrix $\mathbf{II}(k)$, which comprises all the terms $\partial y_j / \partial w_{n,l}$, $\forall w_{n,l} \in \mathbf{W}$, $j = 1, 2, \dots, N$, must be interrelated to the appropriate matrix $\mathbf{II}^R(k)$, which represents the referent network.

We shall prove this relationship by induction. For convenience, let us denote $\text{net}(k) = \mathbf{u}^T(k)\mathbf{w}(k)$ and $\text{net}^R(k) = \mathbf{u}^T(k)\mathbf{w}^R(k)$.

Given:

$$\begin{aligned} \mathbf{W}^R(k) &= \beta\mathbf{W}(k) && \text{(static equivalence),} \\ \Phi'(\text{net}^R(k)) &= \frac{1}{\beta}\Phi'(\text{net}(k)) && \text{(activation function derivative),} \\ y_j^R(k) &= \Phi(\text{net}^R(k)) \\ &= \Phi(\text{net}(k)) \\ &= y_j(k), \quad j = 1, \dots, N && \text{(activation).} \end{aligned}$$

Induction base: the recursion (D.11) starts as

$$\begin{aligned} (\pi_{n,l}^j(k=1))^{\text{R}} &= \Phi'(\text{net}^{\text{R}}(k)) \left[\sum_{m=1}^N w_{j,m+p+1}^{\text{R}}(k=0) \pi_{n,l}^m(k=0) + \delta_{nj} u_l(k=0) \right] \\ &= \frac{1}{\beta} \Phi'(\text{net}(k)) \delta_{nj} u_l(k=0) \\ &= \frac{1}{\beta} \pi_{n,l}^j(k=1), \end{aligned}$$

which gives $\mathbf{\Pi}^{\text{R}}(k=1) = (1/\beta) \mathbf{\Pi}(k=1)$.

Induction step:

$$(\pi_{n,l}^j(k))^{\text{R}} = \frac{1}{\beta} \pi_{n,l}^j(k) \quad \text{and} \quad \mathbf{\Pi}^{\text{R}}(k) = \frac{1}{\beta} \mathbf{\Pi}(k) \quad (\text{assumption}).$$

Now, for the $(k+1)$ st step we have

$$\begin{aligned} (\pi_{n,l}^j(k+1))^{\text{R}} &= \Phi'(\text{net}^{\text{R}}) \left[\sum_{m=1}^N w_{j,m+p+1}^{\text{R}}(k) \pi_{n,l}^m(k) + \delta_{nj} u_l(k) \right] \\ &= \frac{1}{\beta} \Phi'(\text{net}) \left[\sum_{m=1}^N \beta w_{j,m+p+1}(k) \frac{1}{\beta} \pi_{n,l}^m(k) + \delta_{nj} u_l(k) \right] \\ &= \frac{1}{\beta} \pi_{n,l}^j(k+1), \end{aligned}$$

which means that

$$\mathbf{\Pi}^{\text{R}}(k+1) = \frac{1}{\beta} \mathbf{\Pi}(k+1).$$

Based upon the established relationship, the learning process for the referent RNN can be expressed as

$$\Delta \mathbf{W}^{\text{R}}(k) = \beta \Delta \mathbf{W}(k) = \beta \eta e(k) \mathbf{\Pi}_1(k) = \beta^2 \eta e(k) \mathbf{\Pi}_1^{\text{R}}(k) = \eta^{\text{R}} e(k) \mathbf{\Pi}_1^{\text{R}}(k). \quad (12.23)$$

Hence, the referent network with the learning rate $\eta^{\text{R}} = \beta^2 \eta$ and gain $\beta^{\text{R}} = 1$ is equivalent in the dynamic sense, with respect to the RTRL algorithm, to an arbitrary RNN with gain β and learning rate η . \square

12.5 Extension to a General RTRL Trained RNN

It is now straightforward to show that the conditions for static and dynamic equivalence derived so far are valid for a general recurrent neural network trained by a gradient algorithm. For instance, for a general RTRL trained RNN, the cost function comprises squared error terms over all the output neurons, i.e.

$$E(k) = \frac{1}{2} \sum_{j \in \mathcal{C}} e_j^2(k), \quad (12.24)$$

where \mathcal{C} denotes the set of those neurons whose outputs are included in the cost function. The crucial equations for the static and dynamic equivalence remain intact. It should, however, be noted that the weight update equation in the RTRL algorithm (12.23) now becomes

$$\begin{aligned}\Delta \mathbf{W}^{\text{R}}(k) &= \beta \Delta \mathbf{W}(k) = \beta \eta \sum_{i \in \mathcal{C}} e_i(k) \mathbf{\Pi}(k) \\ &= \beta^2 \eta \sum_{i \in \mathcal{C}} e_i(k) \mathbf{\Pi}^{\text{R}}(k) = \eta^{\text{R}} \sum_{i \in \mathcal{C}} e_i(k) \mathbf{\Pi}^{\text{R}}(k),\end{aligned}\quad (12.25)$$

which is structurally the same as the one used in the proof of Theorem 12.4.4. Hence the relationships between β and η derived for a single-output neuron RNN are valid for a general RNN with any number of output neurons.

12.6 Extension to Other Commonly Used Activation Functions

Other frequently used nonlinear activation functions of a neuron, such as the hyperbolic tangent $\Phi(x) = \tanh(\beta x)$ and inverse tangent $\Phi(x) = \arctan(\beta x)$, have the same functional dependence between β , x and their derivatives as the logistic function. Since for the dynamic equivalence of two isomorphic networks, it is crucial to consider the first derivative of the underlying activation function, let us recall that for all of the nonlinear activation functions mentioned, the first derivative of a function with an arbitrary gain can be connected to the derivative with respect to variable x as

$$\beta \frac{d\Phi(\beta, x)}{d(\beta x)} = \frac{d\Phi(\beta, x)}{dx}, \quad (12.26)$$

which means that the results derived for the logistic activation function are also valid for a large class of nonlinear activation functions with the first derivative that satisfies (12.26). This can be formally expressed as follows (Mandic and Krcmar 2000).

Corollary 12.6.1. *Theorem 12.4.4 holds for every differentiable nonlinear activation function with the first derivative that satisfies*

$$\beta \frac{d\Phi(\beta, x)}{d(\beta x)} = \frac{d\Phi(\beta, x)}{dx}. \quad (12.27)$$

12.7 Extension to Other Commonly Used Learning Algorithms for Recurrent Neural Networks

After having derived the relationship between the learning rate η and the gain in the activation function β for the RTRL trained recurrent neural network of an arbitrary size, and knowing that the dynamical equivalence of an arbitrary and the referent network is highly dependent on the learning algorithm chosen, let us consider two other frequently used learning algorithms for a general RNN, namely the backpropagation through time (BPTT) algorithm and the recurrent backpropagation (RBP) algorithm.

In both cases, the derivation of the conditions of the static equivalence between an arbitrary and the referent network follow the same spirit as for the RTRL trained RNN, and will be omitted. Moreover, the analysis given in Section 12.6, holds for these algorithms, and we will therefore consider the whole class of functions for which (12.26) holds, without going into detail for a particular function belonging to the class.

12.7.1 Relationships Between β and η for the Backpropagation Through Time Algorithm

The backpropagation through time algorithm for training a recurrent neural network stems from the common backpropagation algorithm. It may be derived by unfolding the temporal operation of the network into a multilayer feedforward network, the topology of which grows by one such block at every time step (Haykin 1994; Werbos 1990). As our intention is to show the conditions of dynamical equivalence between an arbitrary and referent RNN, with respect to the parameters β and η , we will concentrate only on the points that are specific to BPTT with regards to the RTRL algorithm. Therefore, the correction to the element $w_{i,j}(k)$ of the weight matrix \mathbf{W} for the BPTT trained RNN can be expressed as

$$\Delta w_{i,j}(k) = -\eta \sum_{k=k_0+1}^{k_1} \delta_j(k) x_i(k-1), \quad (12.28)$$

where the BPTT is performed back to time $(k_0 + 1)$ and the gradients δ are calculated as (Haykin 1994; Werbos 1990)

$$\delta_j(k) = \begin{cases} \Phi'(v_j(k)) e_j(k), & k = k_1, \\ \Phi'(v_j(k)) \left[e_j(k) + \sum_{i \in \mathcal{A}} w_{i,j} \delta_i(k-1) \right], & k_0 < k < k_1. \end{cases} \quad (12.29)$$

The symbol \mathcal{A} denotes the set of output neurons and $e_j(k)$, $j \in \mathcal{A}$, are the corresponding errors. The first line in (12.29) is structurally the same as in the standard backpropagation algorithm, whereas in the second line in (12.29), there is a sum of product terms $w_{i,j} \delta_i(k-1)$, which is different from the backpropagation algorithm, since it comprises the unfolded feedback terms. Examining these product terms, which are structurally the same as appropriate terms in (D.11), it follows that (Mandic and Chambers 1999a)

$$w_{i,j} \delta_i = w_{i,j}^R \delta_i^R. \quad (12.30)$$

Now, from the results obtained for the RTRL trained RNN and relationships for feedforward networks trained by backpropagation (Thimm *et al.* 1996), the conditions of static and dynamic equivalence of an arbitrary and the referent BPTT trained RNN are expressed in the following corollary.

Corollary 12.7.1. *For a recurrent neural network, with weight matrix \mathbf{W} , gain in the activation function β and learning rate in the BPTT algorithm η , to be equivalent to the referent network, characterised by \mathbf{W}^R , $\beta^R = 1$ and η^R , with the same topology and architecture (isomorphic), the following conditions must hold.*

(i) *The networks are equivalent in the static sense, i.e.*

$$\mathbf{W}^R(k) = \beta \mathbf{W}(k). \quad (12.31)$$

(ii) *The learning rate η of the network considered and the learning rate η^R of the equivalent referent network are related by*

$$\eta^R = \beta^2 \eta. \quad (12.32)$$

As seen from Corollary 12.7.1, the conditions for the static and dynamic equivalence of an arbitrary and the referent RNN trained by the BPTT algorithm are the same as for the RTRL trained RNN.

12.7.2 Results for the Recurrent Backpropagation Algorithm

Without going deeply into the core of the RBP algorithm (Haykin 1994; Pineda 1989), let us only consider the key relation for the gradient calculation, which can be expressed as

$$\delta_n(k) = \Phi' \left(\sum_i \delta_i(k) w_{i,n}(k) + e_n(k) \right), \quad n = 1, \dots, N, \quad (12.33)$$

which is structurally the same as the appropriate term for the RTRL algorithm. Therefore, we can express the conditions of the static and dynamic equivalence of an arbitrary and the referent RBP trained RNN, as follows.

Corollary 12.7.2. *The conditions of static and dynamic equivalence of an arbitrary and the referent RNN trained by the recursive backpropagation algorithm are the same as for the RTRL and BPTT trained networks.*

By continuity, the analysis provided so far is valid for an RNN of any size, number of hidden layers and character of feedback.

12.7.3 Results for Algorithms with a Momentum Term

A momentum term can be introduced into the update equation of gradient learning algorithms as (Haykin 1994)

$$\Delta \mathbf{W}(k) = \alpha \Delta \mathbf{W}(k-1) - \eta \nabla_{\mathbf{W}(k-1)} E(k), \quad (12.34)$$

where α is a momentum constant. This introduces an infinite impulse response element into the learning rule, which in some cases helps to increase the rate of convergence of a gradient algorithm (An *et al.* 1994). To preserve stability of (12.34), $0 < \alpha < 1$.

As (12.34) is linear in \mathbf{W} , the analysis for the plain gradient algorithm for neural networks, applies also for the momentum algorithm, as shown in Mandic and Krmar (2000).

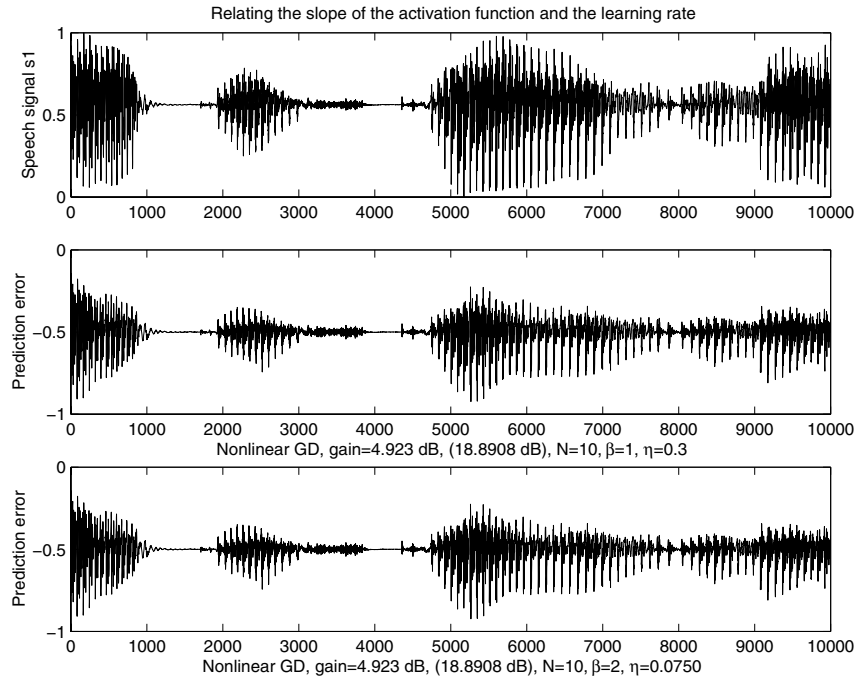


Figure 12.3 Verification of the β - η - W relationships based upon the prediction of the speech signal s_1

Corollary 12.7.3. *The conditions of static and dynamic equivalence of an arbitrary and the referent recurrent neural network trained by a gradient descent algorithm with a momentum factor are the same as for the previously analysed algorithms, providing*

$$\alpha^R = \alpha, \quad (12.35)$$

where α^R is the momentum constant of a referent network, whereas α is the momentum constant of an arbitrary network.

12.8 Simulation Results

To support the analysis, simulation results that illustrate the derived relationships are provided. A tap delay line of length $N = 10$ was used to feed a neuron that performed prediction, trained by a direct gradient descent algorithm.⁴ The input signals were two speech signals denoted by s_1 and s_2 . Firstly, the data were fed into a referent network with $\beta = 1$, the network was adapted and the outputs were recorded. Afterwards the learning rate of an arbitrary network was adjusted according to the β - η relationship and the whole procedure was repeated. The results of simulations for signal s_1 are shown in Figure 12.3, whereas Figure 12.4 shows the results of simulations for signal

⁴ The aim was not to build a good predictor, but to demonstrate the validity of the derived results, hence the quality of the performance was not important for this experiment.

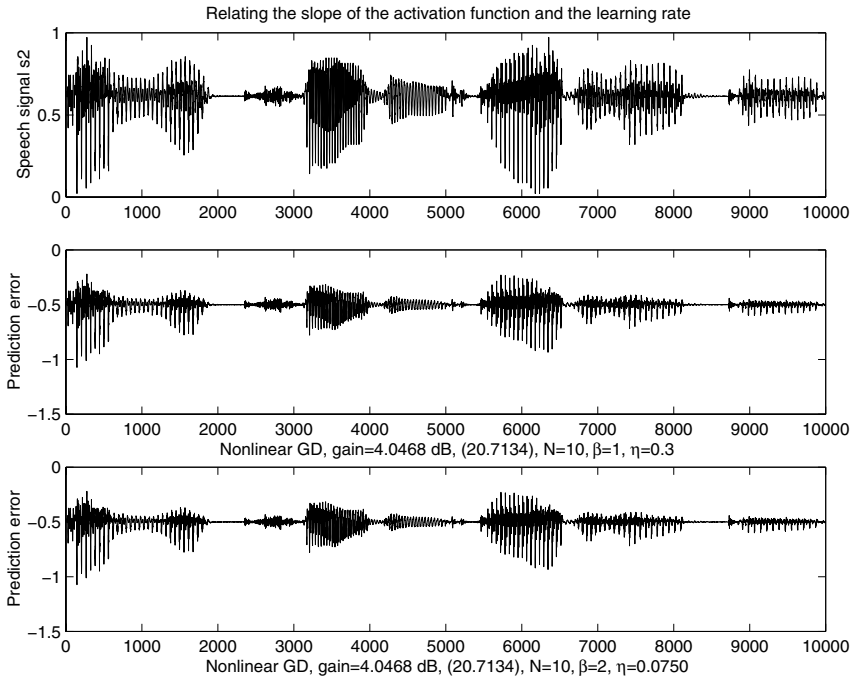


Figure 12.4 Verification of the β - η - \mathbf{W} relationships based upon the prediction of the speech signal s_2

s_2 . It was first ensured that the conditions of static equivalence $\mathbf{W}^R(k) = \beta\mathbf{W}(k)$ were preserved. In both Figures 12.3 and 12.4, the top graph shows the input signal to the adaptive neuron. The middle graph shows the output prediction error for the referent network with $\beta = 1$, whereas the bottom graph shows the output prediction error for an arbitrary network. For the gain $\beta = 2$ for the arbitrary network used to generate the results in Figure 12.3, after the conditions of static and dynamic equivalence were satisfied, the learning rate became $\eta = \eta^R/\beta^2 = 0.3/4 = 0.075$, and likewise for the situation shown in Figure 12.4. The output prediction error and the ratio between the signal and error variance (and the signal and error power in brackets) in decibels were identical for the referent and arbitrary network, as expected.

12.9 Summary of Relationships Between β and η for General Recurrent Neural Networks

The relationship between the gain β in a general activation function of a neuron and the step size η in the RTRL-based training of a general RNN is provided. Both static and dynamic equivalence of an arbitrary RNN and the referent network, with respect to β and η have been provided. As the conditions of dynamical equivalence are dependent on the choice of the underlying learning algorithm, the same analysis has been undertaken for two frequently used learning algorithms for training RNNs, namely the BPTT and the RBP algorithms. It has been shown that a general RNN

can be replaced with the referent isomorphic RNN, with gain $\beta^R = 1$ and modified learning rate η^R , hence providing one degree of freedom less in a nonlinear optimisation paradigm of training RNNs. Although the analysis has been undertaken on the example of the logistic nonlinearity, it is shown that it holds for a large class of $C^1(-\infty, +\infty)$ functions with the first derivative which satisfies

$$\beta \frac{d\Phi(\beta, x)}{d(\beta x)} = \frac{\Phi(\beta, x)}{dx}.$$

12.10 Relationship Between η and β for Modular Neural Networks: Perspective

We next provide the relationship between the learning rate and the gain of a nonlinear activation function of a neuron within the framework of modular neural networks. This leads to reduction in the computational complexity of learning algorithms which continuously adapt the weights of a modular network because there is a smaller number of independent parameters to optimise. Although the analysis is provided for networks with recurrent modules, and particular algorithms are analysed for nested modular recurrent networks, the results obtained are valid for any modular networks trained by the algorithms under consideration.

Based upon results for ordinary recurrent neural networks, we extend our analysis to modular neural networks and provide the static and dynamic equivalence between an arbitrary modular network described by β , η and $\mathbf{W}(k)$ and a referent modular network described by $\beta^R = 1$, η^R and $\mathbf{W}^R(k)$. We show that there is a deterministic relationship between them, which allows one degree of freedom less in the nonlinear optimisation task of learning in such a framework. The relationships are provided for both the gradient descent (GD) and the extended recursive least-squares (ERLS) learning algorithms (Baltersee and Chambers 1998; Mandic *et al.* 1998), and a general nonlinear activation function of a neuron.

12.11 Static Equivalence Between an Arbitrary and a Referent Modular Neural Network

By static equivalence between two modular neural networks, we consider the equivalence between the outputs of the neurons in an arbitrary modular network and the outputs of the corresponding neurons in a referent modular network (Mandic and Chambers 1999a), for a fixed time-instant k , i.e.

$$\begin{aligned} y_{i,n}(k) = y_{i,n}^R(k) &\Leftrightarrow \Phi(\beta, \mathbf{w}_{i,n}(k), \mathbf{u}_{i,n}(k)) \\ &= \Phi(1, \mathbf{w}_{i,n}^R(k), \mathbf{u}_{i,n}(k)) \Leftrightarrow \beta \mathbf{w}_{i,n}(k) = \mathbf{w}_{i,n}^R(k), \end{aligned} \quad (12.36)$$

where $\mathbf{w}_{i,n}(k)$ and $\mathbf{u}_{i,n}(k)$ are, respectively, the set of weights and the set of inputs which belong to the neuron n , $n = 1, \dots, N$, in module i , $i = 1, \dots, M$. This is valid irrespectively whether the modules are parallel or nested. For the i th neuron in the n th module and the logistic nonlinearity we have

$$\frac{1}{1 + e^{-\beta \mathbf{u}_{i,n}^T \mathbf{w}_{i,n}}} = \frac{1}{1 + e^{-\mathbf{u}_{i,n}^T \mathbf{w}_{i,n}^R}} \Leftrightarrow \beta \mathbf{w}_{i,n} = \mathbf{w}_{i,n}^R, \quad (12.37)$$

where the time index k is neglected and illustrates the extension of the previous analysis result for a general modular neural network.

The following lemma, which is independent of the underlying learning algorithm, comprises the above analysis.

Lemma 12.11.1 (see Mandic and Chambers 1999a). *An arbitrary modular network with weight matrix $\mathbf{W}(k)$, and a gain in the activation function β , is equivalent in the static sense to the isomorphic referent modular network, characterised by $\mathbf{W}^R(k)$ and $\beta^R = 1$, if*

$$\beta\mathbf{W}(k) = \mathbf{W}^R(k). \quad (12.38)$$

12.12 Dynamic Equivalence Between an Arbitrary and a Referent Modular Network

While by static equivalence between two networks we consider the calculation of the outputs of the networks, for a given weight matrix and input vector, by dynamic equivalence, we consider the equality between two networks in the sense of adaptation of the weights. Hence, it is dependent on the underlying learning algorithm. Since we have to concentrate on a particular algorithm for a particular neural network, it is convenient to use the already described pipelined recurrent neural network. The overall cost function of the PRNN is given by⁵ (Haykin and Li 1995)

$$E(k) = \frac{1}{2} \sum_{i=1}^M \lambda^{i-1} e_i^2(k), \quad (12.39)$$

where $e_i(k)$ is the instantaneous output error from module i and a weighting factor $\lambda \in (0, 1]$, is introduced which determines the weighting of the individual modules (Haykin and Li 1995). Obviously, this function introduces some sense of forgetting for nested modules and importance for parallel modules. The following analysis is hence independent of the chosen configuration.

For a general class of nonlinear activation functions, the weight updating includes the first derivative of a nonlinear activation function of a neuron, where

$$\Phi'(\beta, \mathbf{w}, \mathbf{u}) = \beta\Phi'(1, \mathbf{w}^R, \mathbf{u}). \quad (12.40)$$

For the logistic function, for instance, we have

$$\Phi'(x) = \frac{\beta e^{-\beta x}}{(1 + e^{-\beta x})^2} = \beta\Phi'(x^R), \quad (12.41)$$

where $x^R = \beta x$ denotes the argument of the referent logistic function (with $\beta^R = 1$), so that the network considered is equivalent in the static sense to the referent network.

⁵ The same form of the cost function is used for parallel modular networks. If the modules are of equal importance, then $\lambda = 1/M$.

12.12.1 *Dynamic Equivalence for a GD Learning Algorithm*

Since recurrent modules degenerate into feedforward ones when the feedback is removed, for generality, we provide the analysis for recurrent modules. All the results so derived are also valid for the feedforward modules. From the equivalence in the static sense, the weight update equation for the referent network, can be written as

$$\mathbf{W}^R(k) = \mathbf{W}^R(k-1) + \beta \Delta \mathbf{W}(k), \quad (12.42)$$

which gives

$$\Delta \mathbf{W}^R(k) = \beta \Delta \mathbf{W}(k) = \beta \left(\eta \sum_{i=1}^M e_i(k) \frac{\partial y_{i,1}(k)}{\partial \mathbf{W}(k)} \right) = \eta \beta \sum_{i=1}^M e_i(k) \mathbf{\Pi}_{i,1}(k), \quad (12.43)$$

where $\mathbf{\Pi}_{i,1}(k)$ is the matrix with elements $\pi_{n,l}^{i,1}(k)$ (D.23)–(D.30).

The matrix $\mathbf{\Pi}(k)$, which comprises all the terms

$$\frac{\partial y_{i,j}}{\partial w_{n,l}}, \quad \forall w_{n,l} \in \mathbf{W}, \quad i = 1, \dots, M, \quad j = 1, \dots, N,$$

must be related to the appropriate matrix $\mathbf{\Pi}^R(k)$, which represents the gradients of the referent network. Thus, we have

$$\begin{aligned} (\pi_{n,l}^{i,j}(k+1))^R &= \Phi'(v_{i,j}^R(k)) \left[\sum_{m=1}^N w_{j,m+p+1}^R(k) \pi_{n,l}^{i,m}(k) + \delta_{nj} u_{i,l}(k) \right] \\ &= \frac{1}{\beta} \Phi'(v_{i,j}(k)) \left[\sum_{m=1}^N \beta w_{j,m+p+1}(k) \frac{1}{\beta} \pi_{n,l}^{i,m}(k) + \delta_{nj} u_{i,l}(k) \right] \\ &= \frac{1}{\beta} \pi_{n,l}^{i,j}(k+1), \end{aligned} \quad (12.44)$$

which gives

$$\mathbf{\Pi}^R(k+1) = \frac{1}{\beta} \mathbf{\Pi}(k+1). \quad (12.45)$$

The weight adaptation process for the referent PRNN can be now expressed as

$$\begin{aligned} \Delta \mathbf{W}^R(k) &= \beta \Delta \mathbf{W}(k) = \beta \eta \sum_{i=1}^M e_i(k) \mathbf{\Pi}_{i,1}(k) \\ &= \beta^2 \eta \sum_{i=1}^M e_i(k) \mathbf{\Pi}_{i,1}^R(k) = \eta^R \sum_{i=1}^M e_i(k) \mathbf{\Pi}_{i,1}^R(k), \end{aligned} \quad (12.46)$$

which gives the required dynamic relationship and is encompassed in the following lemma.

Lemma 12.12.1 (see Mandic and Chambers 1999a). *An arbitrary modular neural network represented by β , η and $\mathbf{W}(k)$ is equivalent in the dynamic sense*

in terms of gradient-descent-based learning to a referent isomorphic modular neural network represented by $\beta^R = 1$, η^R and $\mathbf{W}^R(k) = \beta\mathbf{W}(k)$, if

$$\eta^R = \beta^2\eta. \quad (12.47)$$

12.12.2 Dynamic Equivalence Between Modular Recurrent Neural Networks for the ERLS Learning Algorithm

The extended recursive least-squares algorithm, given in Appendix D.3, as introduced in Baltersee and Chambers (1998) and Mandic *et al.* (1998) is based upon representing the dynamics of the PRNN in the state space form (Puskoris and Feldkamp 1994; Ruck *et al.* 1992)

$$\left. \begin{aligned} \mathbf{w}(k) &= \mathbf{w}(k-1) + \mathbf{q}(k) && \text{system equation,} \\ \mathbf{x}(k) &= \mathbf{h}(\mathbf{w}(k)) + \mathbf{v}(k) && \text{measurement equation,} \end{aligned} \right\} \quad (12.48)$$

where $\mathbf{w}(k)$ is the $N(N+p+1) \times 1$ vector obtained by rearranging the weight matrix $\mathbf{W}(k)$, $\mathbf{x}(k)$ is an $M \times 1$ observation vector, $\mathbf{q}(k) \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is a vector of white Gaussian noise (WGN), as well as $\mathbf{v}(k) \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$ (Williams 1992). The first equation in (12.48) is the system equation, represented by a random walk, and satisfies the properties of the static equivalence, given in Lemma 12.11.1. The measurement equation in (12.48) is linearised using a first-order Taylor expansion, i.e.

$$\mathbf{h}(\mathbf{w}(k)) \approx \mathbf{h}(\hat{\mathbf{w}}(k | k-1)) + \nabla \mathbf{h}^T(\hat{\mathbf{w}}(k | k-1))[\mathbf{w}(k) - \hat{\mathbf{w}}(k | k-1)], \quad (12.49)$$

where the gradient of $\mathbf{h}(\cdot)$ can be expressed as

$$\nabla \mathbf{h}^T = \frac{\partial \mathbf{h}(\hat{\mathbf{w}}(k | k-1))}{\partial \hat{\mathbf{w}}(k | k-1)} = \mathbf{H}(k). \quad (12.50)$$

Furthermore, the vector $\mathbf{h}(k)$, which is the result of the nonlinear mapping $\mathbf{h}(k) = \mathbf{h}(\mathbf{w}(k))$ is actually the $M \times 1$ vector of the outputs of the PRNN modules (Baltersee and Chambers 1998; Mandic *et al.* 1998):

$$\mathbf{h}^T(k) = [y_{1,1}(k), y_{2,1}(k), \dots, y_{M,1}(k)] \quad (12.51)$$

That means that the equivalence needed for the observation equation boils down to the dynamic equivalence derived for the GD learning (12.44)–(12.46).

Lemma 12.12.2 (see Mandic and Chambers 1999a). *An arbitrary modular recurrent neural network represented by β , η and $\mathbf{W}(k)$ is equivalent in the dynamic sense in terms of the extended recursive least-squares learning algorithm to a referent isomorphic modular recurrent neural network represented by $\beta^R = 1$, η^R and $\mathbf{W}^R(k)$, if*

- (i) they are equivalent in the static sense, $\mathbf{W}^R(k) = \beta\mathbf{W}(k)$, and
- (ii) the learning rates are related as $\eta^R = \beta^2\eta$.

Notice that condition (i) is correspondent to the system equation of (12.48), whereas the condition (ii) corresponds to the measurement equation of (12.48).

12.12.3 Equivalence Between an Arbitrary and the Referent PRNN

Some other learning algorithms for training modular neural networks rest upon either general backpropagation, such as the BPTT algorithm (Werbos 1990), or combine general backpropagation and the direct gradient descent algorithms (RTRL), such as the RBP algorithm (Pineda 1987). Naturally, from Mandic and Chambers (1999e) and the above analysis, the relationships derived are valid for both the BPTT and RBP algorithms. From the static equivalence given in Lemma 12.11.1 and dynamic equivalence given in Lemmas 12.12.1 and 12.12.2, the following theorem encompasses a general equivalence between an arbitrary modular neural network and the referent, modular neural network.

Theorem 12.12.3 (see Mandic and Chambers 1999a). *An arbitrary modular neural network represented by β , η and $\mathbf{W}(k)$ is equivalent to a referent isomorphic modular neural network represented by $\beta^R = 1$, η^R and $\mathbf{W}^R(k)$, if*

- (i) *they are equivalent in the static sense, i.e. $\mathbf{W}^R(k) = \beta\mathbf{W}(k)$, and*
- (ii) *they are equivalent in the dynamic sense, i.e. $\eta^R = \beta^2\eta$.*

As pointed out for common recurrent neural networks, the above analysis is also valid for the hyperbolic tangent $\gamma \tanh(\beta x) : \mathbb{R} \rightarrow (-\gamma, \gamma)$.

12.13 Note on the β - η - \mathbf{W} Relationships and Contractivity

Sigmoid activation functions used in neural networks have been shown to be either contractive or expansive. It is important to preserve this property for functionally equivalent networks that have different parameters, since the *a posteriori* and normalised algorithms rely upon contractivity of the nonlinear activation function of a neuron. In a real neural network, for the logistic activation function $\Phi(\xi) = 1/(1 + e^{-\beta\xi})$, for instance, ξ is replaced by the activation potential $\text{net}(k) = \mathbf{x}^T(k)\mathbf{w}(k)$. As the conditions of static and dynamic equivalence derived in this chapter effectively change the weights and learning rates, there is a need to address the contractivity/expansivity preservation between an arbitrary and the referent network.

A close inspection of the static equivalence equations shows that their form is

$$\mathbf{w}^R(k) = \beta\mathbf{w}_{\text{arbitrary}}(k). \quad (12.52)$$

Recall that for the referent network $\beta = 1$, the activation potentials of the referent and arbitrary network are given, respectively, by

$$\text{net}^R(k) = \mathbf{x}^T \mathbf{w}^R(k) \quad (12.53)$$

and

$$\text{net}(k) = \mathbf{x}^T \frac{\mathbf{w}^R(k)}{\beta}. \quad (12.54)$$

However, the outputs of either network are identical, since for the arbitrary network

$$\Phi(\beta, \text{net}(k)) = \Phi(\beta \text{net}(k)) = \Phi\left(\beta \mathbf{x}^T \frac{\mathbf{w}^R(k)}{\beta}\right) = \Phi(1, \text{net}^R(k)). \quad (12.55)$$

Hence, the contractivity/expansivity is preserved upon application of the equivalence relationships between an arbitrary and a referent neural network.

12.14 Summary

The relationship between the learning rate η and the gain in the general activation function β for a nonlinear optimisation algorithm which adapts the weights of the modular neural network has been derived. This relationship is derived both in the static sense (equality of the outputs of the neurons) and the dynamic sense (equality in learning processes), for both the gradient descent (GD) and the extended recursive least-squares (ERLS) algorithms. Such a result enables the use of one degree of freedom less when adjusting variable parameters of a general neural network, and hence reduces computational complexity of learning. The results provided are shown to be easily extended for the backpropagation through time and the recurrent backpropagation algorithms, when applied in this framework.

Appendix A

The \mathcal{O} Notation and Vector and Matrix Differentiation

A.1 The \mathcal{O} Notation

Definition A.1.1. Let f and g be $f, g : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ functions. If there exist positive numbers n_0 and c such that $f(n) \leq cg(n)$ for all $n \geq n_0$, the \mathcal{O} notation can be introduced as

$$f(n) = \mathcal{O}(g(n)), \quad (\text{A.1})$$

i.e. function f is asymptotically dominated by function g .

An algorithm is said to run in *polynomial time* if there exists $k \in \mathbb{Z}$ such that (Blondel 2000)

$$T(s) = \mathcal{O}(s^k), \quad (\text{A.2})$$

where $T(s)$ is the execution time and s is the length of the input.

A.2 Vector and Matrix Differentiation

Let us denote vectors by lowercase bold letters and matrices by capital bold letters. Some frequently used vector and matrix differentiation rules used in this book are

- $\frac{d(\mathbf{x}^T \mathbf{A})}{d\mathbf{x}} = \mathbf{A}$,
 - $\frac{d(\mathbf{x}^T \mathbf{A} \mathbf{y})}{d\mathbf{A}} = \mathbf{x} \mathbf{y}^T$,
 - $\frac{d(\mathbf{A} \mathbf{x} + \mathbf{b})^T \mathbf{C} (\mathbf{D} \mathbf{x} + \mathbf{e})}{d\mathbf{x}} = \mathbf{A}^T \mathbf{C} (\mathbf{D} \mathbf{x} + \mathbf{e}) + \mathbf{D}^T \mathbf{C}^T (\mathbf{A} \mathbf{x} + \mathbf{b})$,
- (i) $\frac{d(\mathbf{x}^T \mathbf{A} \mathbf{x})}{d\mathbf{x}} = (\mathbf{A} + \mathbf{A}^T) \mathbf{x}$,
- (ii) $\frac{d(\mathbf{x}^T \mathbf{x})}{d\mathbf{x}} = 2\mathbf{x}$,

- $\frac{d^2(\mathbf{y}^T \mathbf{x})}{d\mathbf{x}^2} = 0,$
- $\frac{d^2(\mathbf{A}\mathbf{x} + \mathbf{b})\mathbf{C}(\mathbf{D}\mathbf{x} + \mathbf{e})}{d\mathbf{x}^2} = \mathbf{A}^T \mathbf{C} \mathbf{D} + \mathbf{D}^T \mathbf{C}^T \mathbf{A}.$

Appendix B

Concepts from the Approximation Theory

Definition B.1.1 (see **Mhaskar and Micchelli 1992**). The function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ (not necessarily continuous) is called a *Kolmogorov* function, if for any integer $s \geq 1$, any compact set $K \subset \mathbb{R}^s$, any continuous function $f : K \rightarrow \mathbb{R}$ and any $\varepsilon > 0$, there exists an integer N , numbers $c_k, t_k \in \mathbb{R}$ and $\lambda_k \in \mathbb{R}^s$, $1 \leq k \leq N$, possibly depending upon s, K, f, ε , such that

$$\sup_{x \in K} \left| f(x) - \sum_{k=1}^N c_k \sigma(\lambda_k x - t_k) \right| < \varepsilon. \quad (\text{B.1})$$

Definition B.1.2. A sigmoidal function with properties,

$$\lim_{x \rightarrow -\infty} \frac{\sigma(x)}{x^k} = 0, \quad (\text{B.2})$$

$$\lim_{x \rightarrow +\infty} \frac{\sigma(x)}{x^k} = 1,$$

$$|\sigma(x)| \leq K(1 + |x|)^k, \quad K > 0, \quad (\text{B.3})$$

is called the k th degree sigmoidal function.

In other words, these functions are bounded on \mathbb{R} by a polynomial of degree $d \leq k$.

It follows that function σ is a candidate for a Kolmogorov function if it is not a polynomial.

Theorem B.1.3 (see **Kolmogorov 1957**). *There exist fixed increasing continuous functions $\psi_{pq}(x)$, on $I = [0, 1]$ so that each continuous function f on I^n can be written in the form*

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \psi_{pq}(x_p) \right), \quad (\text{B.4})$$

where Φ_q are properly chosen continuous functions of one variable.

This result asserts that every multivariate continuous function can be represented by the superposition of a small number of univariate continuous functions.

Theorem B.1.4 (Kolmogorov–Sprecher Theorem). For each integer $n \geq 2$, there exists a real monotonic increasing function $\psi(x)$, $\psi([0, 1]) = [0, 1]$, dependent on n and having the following property.

For each preassigned number $\delta > 0$, there is a rational number ε , $0 < \varepsilon < \delta$, such that every real continuous function of n variables, $\phi(\mathbf{x})$, defined on I^n , can be exactly represented by

$$f(\mathbf{x}) = \sum_{j=1}^{2n+1} \chi \left[\sum_{i=1}^n \lambda^i \psi(x_i + \varepsilon(j-1)) + j - 1 \right], \quad (\text{B.5})$$

where χ is a real and continuous function dependent upon f and λ is a constant independent of f .

Since no constructive method for the determination of χ is known, a direct application of the Kolmogorov–Sprecher Theorem is rather difficult.

Theorem B.1.5 (Weierstrass Theorem). If f is a continuous real-valued function on $[a, b] \in \mathbb{R}$, then for any $\varepsilon > 0$, there exists a polynomial P on $[a, b] \in \mathbb{R}$ such that

$$|f(x) - P(x)| < \varepsilon, \quad \forall x \in [a, b]. \quad (\text{B.6})$$

In other words, any continuous function on a closed and bounded interval can be uniformly approximated on that interval by polynomials to any degree of accuracy.

Definition B.1.6. Let P be a probability measure on \mathbb{R}^m . For measurable functions $f_1, f_2 : \mathbb{R}^m \rightarrow \mathbb{R}$ we say that f_1 approximates f_2 with accuracy $\varepsilon > 0$ and confidence $\delta > 0$ in probability if

$$P(x \in \mathbb{R}^m \mid |f_1(x) - f_2(x)| > \varepsilon) < \delta. \quad (\text{B.7})$$

The function f_1 interpolates f_2 on p examples x_1, \dots, x_p if $f_1(x_i) = f_2(x_i)$, $i = 1, \dots, p$.

Definition B.1.7. A function f that satisfies the condition

$$|f(x) - f(y)| \leq L|x - y|, \quad x, y \in \mathbb{R}, \quad L = \text{const.}, \quad (\text{B.8})$$

is called a Lipschitz function.

Definition B.1.8. A closure of a subset D of a topological space S , usually denoted by \bar{D} , is the set of points in S with the property that every neighbourhood of such a point has a nonempty intersection with D .

If S is a set, then by the closure \bar{S} of S we mean the set of all points in S together with the set of all limit points of S . A set S is closed if it is identical to its closure \bar{S} .

Definition B.1.9. A subset D of a topological space S is called dense if $\bar{D} = S$.

Topologies considered are naturally defined by a metric. The most commonly used metrics are \mathcal{L}_p , and among them the supremum metric. If D is dense in S , then each element of S can be approximated arbitrarily well by elements of D . Examples are the set of rational numbers, which is dense in \mathbb{R} , and the set of polynomials that is dense in the space of continuous functions.

Definition B.1.10. A compact set is one in which every infinite subset contains at least one limit point.

Every closed, bounded, finite-dimensional set in a metric linear space is compact.

Definition B.1.11. A cubic spline is a spline constructed of piecewise third-order polynomials defined by a set of control points.

A spline is an interpolating polynomial which uses information from neighbouring points to obtain smoothness.

Appendix C

Complex Sigmoid Activation Functions, Holomorphic Mappings and Modular Groups

C.1 Complex Sigmoid Activation Functions

Summarising some important notions from complex analysis, let w and z be complex numbers. Some of the elementary transformations defined on these complex numbers are given in the following table.

Mapping	Formula	Domain
Inversion	$f(z) = 1/z$	
Magnification	$f(z) = az$	$a \in \mathbb{R} \neq 0,$
Magnification + Rotation	$f(z) = az$	$a \in \mathbb{C} \neq 0,$
Möbius Transformation	$f(z) = \frac{az + b}{cz + d}$	$a, b, c, d \in \mathbb{C},$
Rotation	$f(z) = e^{j\theta}$	$\theta \in \mathbb{R},$
Translation	$f(z) = z + a$	$a \in \mathbb{C}.$

A linear transformation can be considered as a composition of a rotation, magnification and translation.

Definition C.1.1. A differentiable function of a complex variable is *holomorphic* if its derivative is continuous. If the first derivative of a holomorphic function is nonzero at some point z_0 , then that function is *conformal* at z_0 .

Definition C.1.2. A complex function which is analytic at all finite points of the complex plane is called entire.

Definition C.1.3. A meromorphic function has only a finite number of isolated poles and zeros. It can be thought of as a rational function of two entire functions.

Definition C.1.4 (Liouville Theorem). A bounded entire function in the complex plane is constant.

Cauchy–Riemann Equations: for a total derivative df/dz of a complex function $f(x, y) = u(x, y) + jv(x, y)$, where $z = x + jy$ and $dz = dx + j dy$, to exist, the following conditions, also known as Cauchy–Riemann equations, have to be satisfied:

$$\left. \begin{aligned} \frac{\partial u}{\partial x} &= \frac{\partial v}{\partial y}, \\ \frac{\partial v}{\partial x} &= -\frac{\partial u}{\partial y}. \end{aligned} \right\} \quad (\text{C.1})$$

Some properties of complex activation functions

For a function approximation problem, we have

$$f(x) = \sum_{i=1}^n c_i \sigma(x - a_i) = \sum_{i=1}^n \frac{c_i}{1 + e^{-x} e^{a_i}} = e^x \sum_{i=1}^n c_i \frac{1}{e^x + e^{a_i}}, \quad (\text{C.2})$$

which becomes

$$f(x) = z \sum_{i=1}^n \frac{c_i}{z + \alpha_i} = r(z) \quad (\text{C.3})$$

with the change of variables $z = e^x$ and $\alpha_i = e^{a_i}$. The rational function $r(z)$ is given by

$$r(z) = \sum_{i=1}^n \frac{c_i}{z + \alpha_i} = \frac{P(z)}{Q(z)}, \quad z \in \mathbb{C}. \quad (\text{C.4})$$

Recall that any analytic function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ such as the standard sigmoid has a convergent power series expansion $\sigma(x) = \sum_{i=0}^{\infty} \sigma_i (x - a)^i$ about a point $a \in \mathbb{R}$. When we substitute x by a complex number $z = x + jy$, we obtain a series $\sum_{i=0}^{\infty} \sigma_i (z - a)^i$, which converges in a disc $|z - a| < R$, where R is the radius of convergence of power series. Coefficients a_i correspond to the poles $\alpha_i = e^{a_i}$, whereas scaling factors c_i represent the residues of $r(z)$ at e^{a_i} (Williamson and Helmke 1995).

C.1.1 Modular Groups

Since a Möbius transformation remains unchanged if all the coefficients (a, b, c, d) are multiplied by the same nonzero constant, we will assume $ad - bc = 1$.

If we associate a matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad (\text{C.5})$$

with a Möbius transformation, then its determinant $\Delta = ad - bc = 1$. If A and B are matrices associated with Möbius transformations f and g , then a composition $f \circ g$ is described by the matrix product AB . In order to show that the set of all Möbius transformations forms a group under composition, we introduce the identity transformation

$$f(z) = z = \frac{1z + 0}{0z + 1},$$

which is described by the identity matrix I , which is the neutral element of the group. The matrix inverse A^{-1} is associated with the inverse of f given by

$$f^{-1}(z) = \frac{dz - b}{-cz + a}, \quad (\text{C.6})$$

which is the last condition for the set of all Möbius transformations to form the group under composition. Now, the set of all Möbius transformations of the form,

$$f(z) = \frac{az + b}{cz + d},$$

where a, b, c, d are integers with $ad - bc = 1$ forms a modular group and is denoted by Γ . This allows us to express the Möbius transformation as

$$Az = \frac{az + b}{cz + d}. \quad (\text{C.7})$$

The next theorem shows that the group Γ is generated by two transformations, $Tz = z + 1$ and $Sz = -1/z$ (Apostol 1997).

Theorem C.1.5 (see Apostol 1997). *The modular group Γ is generated by the two matrices*

$$T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad S = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad (\text{C.8})$$

and every $A \in \Gamma$ can be expressed in the form,

$$A = T^{n_1} S T^{n_2} S \cdots S T^{n_k}, \quad (\text{C.9})$$

where n_i are integers.

So, for instance, a modular representation for the matrix

$$A = \begin{bmatrix} 4 & 9 \\ 11 & 25 \end{bmatrix}$$

is $A = ST^{-3}ST^{-4}ST^2$. However, this solution is generally not unique.

Appendix D

Learning Algorithms for RNNs

D.1 The RTRL Algorithm

The structure of a single RNN is shown in Figure D.1. The neurons (nodes) are depicted by circles and incorporate the operation Φ (sum of inputs). For the n th neuron, its weights form a $(p + F + 1) \times 1$ dimensional weight vector $\mathbf{w}_n^T = [w_{n,1}, \dots, w_{n,p+F+1}]$, where p is the number of external inputs and F is the number of feedback connections, one remaining element of the weight vector \mathbf{w} being the bias input weight. The feedback connections represent the delayed output signals of the RNN. In the case of the network shown in Figure D.1, we have $N = F$. Such a network is called a fully connected recurrent neural network (FCRNN) (Williams and Zipser 1989a). The following equations fully describe the FCRNN,

$$y_n(k) = \Phi(v_n(k)), \quad n = 1, 2, \dots, N, \quad (\text{D.1})$$

$$v_n(k) = \sum_{l=1}^{p+N+1} w_{n,l}(k)u_l(k), \quad (\text{D.2})$$

$$\mathbf{u}_n^T(k) = [s(k-1), \dots, s(k-p), 1, y_1(k-1), y_2(k-1), \dots, y_N(k-1)], \quad (\text{D.3})$$

where the $(p + N + 1) \times 1$ dimensional vector \mathbf{u} comprises both the external and feedback inputs to a neuron, with vector \mathbf{u} having ‘unity’ for the constant bias input.

For the nonlinear time series prediction paradigm, there is only one output neuron of the RNN. RTRL-based training of the RNN is based upon minimising the instantaneous squared error at the output of the first neuron of the RNN (Haykin 1994; Williams and Zipser 1989a), which can be expressed as

$$\min(\frac{1}{2}e^2(k)) = \min(\frac{1}{2}[s(k) - y_1(k)]^2), \quad (\text{D.4})$$

where $e(k)$ denotes the error at the output y_1 of the RNN and $s(k)$ is the teaching signal. Hence, the correction for the l th weight of neuron k at the time instant k can

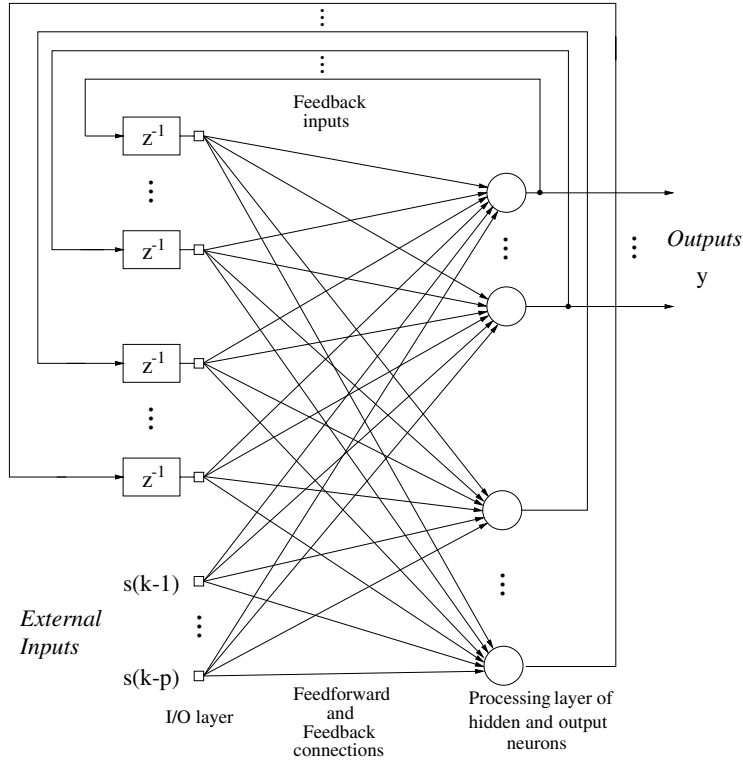


Figure D.1 Single recurrent neural network

be derived as follows:

$$\begin{aligned}\Delta w_{n,l}(k) &= -\frac{\eta}{2} \frac{\partial}{\partial w_{n,l}(k)} e^2(k) \\ &= -\eta e(k) \frac{\partial e(k)}{\partial w_{n,l}(k)}.\end{aligned}\tag{D.5}$$

Since the external signal vector \mathbf{s} does not depend on the elements of \mathbf{W} , the error gradient becomes

$$\frac{\partial e(k)}{\partial w_{n,l}(k)} = -\frac{\partial y_1(k)}{\partial w_{n,l}(k)}.\tag{D.6}$$

Using the chain rule, this can be rewritten as¹

$$\begin{aligned}\frac{\partial y_1(k)}{\partial w_{n,l}(k)} &= \Phi'(v_1(k)) \frac{\partial v_1(k)}{\partial w_{n,l}(k)} \\ &= \Phi'(v_1(k)) \left(\sum_{\alpha=1}^N \frac{\partial y_{\alpha}(k-1)}{\partial w_{n,l}(k)} w_{1,\alpha+p+1}(k) + \delta_{n1} u_l(k) \right),\end{aligned}\tag{D.7}$$

¹ A detailed derivation of the RTRL algorithm can be found in Williams and Zipser (1989a), Haykin (1994) and Williams and Zipser (1989b).

where

$$\delta_{nl} = \begin{cases} 1, & n = l, \\ 0, & n \neq l. \end{cases} \quad (\text{D.8})$$

Under the assumption, also used in the RTRL algorithm (Narendra and Parthasarathy 1990; Robinson and Fallside 1987; Williams and Zipser 1989a), that when the learning rate η is sufficiently small, we have

$$\frac{\partial y_\alpha(k-1)}{\partial w_{n,l}(k)} \approx \frac{\partial y_\alpha(k-1)}{\partial w_{n,l}(k-1)}. \quad (\text{D.9})$$

A triply indexed set of variables $\{\pi_{n,l}^j(k)\}$ can be introduced to characterise the RTRL algorithm for the RNN, as

$$\pi_{n,l}^j = \frac{\partial y_j(k)}{\partial w_{n,l}} \quad 1 \leq j, n \leq N, \quad 1 \leq l \leq p+1+N, \quad (\text{D.10})$$

which is used to compute recursively the values of $\pi_{n,l}^j$ for every time step k and all appropriate j , n and l as follows,

$$\pi_{n,l}^j(k+1) = \Phi'(v_j) \left[\sum_{m=1}^N w_{j,m+p+1}(k) \pi_{n,l}^m(k) + \delta_{nj} u_l(k) \right], \quad (\text{D.11})$$

with the values for j , n and l as in (D.11) and the initial conditions

$$\pi_{n,l}^j(0) = 0. \quad (\text{D.12})$$

To simplify the presentation, we introduce three new matrices, the $N \times (N+p+1)$ matrix $\mathbf{\Pi}_j(k)$, the $N \times (N+p+1)$ matrix $\mathbf{U}_j(k)$ and the $N \times N$ diagonal matrix $\mathbf{F}(k)$, as (Haykin 1999b)

$$\mathbf{\Pi}_j(k) = \frac{\partial \mathbf{y}(k)}{\partial \mathbf{w}_j(k)}, \quad \mathbf{y} = [y_1(k), \dots, y_N(k)], \quad j = 1, 2, \dots, N, \quad (\text{D.13})$$

$$\mathbf{U}_j(k) = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{u}(k) \\ \vdots \\ \mathbf{0} \end{bmatrix} \leftarrow j\text{th row}, \quad j = 1, 2, \dots, N, \quad (\text{D.14})$$

$$\mathbf{F}(k) = \text{diag}[\Phi'(\mathbf{u}(k)^T \mathbf{w}_1(k)), \dots, \Phi'(\mathbf{u}(k)^T \mathbf{w}_N(k))]. \quad (\text{D.15})$$

Hence, the gradient updating equation regarding the recurrent neuron can be symbolically expressed as (Haykin 1994; Williams and Zipser 1989a)

$$\mathbf{\Pi}_j(k+1) = \mathbf{F}(k)[\mathbf{U}_j(k) + \mathbf{W}_a(k)\mathbf{\Pi}_j(k)], \quad j = 1, 2, \dots, N, \quad (\text{D.16})$$

where \mathbf{W}_a denotes the set of those entries in \mathbf{W} which correspond to the feedback connections.

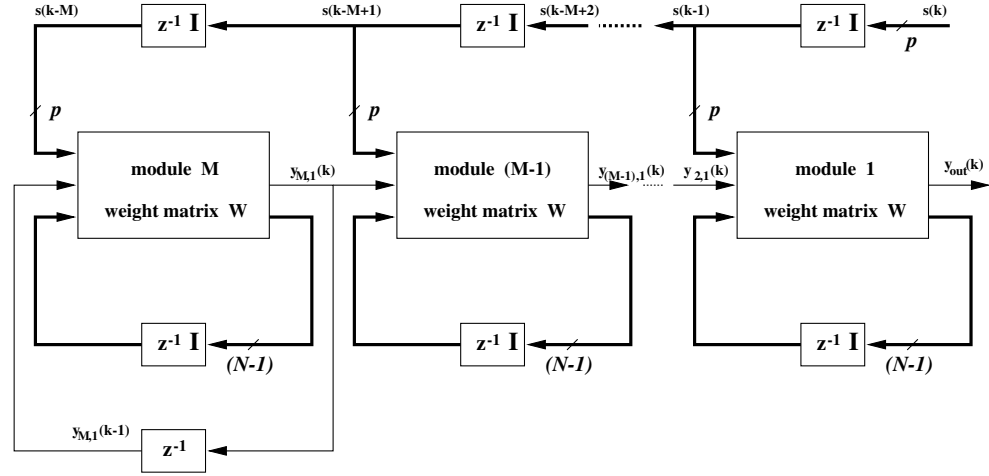


Figure D.2 Pipelined recurrent neural network

D.1.1 Teacher Forcing Modification of the RTRL Algorithm

The idea is to replace the actual output $y_1(k)$ by a teacher signal $s(k)$ in subsequent computation of the behaviour of the network. The derivation of the algorithm follows the same concept as for the RTRL algorithm. Notice that the partial derivatives of $s(k)$ with respect to $\mathbf{W}(k)$ are zero. Having this in mind, the equations that calculate the sensitivities \mathbf{II} have the same form as for the RTRL, except that the values of \mathbf{II} that refer to $s(k)$ are zero.

D.2 Gradient Descent Learning Algorithm for the PRNN

The PRNN is a modular neural network and consists of a certain number M of RNNs as its modules, with each module consisting of N neurons. In the PRNN configuration, the M modules, which are RNNs, are connected as shown in Figure D.2. The $(p \times 1)$ -dimensional external signal vector $\mathbf{s}^T(k) = [s(k-1), \dots, s(k-p)]$ is delayed by m time steps ($z^{-m}\mathbf{I}$) before feeding the module m . All the modules operate using the same weight matrix \mathbf{W} . The overall output signal of the PRNN is $y_{out}(k) = y_{1,1}(k)$, i.e. the output of the first neuron of the first module. Thus, the overall cost function of the PRNN becomes (Baltersee and Chambers 1998; Haykin and Li 1995)

$$E(k) = \frac{1}{2} \sum_{i=1}^M \lambda^{i-1} e_i^2(k), \quad (\text{D.17})$$

where $e_i(k)$ is the error from module i and a *forgetting factor* $\lambda \in (0, 1]$, is introduced which determines the weighting of the individual modules.

A full mathematical description of the PRNN is given in the following equations:

$$y_{i,n}(k) = \Phi(v_{i,n}(k)), \quad i = 1, \dots, M, \quad n = 1, \dots, N, \quad (\text{D.18})$$

$$v_{i,n}(k) = \sum_{l=1}^{p+N+1} w_{n,l}(k) u_{i,l}(k), \quad (\text{D.19})$$

$$\mathbf{u}_i^T(k) = [s(k-i), \dots, s(k-i-p+1), 1, y_{i+1,1}(k), \\ y_{i,2}(k-1), \dots, y_{i,N}(k-1)] \quad \text{for } 1 \leq i \leq M-1, \quad (\text{D.20})$$

$$\mathbf{u}_M^T(k) = [s(k-M), \dots, s(k-M-p+1), 1, y_{M,1}(k-1), \\ y_{M,2}(k-1), \dots, y_{M,N}(k-1)] \quad \text{for } i = M. \quad (\text{D.21})$$

At the time step k , for each module i , $i = 1, \dots, M$, the one-step forward prediction error $e_i(k)$ associated with a module, is

$$e_i(k) = s(k-i+1) - y_{i,1}(k). \quad (\text{D.22})$$

The cost function for the PRNN is

$$E(k) = \frac{1}{2} \sum_{i=1}^M \lambda^{i-1} e_i^2(k), \quad (\text{D.23})$$

i.e. a weighted sum of squared errors at the output of every module of the PRNN. Hence, the correction for the l th weight of neuron n at the time instant k is derived as follows:

$$\Delta w_{n,l}(k) = -\frac{\eta}{2} \frac{\partial}{\partial w_{n,l}(k)} \left(\sum_{i=1}^M \lambda^{i-1} e_i^2(k) \right) \\ = -\eta \sum_{i=1}^M \lambda^{i-1} e_i(k) \frac{\partial e_i(k)}{\partial w_{n,l}(k)}. \quad (\text{D.24})$$

Now,

$$\frac{\partial e_i(k)}{\partial w_{n,l}(k)} = -\frac{\partial y_{i,1}(k)}{\partial w_{n,l}(k)}, \quad (\text{D.25})$$

which can be rewritten as

$$\frac{\partial y_{i,1}(k)}{\partial w_{n,l}(k)} = \Phi'(v_{i,1}(k)) \frac{\partial v_{i,1}(k)}{\partial w_{n,l}(k)}. \quad (\text{D.26})$$

Now, inserting (D.18)–(D.21) into (D.26) yields (Baltersee and Chambers 1998; Haykin and Li 1995)

$$\Phi'(v_{i,1}(k)) \frac{\partial v_{i,1}(k)}{\partial w_{n,l}(k)} = \Phi'(v_{i,1}(k)) \left(\sum_{\alpha=1}^{p+N+1} \left(\frac{\partial w_{1,\alpha}(k)}{\partial w_{n,l}(k)} u_{i,\alpha}(k) + \frac{\partial u_{i,\alpha}(k)}{\partial w_{n,l}(k)} w_{1,\alpha}(k) \right) \right). \quad (\text{D.27})$$

The first term in (D.27) is zero except for $n = 1$ and $l = \alpha$ (Williams and Zipser 1989a), and the only elements of the input vector \mathbf{u} that depend on the elements of

\mathbf{W} are the feedback values. Therefore, Equation (D.27) can be simplified to

$$\Phi'(v_{i,1}(k)) \frac{\partial v_{i,1}(k)}{\partial w_{n,l}(k)} = \Phi'(v_{i,1}(k)) \left(\sum_{\alpha=1}^N \frac{\partial y_{i,\alpha}(k-1)}{\partial w_{n,l}(k)} w_{1,\alpha+p+1}(k) + \delta_{n1} u_{i,l}(k) \right), \quad (\text{D.28})$$

where

$$\delta_{nl} = \begin{cases} 1, & n = l, \\ 0, & n \neq l. \end{cases} \quad (\text{D.29})$$

A quadruply indexed set of variables $\{\pi_{n,l}^{ij}(k)\} = \partial y_{i,j}(k) / \partial w_{n,l}$ can be introduced to characterise the RTRL algorithm for the PRNN, as

$$\pi_{n,l}^{ij} = \frac{\partial y_{i,j}(k)}{\partial w_{n,l}}, \quad 1 \leq i \leq M, \quad 1 \leq j, \quad n \leq N, \quad 1 \leq l \leq p+1+N. \quad (\text{D.30})$$

The variant of the RTRL algorithm suitable for the PRNN is used to compute recursively the values of $\pi_{n,l}^{ij}$ for every time step k and all appropriate i, j, n and l as follows,

$$\pi_{n,l}^{ij}(k+1) = \Phi'(v_{i,j}) \left[\sum_{m=1}^N w_{j,m+p+1}(k) \pi_{n,l}^{im}(k) + \delta_{nj} u_{i,l}(k) \right], \quad (\text{D.31})$$

with the values for i, j, n and l as in (D.31) and the initial conditions,

$$\pi_{n,l}^{ij}(0) = 0, \quad (\text{D.32})$$

where $u_{i,l}$ and $v_{i,j}$ are, respectively, the l th input to the i th module of the PRNN and the internal activation function of the j th neuron in the i th module. Updating process (D.30) can be written in the same matrix form as (D.16).

D.3 The ERLS Algorithm

The extended recursive least-squares (ERLS) algorithm, which is presented here (Baltersee and Chambers 1998; Mandic *et al.* 1998), is based upon the idea of the extended Kalman filter (EKF) (Haykin 1996a; Iiguni and Sakai 1992; Kay 1993). It is presented for the general case of the PRNN and boils down to the ERLS algorithm for the RNN for a PRNN with one module. The cost function of the PRNN becomes

$$E_{\text{ERLS}}(k) = \frac{1}{2} \sum_{l=1}^k \xi^{k-l} E(l), \quad (\text{D.33})$$

which is to be minimised with respect to the elements of the weight matrix \mathbf{W} . The newly introduced constant $\xi \in (0, 1]$ represents a *forgetting factor* so that the resulting learning algorithm becomes suitable for the prediction task of non-stationary signals. The ERLS algorithm is used to solve the nonlinear minimisation problem of (D.33). In order to derive the ERLS algorithm, the vector state–vector observation Kalman

filter equations are considered (Haykin *et al.* 1997; Kay 1993),

$$\mathbf{w}(k) = \mathbf{a}(\mathbf{w}(k-1)) + \mathbf{u}(k), \quad (\text{D.34})$$

$$\mathbf{x}(k) = \mathbf{h}(\mathbf{w}(k)) + \mathbf{v}(k), \quad (\text{D.35})$$

where $\mathbf{w}(k)$ becomes the $N(N+p+1) \times 1$ weight vector, $\mathbf{x}(k)$ is the $M \times 1$ observation (signal) vector, $\mathbf{u}(k)$ is a white Gaussian noise vector, $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ and $\mathbf{v}(k)$ is observation noise, WGN vector $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$.² Furthermore, we have the nonlinear mapping functions,

$$\mathbf{a} : \mathbb{R}^{N(N+p+1)} \rightarrow \mathbb{R}^{N(N+p+1)} \quad (\text{D.36})$$

and

$$\mathbf{h} : \mathbb{R}^{N(N+p+1)} \rightarrow \mathbb{R}^M, \quad (\text{D.37})$$

which, respectively, map the space spanned over the weighting vector \mathbf{w} onto the same space and the weight vector space onto the ‘output of the PRNN’-dimensional space. For prediction of speech, however, the function $\mathbf{a}(\cdot)$ is unknown, so that the state Equation (D.34) may be approximated by the random walk model (Haykin 1994; Puskoris and Feldkamp 1994; Ruck *et al.* 1992)

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{u}(k). \quad (\text{D.38})$$

As for the EKF, the nonlinear mapping function $\mathbf{h}(\cdot)$ is linearised using the first-order Taylor expansion around the estimate of $\mathbf{w}(k)$, based on the previous data, i.e. $\hat{\mathbf{w}}(k | k-1)$, which yields

$$\mathbf{h}(\mathbf{w}(k)) \approx \mathbf{h}(\hat{\mathbf{w}}(k | k-1)) + \nabla \mathbf{h}^T [\mathbf{w}(k) - \hat{\mathbf{w}}(k | k-1)], \quad (\text{D.39})$$

where the gradient of $\mathbf{h}(\cdot)$ can be written as

$$\nabla \mathbf{h}^T = \frac{\partial \mathbf{h}(\hat{\mathbf{w}}(k | k-1))}{\partial \hat{\mathbf{w}}(k | k-1)} = \mathbf{H}(k) \quad (\text{D.40})$$

so that the observation equation becomes

$$\mathbf{x}(k) = \mathbf{H}(k)\mathbf{w}(k) + \mathbf{v}(k) + [\mathbf{h}(\hat{\mathbf{w}}(k | k-1)) - \mathbf{H}(k)\hat{\mathbf{w}}(k | k-1)]. \quad (\text{D.41})$$

Moreover, the correlation matrix of the process state noise vector $\mathbf{u}(k)$ equals a scaled version of the minimum mean square error matrix of the EKF (Haykin 1994; Kay 1993)

$$\mathbf{Q}(k) = E[\mathbf{u}(k)\mathbf{u}^T(k)] = (\xi^{-1} - 1)\mathbf{M}(k), \quad (\text{D.42})$$

where ξ is the forgetting factor of (D.33). Using (D.34), (D.41), (D.42) and the definition of the EKF in Kay (1993), the final equations of the ERLS algorithm for the PRNN become (Baltersee and Chambers 1998; Mandic *et al.* 1998)

$$\mathbf{K}(k) = \xi^{-1}\mathbf{M}(k-1)\mathbf{H}^T[\mathbf{C}(k) + \xi^{-1}\mathbf{H}(k)\mathbf{M}(k-1)\mathbf{H}^T(k)]^{-1}, \quad (\text{D.43})$$

$$\hat{\mathbf{w}}(k) = \hat{\mathbf{w}}(k-1) + \mathbf{K}(k)[\mathbf{x}(k) - \mathbf{h}(\hat{\mathbf{w}}(k-1))], \quad (\text{D.44})$$

$$\mathbf{M}(k) = \xi^{-1}[\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k)]\mathbf{M}(k-1). \quad (\text{D.45})$$

² Although the observation noise vector \mathbf{v} is to be, generally speaking, described by its covariance matrix, \mathbf{C} , we will assume that that matrix is diagonal, i.e. that the observation noise \mathbf{v} is satisfactorily described by its variance vector, together with its mean value.

For the PRNN, the $(M \times 1)$ -dimensional vector $\mathbf{x}(k)$ becomes

$$\mathbf{x}^T(k) = [s(k), s(k-1), \dots, s(k-M+1)], \quad (\text{D.46})$$

which is the input signal itself. Furthermore, the $(M \times 1)$ -dimensional vector, $\mathbf{h}(k) = \mathbf{h}(\mathbf{w}(k))$ becomes

$$\mathbf{h}^T(k) = [y_{1,1}(k), y_{2,1}(k), \dots, y_{M,1}(k)]. \quad (\text{D.47})$$

Now, since by (D.34), (D.38)

$$\hat{\mathbf{w}}(k | k-1) = \hat{\mathbf{w}}(k-1 | k-1) = \hat{\mathbf{w}}(k-1)$$

the gradient matrix $\mathbf{H} = \nabla \mathbf{h}$ becomes

$$\mathbf{H}(k) = \frac{\partial \mathbf{h}(\hat{\mathbf{w}}(k-1))}{\partial \hat{\mathbf{w}}(k-1)} \quad (\text{D.48})$$

the elements of which are available from Mandic *et al.* (1998) and Baltersee and Chambers (1998)

$$\frac{\partial y_{M,j}(k)}{\partial w_{n,l}(k)} \approx \Phi'(v_{M,j}(k)) \left[\sum_{\alpha=1}^N \frac{\partial y_{M,\alpha}(k-1)}{\partial w_{n,l}(k-1)} w_{j,\alpha+p+1}(k) + \delta_{nj} u_{M,l}(k) \right] \quad (\text{D.49})$$

for $i = M$, and

$$\begin{aligned} \frac{\partial y_{i,j}(k)}{\partial w_{n,l}(k)} \approx \Phi'(v_{i,j}(k)) & \left[\frac{\partial y_{i+1,j}(k)}{\partial w_{n,l}(k)} w_{j,p+2}(k) \right. \\ & \left. + \sum_{\alpha=2}^N \frac{\partial y_{i,\alpha}(k-1)}{\partial w_{n,l}(k-1)} w_{j,\alpha+p+1}(k) + \delta_{nj} u_{i,l}(k) \right] \quad (\text{D.50}) \end{aligned}$$

for $i \neq M$, where δ_{nj} is given by (D.8), so that the derivation of the ERLS algorithm for the PRNN is now complete. The ERLS algorithm for an RNN can be obtained from the derived algorithm, for the number of modules $M = 1$.

Appendix E

Terminology Used in the Field of Neural Networks

The field of artificial neural networks has developed alongside many disciplines, such as neurobiology, mathematics, statistics, economics, computer science, engineering and physics, to mention but a few. Consequently, the terminology used in the field varies from discipline to discipline. An initiative from the IEEE Neural Networks Council to standardise the terminology has resulted in recommended terminology and several definitions (Eberhart 1990). We present four of them.

Activation Function. Algorithm for computing the activation value of a neurode as a function of its net input. Net input is typically the sum of weighted inputs to the neurode.

Feedforward Network. Network ordered into layers with no feedback paths. The lowest layer is the input layer, the highest is the output layer. The outputs of a given layer go only to higher layers and its inputs come only from lower layers.

Supervised Learning. Learning procedure in which a network is presented with a set of input pattern and target pairs. The network can compare its output to the target and adapt itself according to the learning rules.

Unsupervised Learning. Learning procedure in which the network is presented with a set of input patterns. The network adapts itself according to the statistical associations in the input patterns.

A later effort by Fiesler (1994), which appeared in *Computer Standards and Interfaces*, further tackles this issue. In particular, Fiesler considers functions important for neural networks, such as

1. neuron functions (or transfer functions) which specify the output of a neuron, given its inputs (this includes nonlinearity);
2. learning rules (or learning laws) which define how weights (and offsets) will be updated;

3. clamping functions, which determine if and when certain neurons will be insensitive to incoming information, i.e. they retain their present activation value; and
4. ontogenic functions, which specify changes in the neural network topology.

Hence, a neural network is described by its topology, constraints, initial state and transition function. Topology includes the frame and interconnection structure. In our work we refer to frame as topology and interconnection structure as architecture.

Ljung and Sjöberg have offered a link between terms from the areas of control theory and neural networks (Ljung and Sjöberg 1992), of which we present several frequently used ones:

model structure	net,
model order	number of hidden units,
estimation	training, learning,
iteration	training cycle,
recursive gradient algorithm	backpropagation,
overfit	overtraining.

Appendix F

On the *A Posteriori* Approach in Science and Engineering

F.1 History of *A Posteriori* Techniques

In the Oxford Interactive Encyclopedia, 1997, the notions of *a priori* and *a posteriori* are defined as follows: ‘*a priori* is a term from epistemology meaning knowledge or concepts which can be gained independently of all experience. It is contrasted with *a posteriori* knowledge, in which experience plays an essential role’. *A posteriori* techniques have been considered for more than two millennia now. Probably the oldest written study on *a posteriori* reasoning techniques in logic was by Aristotle, sometime between 343 BC and 323 BC (Aristotle 1975). He developed *a posteriori* conclusion techniques. In the late sixteenth century, Galileo composed a manuscript between 1589 and 1591, nowadays known as MS27, while he was teaching or preparing to teach at the University of Pisa (Wallace 1992). The manuscript was a study, based upon Aristotle’s famous books *Prior Analytics* and *Posterior Analytics*¹ (Aristotle 1975), upon which logic had been taught at universities ever since the fourth century BC. Galileo’s work, based upon the canons introduced by Aristotle in *Posterior Analytics*, includes *Discourse on Bodies on or in Water* and *Letters on Sunspots*. He studies the role of the notions of foreknowledge and demonstration in science in general. ‘A science can give a real definition of its total subject *a posteriori* only, because the real definition is not foreknown in the science, therefore it is sought, therefore it is demonstrable’ (Wallace 1992). ‘We know something either *a posteriori* or *a priori*, *a posteriori* through demonstration of the fact, *a priori* through demonstration of the reasoned fact. *A posteriori* is referred to as demonstration *from an effect* or *conjectural*’ (Wallace 1992).

Galileo also adapted the *regressus* procedure to accommodate it with his experimental techniques. The *regressus*, after Galileo, required that cause and effect be convertible.

¹ Aristotle used the word *analytics* to represent today’s meaning of logic.

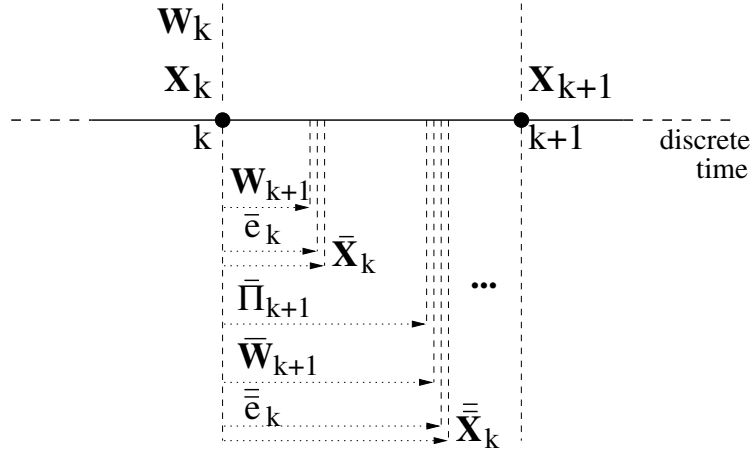


Figure F.1 Time management of the *a posteriori* approach

F.2 The Usage of *A Posteriori*

The notion *a posteriori* is also used in the work based upon Bayes' theorem, in solving linear differential equations, and in Kalman filtering. In modern technical disciplines, *a priori* quantities are estimated from past measurements of a process in hand or presupposed by experience, while *a posteriori* quantities are measured or computed from observations (Stark and Woods 1986).

The *a posteriori* quantities in learning algorithms, such as an error or output, are based upon the newly calculated set of weights $\mathbf{W}(k+1)$, which is available sometime between the consecutive discrete time instants k and $(k+1)$. The *a posteriori* quantities will be denoted with the same symbol as the corresponding *a priori* ones, with the addition of a bar, i.e. $\bar{e}(k)$ is the *a posteriori* error at the time instant k , calculated as $\bar{e}(k) = d(k) - \Phi(\mathbf{X}^T(k)\mathbf{W}(k+1))$, where $\mathbf{X}(k)$ is some general input vector and $\mathbf{W}(k)$ some general set of weights. The corresponding *a priori* error $e(k)$ is calculated as $e(k) = d(k) - \Phi(\mathbf{X}^T(k)\mathbf{W}(k))$.

For prediction applications, for instance, there may not be sufficient information to track the trajectory in the state space uniquely, or noise in the observable time series may add too much uncertainty to be able to satisfy the prediction requirements.

A way to avoid these problems is to involve some sort of regularisation through, for instance, the use of the prior knowledge of the system. *A posteriori* techniques naturally use such prior knowledge, by data-reusing, and the equations which describe such algorithms exhibit normalisation, which additionally stabilised the system.

F.2.1 *A Posteriori* Techniques in the RNN Framework

To further illustrate this concept in adaptive real-time algorithms, let us consider the situation shown in Figure F.1. All relevant time instants in Figure F.1 are denoted in the index of the appropriate variable, e.g. \mathbf{X}_k instead of $\mathbf{X}(k)$. At the discrete time interval k , the RNN has vectors $\mathbf{X}(k)$ and $\mathbf{W}(k)$ (and inherently $\mathbf{II}(k)$) available for

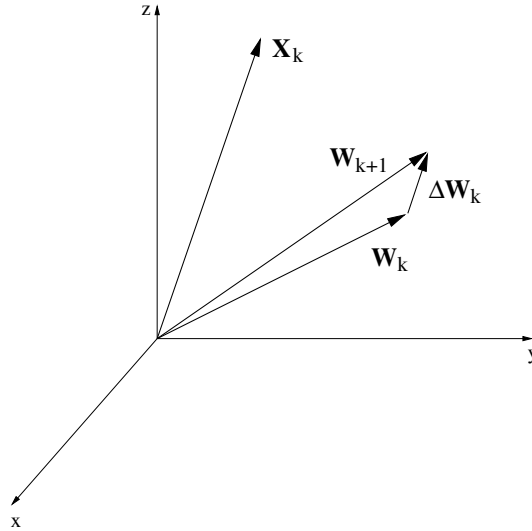


Figure F.2 Weight updating by the LMS

further calculation. For either the *a priori* or *a posteriori* approach, there is a need to calculate $\mathbf{W}(k+1)$, which must be done before the time instant $(k+1)$. Therefore, speaking in terms of discrete time, it appears before $(k+1)$ and thus belongs to k , but this is not strictly true, since $\mathbf{W}(k+1)$ is not available at k . Reiterating this idea, all the variables necessary for *a posteriori* techniques are shown in Figure F.1, together with the notion of the time of their availability.

F.2.2 The Geometric Interpretation of A Posteriori Error Learning

For the sake of simplicity we treat only the linear feedforward case with only one node. The equations which specify the algorithm can be written as

$$\left. \begin{aligned} \mathbf{w}_{i+1}(k) &= \mathbf{w}_i(k) + \eta e_i(k) \mathbf{x}(k), & i = 1, \dots, L, \\ e_i(k) &= d(k) - \mathbf{x}^T(k) \mathbf{w}_i(k), \end{aligned} \right\} \quad (\text{F.1})$$

where $\mathbf{w}_1(k) = \mathbf{w}(k)$ and $\mathbf{w}(k+1) = \mathbf{w}_{L+1}(k)$. For $L = 1$ Equation (F.1) degenerates into the LMS algorithm. From (F.1), it is obvious that the direction of the vectors $\Delta \mathbf{w}_i(k)$ is the same as the direction of the input vector $\mathbf{x}(k)$ (collinear). This is further depicted in Figure F.2 (Widrow and Lehr 1990). As the LMS gives only an approximate solution of the Wiener filtering problem, the quadratic surface defined by $e^2(k)$ has a solution set which is a linear variety, instead of a single minimum. Hence, the space of solution is a hyperplane (Schnauffer and Jenkins 1993), whose dimension is one less than the space upon it rests, and vector $\mathbf{x}(k)$ is perpendicular to the solution hyperplane $S(k)$. Figure F.3 provides the geometric interpretation of the relation between the LMS, NLMS and *a posteriori* (data-reusing) LMS. The direction of vectors \mathbf{w} is $\mathbf{w}(k) + \text{span}(\mathbf{x}(k))$ (Schnauffer and Jenkins 1993). As the LMS algorithm is an approximative algorithm which uses instantaneous estimates

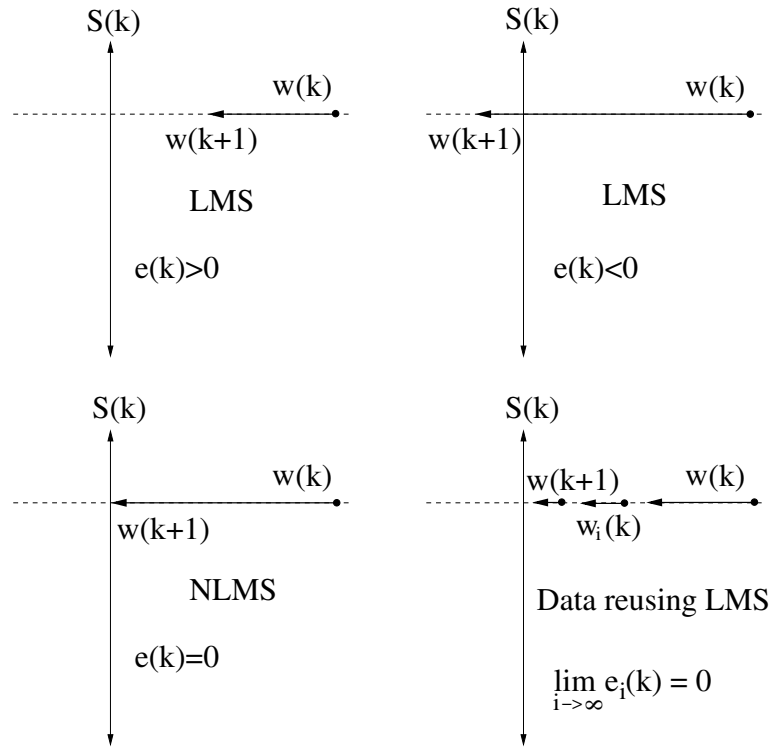


Figure F.3 Geometric interpretation of data-reusing techniques

instead of statistical expectations, the output error of a filter trained by LMS is either positive (the weight update short of an optimal weight update) or negative (the weight update exceeds the optimal weight update), which is shown in the top two diagrams of Figure F.3. The NLMS algorithm, on the other hand is defined to minimise the instantaneous output error ($e(k) = 0$), which is geometrically shown in the bottom left diagram in the figure. The *a posteriori* (data-reuse) algorithms start either from the situation described by $e(k) > 0$ or $e(k) < 0$, i.e. from either of the top two diagrams in Figure F.3. The iterative weight updates then approach the performance of normalised algorithms as the number of iterations increases, as shown in the bottom right diagram in the figure. From Figure F.3, it is evident that repeating a data-reusing technique for a sufficient number of times, the process approaches the normalised (zero-error) solution.

Appendix G

Contraction Mapping Theorems

G.1 Fixed Points and Contraction Mapping Theorems

When numerically solving $F : \mathbb{R} \rightarrow \mathbb{R}$ for its zeros, i.e. to find an x^* such that $F(x^*) = 0$, a convenient approach is to rearrange $F(x) = 0$ into $x = K(x)$ and undertake the iteration (Dennis and Schnabel 1983),

$$x_{i+1} = K(x_i), \quad x_0 \text{ chosen.} \quad (\text{G.1})$$

If (G.1) converges to x^* , then, by continuity,

$$x^* = \lim_{i \rightarrow \infty} K(x_i) = K(x^*). \quad (\text{G.2})$$

Hence, x^* is a fixed point of K and thus a zero of F .

G.1.1 Contraction Mapping Theorem in \mathbb{R}

Theorem G.1.1. *If*

- (i) $x \in [a, b] \Rightarrow K(x) \in [a, b]$,
- (ii) $\exists \gamma < 1$ such that $|K(x) - K(y)| \leq \gamma|x - y|, \forall x, y \in [a, b]$,

then the equation $x = K(x)$ has a unique solution $x^ \in [a, b]$ and the iteration*

$$x_{i+1} = K(x_i) \quad (\text{G.3})$$

converges to x^ for any $x_0 \in [a, b]$.*

Proof. Existence: From (i), we have $a - K(a) \leq 0$ and $b - K(b) \leq 0$, as shown in Figure G.1. The intermediate value theorem (IVT) shows that there is some solution $x^* \in [a, b]$ such that $x^* = K(x^*)$.

Uniqueness: If $\bar{x} \in [a, b]$ is also a solution of (G.3), then

$$|\bar{x} - x^*| = |K(\bar{x}) - K(x^*)| \leq \gamma|\bar{x} - x^*|. \quad (\text{G.4})$$



Figure G.1 The contraction mapping

Since $\gamma < 1$, this is a contradiction unless $\bar{x} \equiv x^*$.

Convergence: We have

$$|x_i - x^*| = |K(x_{i-1}) - K(x^*)| \leq \gamma |x_{i-1} - x^*|. \quad (\text{G.5})$$

Thus $|x_i - x^*| \leq \gamma^i |x_0 - x^*|$ and $\lim_{i \rightarrow \infty} \gamma^i = 0$. Hence $\{x_i\} \xrightarrow{i} x^*$.

Condition (ii) from Theorem G.1.1 is checked by showing that $|K'(x)| \leq \gamma < 1$, $\forall x \in (a, b)$. Namely the mean value theorem (MVT) shows that $\exists \xi \in (a, b)$ such that

$$|K(x) - K(y)| = |K'(\xi)(x - y)| = |K'(\xi)| |x - y| \leq \gamma |x - y|, \quad x, y \in [a, b], \quad \xi \in (a, b). \quad (\text{G.6})$$

G.1.2 Contraction Mapping Theorem in \mathbb{R}^N

Theorem G.1.2. Let \mathcal{M} be a closed subset of \mathbb{R}^N such that

- (i) $K : \mathcal{M} \rightarrow \mathcal{M}$,
- (ii) $\exists \gamma < 1$ such that $\|K(\mathbf{x}) - K(\mathbf{y})\| \leq \gamma \|\mathbf{x} - \mathbf{y}\|$, $\forall \mathbf{x}, \mathbf{y} \in \mathcal{M}$,

then equation

$$\mathbf{x} = K(\mathbf{x}) \quad (\text{G.7})$$

has a unique solution $\mathbf{x}^* \in \mathcal{M}$ and the iteration

$$\mathbf{x}_{i+1} = K(\mathbf{x}_i) \quad (\text{G.8})$$

converges to \mathbf{x}^* for any starting value $\mathbf{x}_0 \in \mathcal{M}$.

G.2 Lipschitz Continuity and Contraction Mapping

We first give the definition of a Lipschitz continuous function.

Definition G.2.1. Let X be a complete metric space with metric d containing a closed nonempty set Ω and let $g : \Omega \rightarrow \Omega$. The function g is said to be Lipschitz continuous with Lipschitz constant $\gamma \in \mathbb{R}$ if

$$\forall x, y \in \Omega, \quad d[g(x), g(y)] \leq \gamma d(x, y).$$

From Definition G.2.1 we differentiate the following cases.

- (i) For $0 \leq \gamma < 1$, g is a contraction mapping on Ω and γ defines the rate of convergence.
- (ii) For $\gamma = 1$, g is a nonexpansive mapping.
- (iii) For $\gamma > 1$, g is a Lipschitz continuous mapping on Ω .

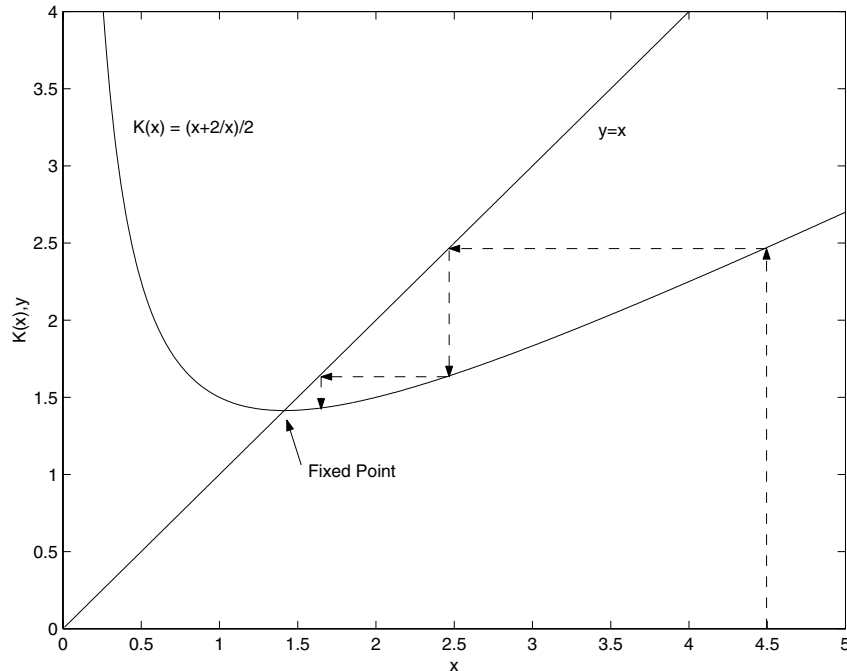


Figure G.2 Babylonian iteration

The following application of the IVT gives an important criterion for the *existence* of the fixed point.

Lemma G.2.2 (Brower's Fixed Point Theorem (Devaney 1989)). *Let $\Omega = [a, b]^N$ be a closed set of \mathbb{R}^N and $f : \Omega \rightarrow \Omega$ be a continuous vector-valued function. Then f has at least one fixed point in Ω .*

The behaviour of state trajectories (orbits) in the vicinity of fixed points defines the character of fixed points. For an asymptotically stable (or *attractive*) fixed point x^* of a function F , there exists a neighbourhood $\mathcal{O}(x^*)$ of x^* such that $\lim_{k \rightarrow \infty} F(x_k) = x^*$, for all $x_k \in \mathcal{O}(x^*)$. In this case, each eigenvalue λ of the Jacobian of F at x^* , is less than unity in magnitude. Eigenvalues of the Jacobian of F which are greater than unity in magnitude give rise to an expansion, whereas eigenvalues smaller than unity provide a contraction. In the former case of F an expansion x^* is a *repulsive* point or repeller. If some eigenvalues of the Jacobian of F are greater and some smaller than unity, x^* is called a saddle point.

G.3 Historical Perspective

Evidence from clay tablets held within the Yale University Babylonian collection suggests that iterative techniques for finding square roots go back several thousand years (Kreith and Chakerian 1999). One such clay tablet depicts how to calculate the diagonals of a square. Babylonians used a positional, base 60 number system. On this

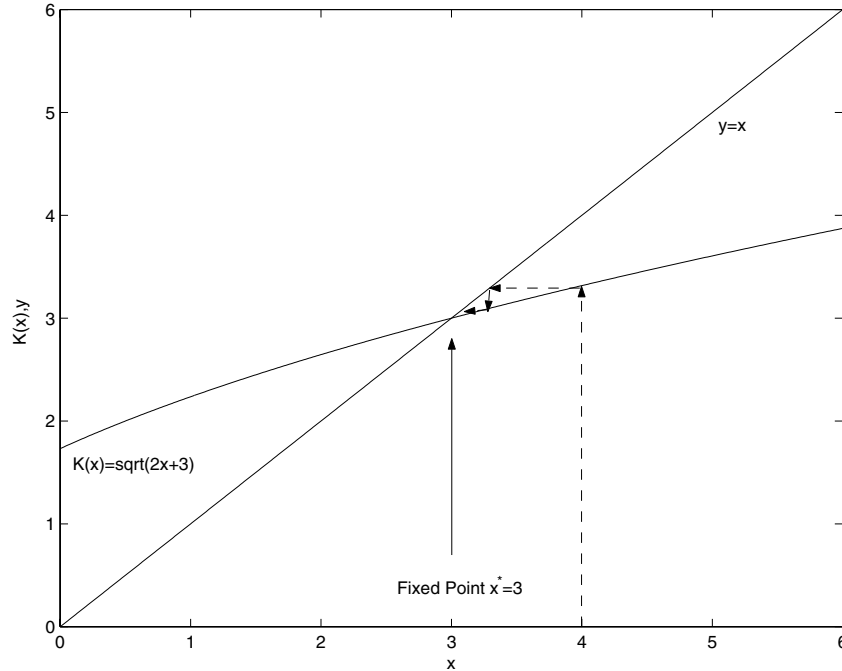


Figure G.3 FPI solution for SQRT

tablet, $30\sqrt{2}$ is calculated as $42 + \frac{25}{60} + \frac{35}{60^2} \approx 42.426\ 389$ geometrically via iteration of doubled squares.

In terms of modern mathematics, the Babylonians employed an iterator

$$x = K(x) = \frac{1}{2} \left(x + \frac{2}{x} \right) \tag{G.9}$$

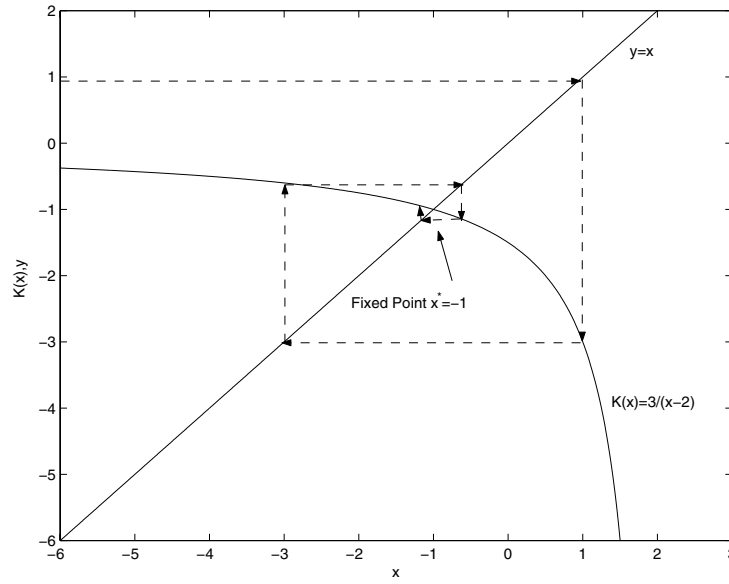
so that successive applications of $K(x)$, which start from $x_0 = 4.5$, give $x_1 = K(x_0) = \frac{3}{2}$, $x_2 = K(x_1) = \frac{17}{12}$, $x_3 = K(x_2) = \frac{577}{408}$ (Kreith and Chakerian 1999), as shown in Figure G.2. The point of intersection of curves $K(x)$ and $y = x$ can be found by solving $\frac{1}{2}(x + 2/x) = x$, which has the value $x = \sqrt{2}$ and is a *fixed point* of K and a solution of $K(x) = x$.

The next example shows how to set up a fixed point iteration.

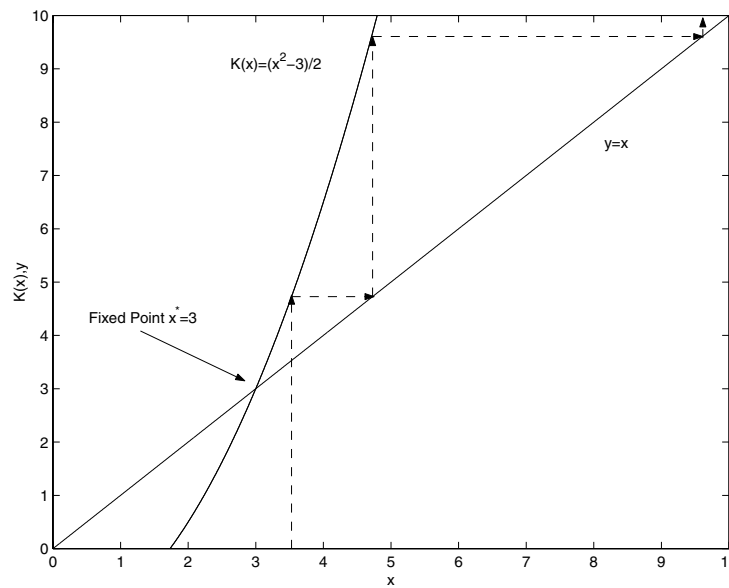
Example G.3.1. Find the roots of function $F(x) = x^2 - 2x - 3$.

Solution. Roots of function $F(x) = x^2 - 2x - 3$ can be found by rearranging $x = K(x) = \sqrt{2x + 3}$. The roots of function F are -1 and 3 . The FPI which started from $x_0 = 4$ gives the sequence of iterates $3.3116, 3.1037, 3.0344, 3.0114, 3.0038, 3.0013, 3.0004, 3.0001, 3.000$, whereby $x^* = 3$ is the fixed point and a solution of the equation. The fixed point iteration is illustrated in Figure G.3. □

Fixed point iterations for another two characteristic cases, namely an oscillatory convergence and divergence are given, respectively, in Figure G.4(a) and Figure G.4(b). For the case depicted in Figure G.4(b), the value of the first derivative



(a) Oscillatory FPI for $K(x) = 3/(x - 2)$



(b) A repeller for $K(x) = (x^2 - 3)/2$

Figure G.4 FPI for an oscillatory case and the case of a repeller

of function $K(x) = (x^2 - 3)/2$ is greater than unity for $x > x^* = 3$. Hence, by CMT (Theorem G.1.1), function K is an expansion and a fixed point iteration for starting values $x_0 > 3$ diverges. Notice, however, that an FPI with a starting value $x_0 < 3$ would converge to fixed point $x^* = 3$ by virtue of CMT. Hence, fixed point $x^* = 3$ is an unstable fixed point of function K . For the case of function $K(x) = \sqrt{2x + 3}$ depicted in Figure G.3, an FPI which starts from $x_0 < 3$ would still converge to fixed point $x^* = 3$ ($|K'(x)| < 1$), hence point $x^* = 3$ is a stable fixed point of function K .

Appendix H

Linear GAS Relaxation

H.1 Relaxation in Linear Systems

The problem of global asymptotic stability (GAS) of an N th-order time-variant difference equation,

$$y(k) = \mathbf{a}^T(k)\mathbf{y}(k-1) = a_1(k)y(k-1) + \dots + a_N(k)y(k-N), \quad (\text{H.1})$$

is important in the theory of linear systems (Barnett and Storey 1970; Golub and Van Loan 1996; Haykin 1996a; Kailath 1980; LaSalle 1986). Equation (H.1) represents an autonomous system, in fact, it is a relaxation equation (Basar and Bernhard 1995) which stems from a general linear system representation,

$$\mathbf{Y}(k+1) = \mathbf{A}(k)\mathbf{Y}(k) + \mathbf{B}(k)\mathbf{u}(k), \quad (\text{H.2})$$

for the zero exogenous input vector $\mathbf{u}(k) = \mathbf{0}$, $\forall k > 0$ (Kailath 1980; LaSalle 1986). The matrix form of Equation (H.1) now becomes

$$\begin{bmatrix} y(k+1) \\ y(k) \\ \vdots \\ y(k-N+1) \end{bmatrix} = \begin{bmatrix} a_1(k) & a_2(k) & \dots & a_N(k) \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} y(k) \\ y(k-1) \\ \vdots \\ y(k-N) \end{bmatrix}$$

or

$$\mathbf{Y}(k+1) = \mathbf{A}(k)\mathbf{Y}(k)$$

and

$$y(k+1) = [1 \ 0 \ \dots \ 0]\mathbf{Y}(k+1). \quad (\text{H.3})$$

The matrix \mathbf{A} is a Frobenius matrix, which is a special form of the companion matrix. The fundamental theorem of matrices (Horn and Johnson 1985; Wilkinson 1965) states that every matrix \mathbf{A} can be reduced by a similarity transformation to a sum of Frobenius matrices (Ciarlet 1989; Wilkinson 1965). That is why it is important to consider the stability results for the Frobenius matrix \mathbf{A} (H.3), since a stability

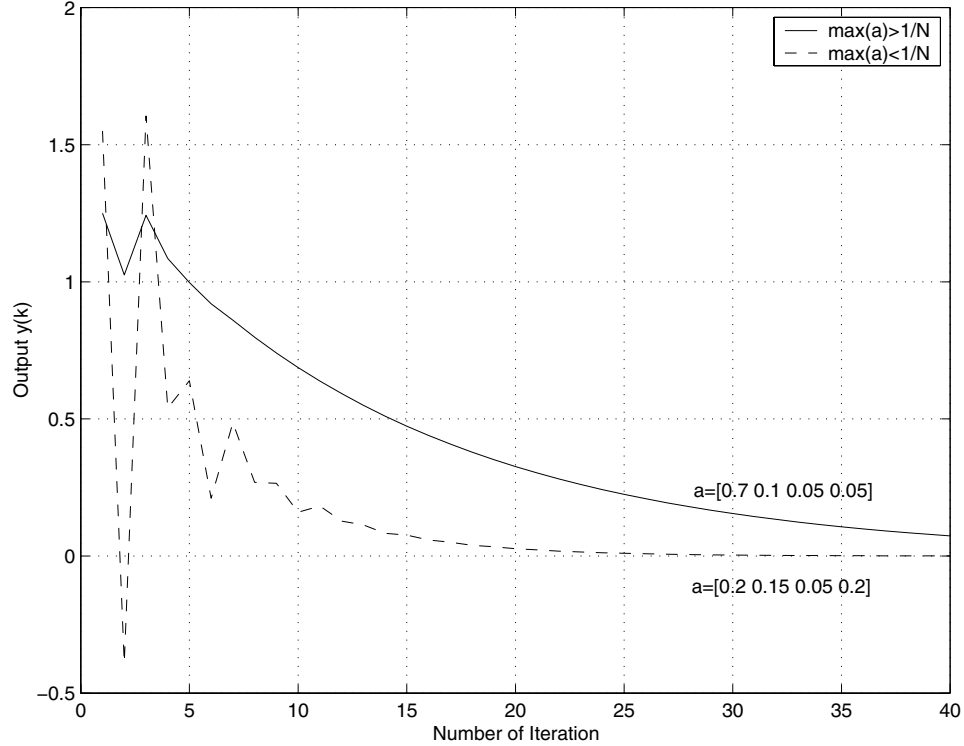


Figure H.1 The monotonic and nonmonotonic convergence

result of a general system $\mathbf{C}\mathbf{x} = \mathbf{y}$ can be obtained through the stability result of (H.3). Moreover, the analysis of convergence and stability of some learning algorithms for adaptive systems can be undertaken using this approach (Mandic and Chambers 2000a). The following analysis gives the stability conditions for relaxation (H.1), for systems with positive coefficients. The analysis of robust relaxation for nonlinear dynamical systems embarks upon this result and can be found in Mandic and Chambers (2000c). The bound on the size of the parameter vector for convergence in the CMT sense is provided below (Mandic and Chambers 2000d). In order to preserve contractivity of relaxation (H.1), we have

$$\begin{aligned}
 |y(k)| &= |a_1(k)y(k-1) + \cdots + a_N(k)y(k-N)| \\
 &\leq a_1|y(k-1)| + \cdots + a_N|y(k-N)| \\
 &< (a_1(k) + \cdots + a_N(k))|y(k-1)| \\
 &\leq N \max_{a_i(k), i=1, \dots, N} a_i(k)|y(k-1)|.
 \end{aligned} \tag{H.4}$$

From (H.4),

$$\max_{a_i(k), i=1, \dots, N} a_i(k) > \frac{1}{N} \Leftrightarrow \|\mathbf{a}(k)\|_\infty > \frac{1}{N}.$$

Figure H.1 shows two cases, the strict aperiodic convergence and a pseudoperiodic convergence. The solid line in Figure H.1 decays monotonically towards zero for lags $k > 4$. The dashed line in Figure H.1 decays in an oscillatory way towards zero for lags $k > 4$. \square

Observation H.1.1. *The system (H.3) with the constraint $\|\mathbf{a}(k)\|_1 < 1$ converges towards zero in the fixed point iteration (FPI) sense, exhibiting linear convergence, with the convergence rate $\|\mathbf{a}(k)\|$.*

This is straightforward to show using results from Appendix G.

H.1.1 Stability Result for $\sum_{i=1}^m a_i = 1$

Let us consider the case with constant parameter vector $\mathbf{a} = [a_1, \dots, a_N]^T$, where $\|\mathbf{a}\|_1 = 1$. In that case, matrix \mathbf{A} from (H.3) becomes a stochastic matrix (Golub and Van Loan 1996; Lipschutz 1965; Stark and Woods 1986), since each of its rows is a probability vector, and process (H.3) can be rewritten as

$$\mathbf{Y}(k+1) = \mathbf{A}\mathbf{Y}(k) = \mathbf{A}^2\mathbf{Y}(k-1) = \dots = \mathbf{A}^k\mathbf{Y}(0), \quad (\text{H.5})$$

whereby the dynamics of (H.3) are fully described by its initial state $\mathbf{Y}(0)$ and the system matrix \mathbf{A} . Since the product of two stochastic matrices is a stochastic matrix, there is a unique fixed vector $\mathbf{t} = [t_1, \dots, t_N]^T$ such that (Golub and Van Loan 1996; Lipschutz 1965)

$$\mathbf{t}\mathbf{A} = \mathbf{t}. \quad (\text{H.6})$$

Vector \mathbf{t} is a probability vector, i.e. $\|\mathbf{t}\|_1 = 1$. Therefore, FPI gives

$$\mathbf{A}^k = \begin{bmatrix} a_1 & a_2 & \dots & a_N \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 \end{bmatrix}^k \xrightarrow{k \rightarrow \infty} \begin{bmatrix} t_1 & t_2 & \dots & t_N \\ t_1 & t_2 & \dots & t_N \\ \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & \dots & t_N \end{bmatrix}. \quad (\text{H.7})$$

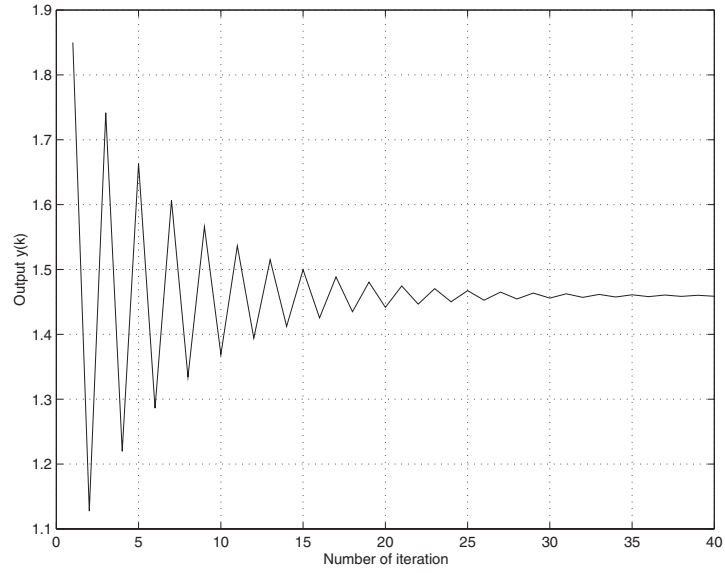
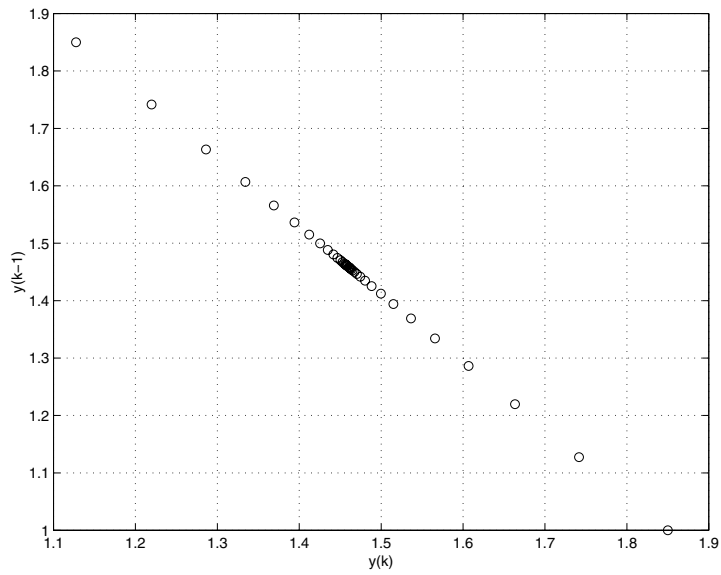
Observation H.1.2 (see Mandic and Chambers 2000d). *The process (H.1) with the constant nonnegative coefficient vector $\mathbf{a} = [a_1, \dots, a_N]^T$, converges to*

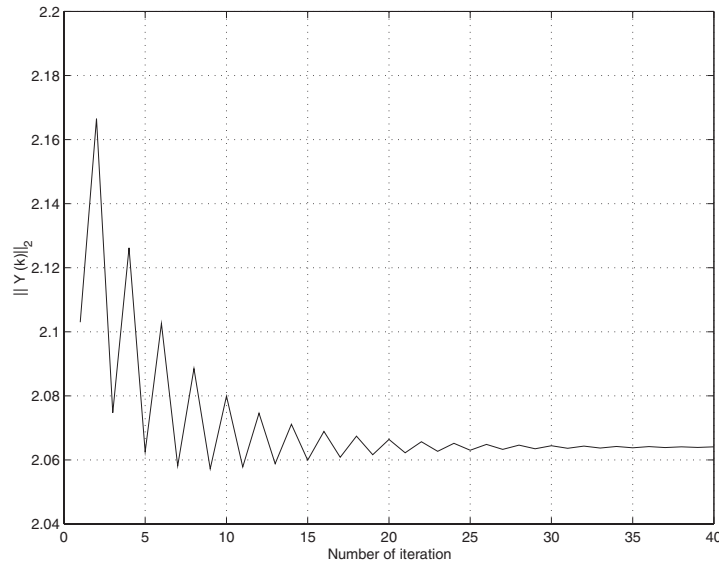
- (i) $|y_\infty| = \left| \sum_{i=1}^N t_i y(k-i) \right| \geq 0$ for $\|\mathbf{a}\|_1 = 1$,
- (ii) $y_\infty = 0$ for $\|\mathbf{a}\|_1 < 1$,

from any finite initial state $\mathbf{Y}(0)$.

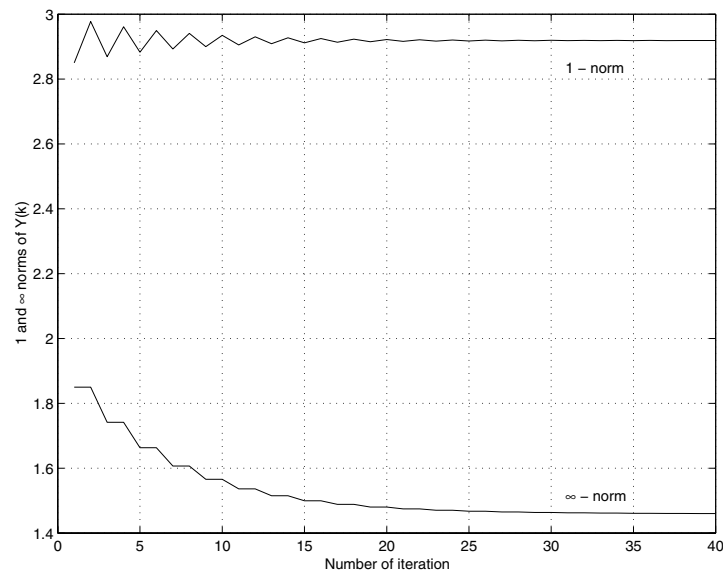
H.2 Examples

We next present some results which depict the need for a strict analysis of relaxation type equations. The initial condition for all examples below was $\mathbf{Y} = [1 \ 2]^T$.

(a) The output $y(k)$ (b) Convergence of \mathbf{Y} in the phase plane**Figure H.2** Convergence of the process $\mathbf{Y}(k) = \mathbf{A}\mathbf{Y}(k-1)$

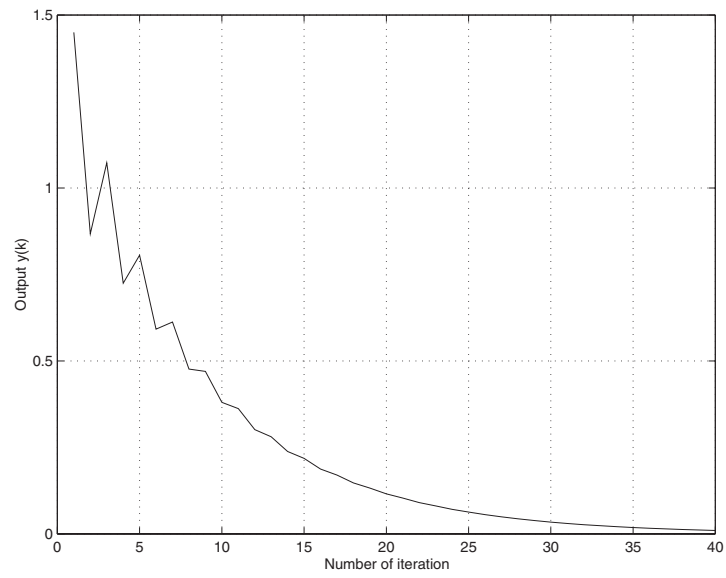
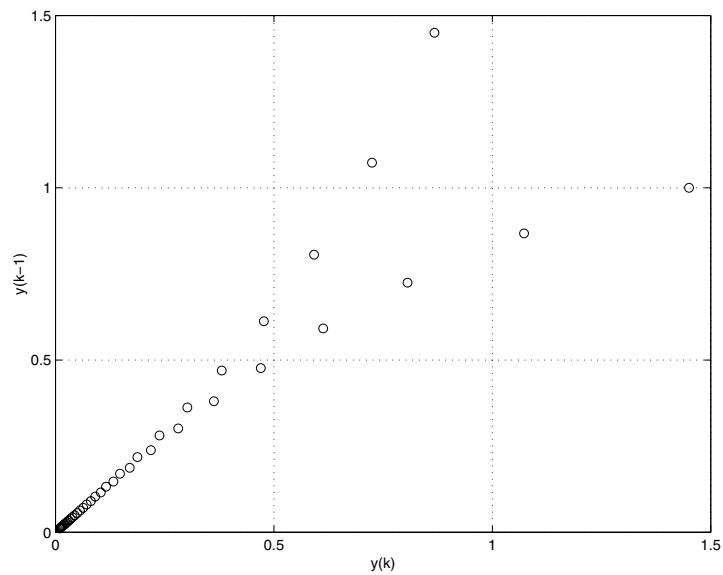


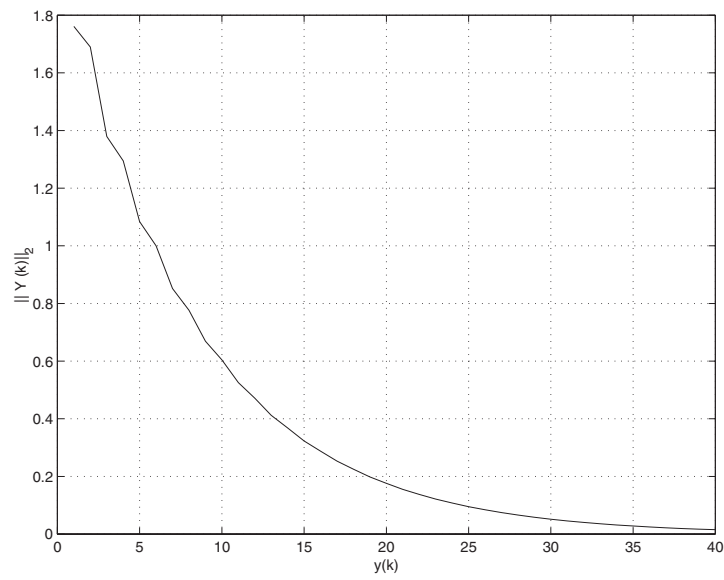
(c) Convergence of \mathbf{Y} in the $\|\cdot\|_2$ norm



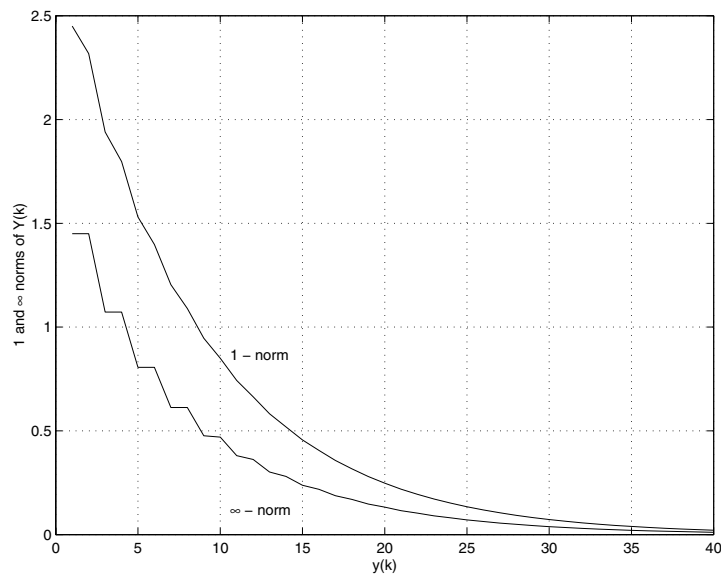
(d) Convergence of \mathbf{Y} in the $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norm

Figure H.2 *Cont.*

(a) The output $y(k)$ (b) Convergence of \mathbf{Y} in the phase plane**Figure H.3** Convergence of the process $\mathbf{Y}(k) = \mathbf{A}\mathbf{Y}(k-1)$

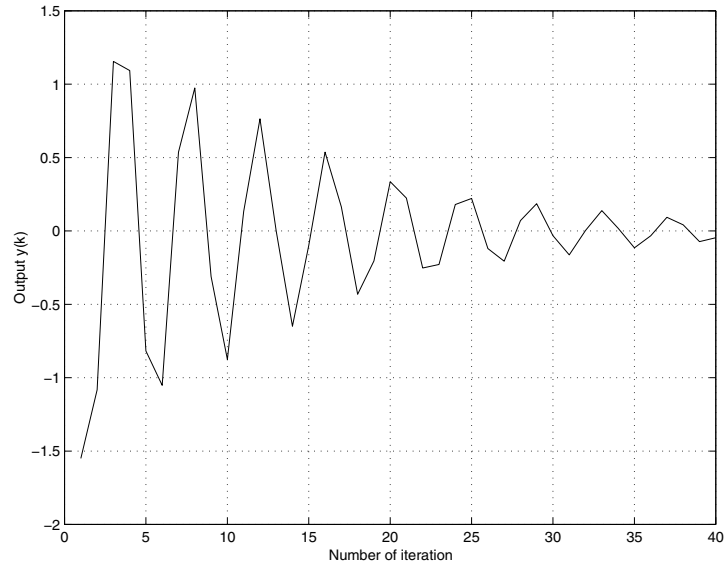
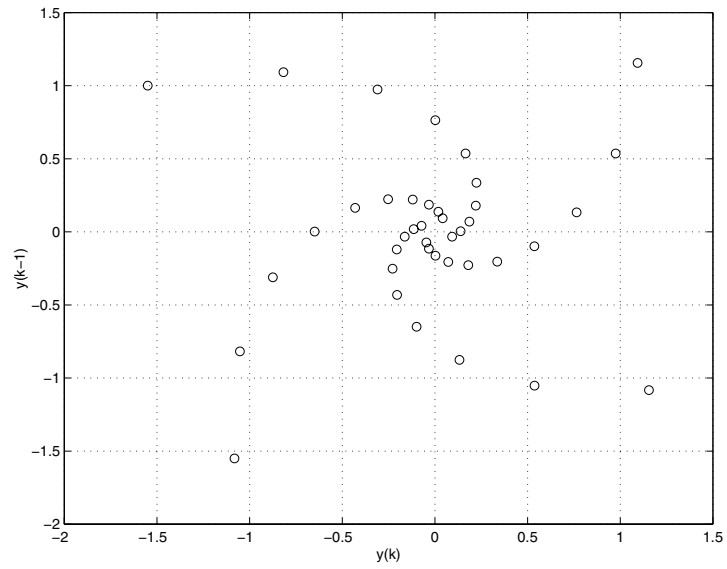


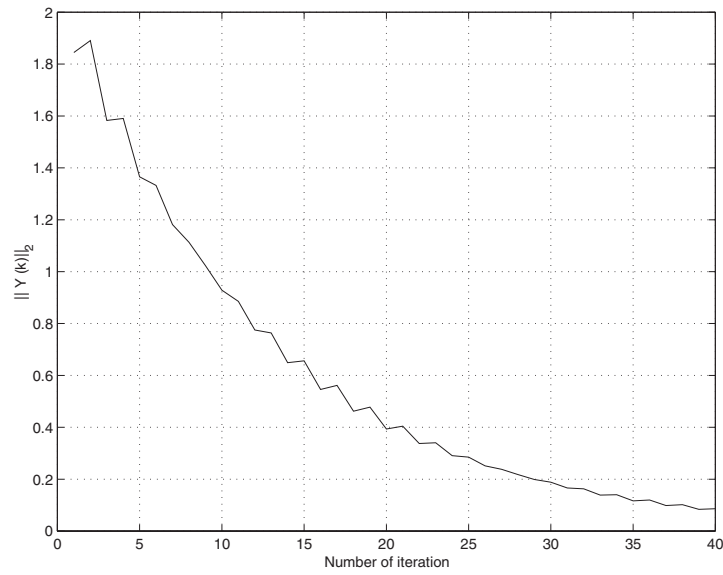
(c) Convergence of \mathbf{Y} in the $\|\cdot\|_2$ norm



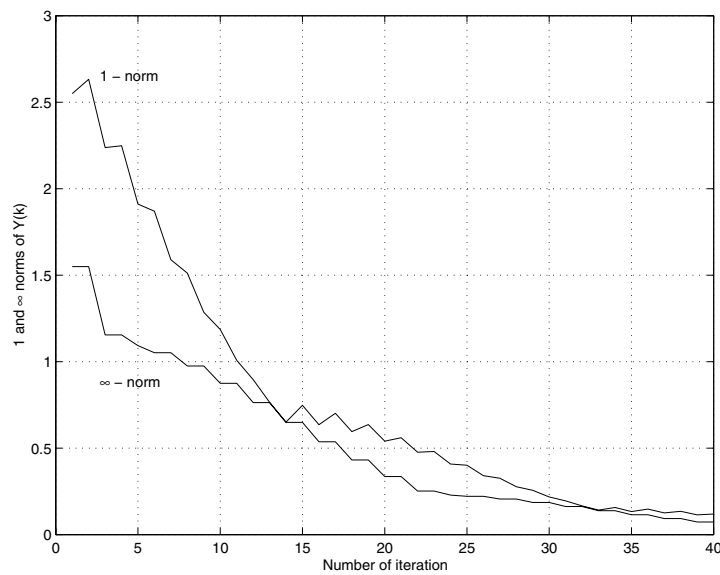
(d) Convergence of \mathbf{Y} in the $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norm

Figure H.3 *Cont.*

(a) The output $y(k)$ (b) Convergence of \mathbf{Y} in the phase plane**Figure H.4** Convergence of the process $\mathbf{Y}(k) = \mathbf{A}\mathbf{Y}(k-1)$

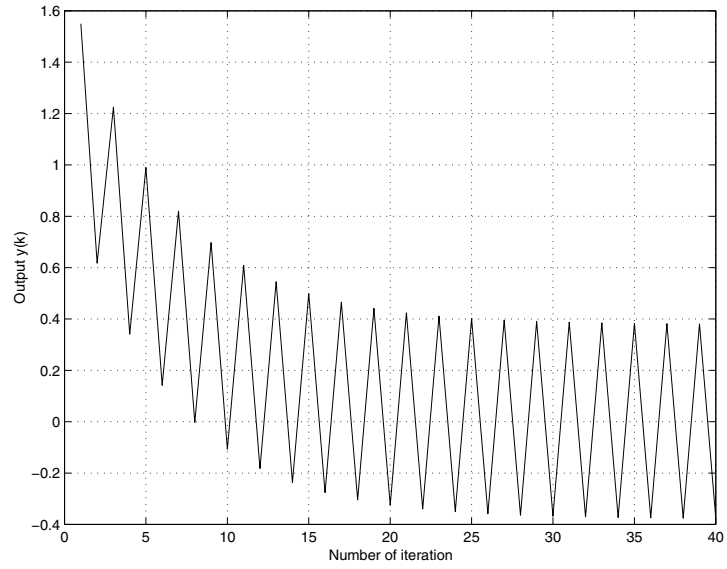
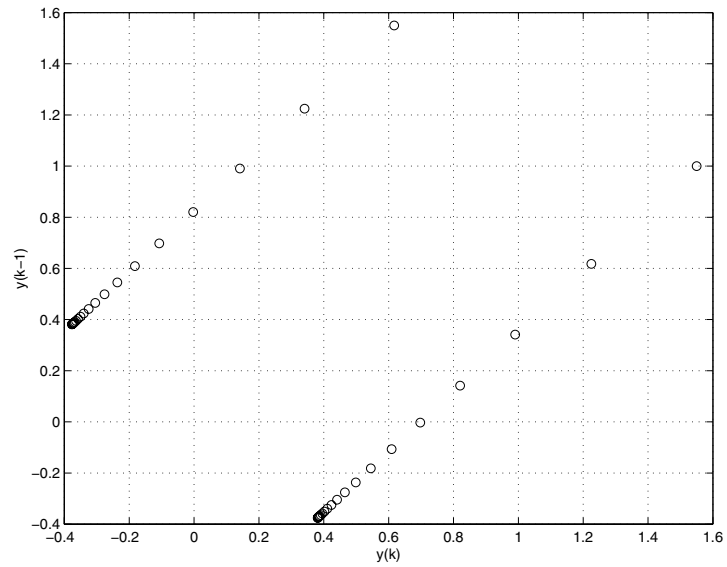


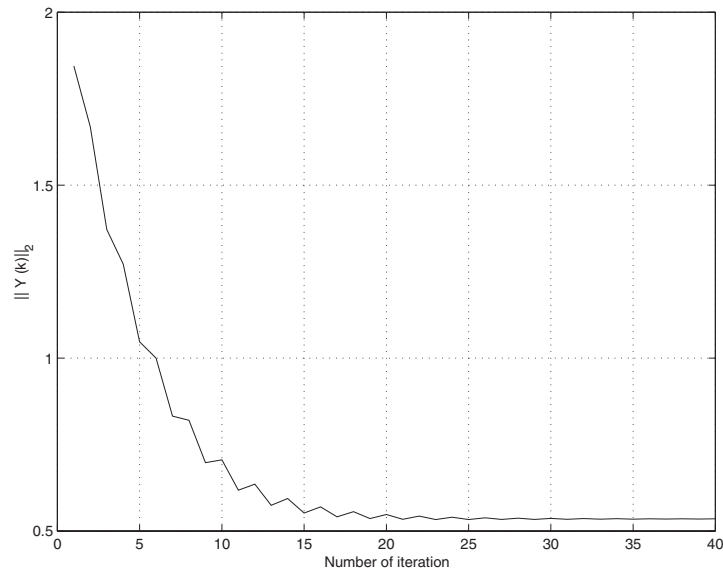
(c) Convergence of \mathbf{Y} in the $\|\cdot\|_2$ norm



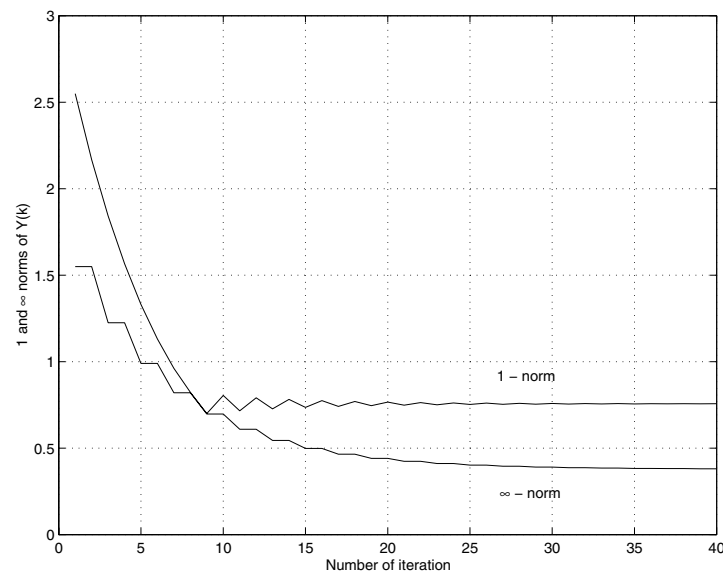
(d) Convergence of \mathbf{Y} in the $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norm

Figure H.4 *Cont.*

(a) The output $y(k)$ (b) Convergence of \mathbf{Y} in the phase plane**Figure H.5** Convergence of the process $\mathbf{Y}(k) = \mathbf{A}\mathbf{Y}(k-1)$



(c) Convergence of \mathbf{Y} in the $\|\cdot\|_2$ norm



(d) Convergence of \mathbf{Y} in the $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norm

Figure H.5 *Cont.*

- (a) For a system described by coefficients $a_1 = 0.15$, $a_2 = 0.85$, the relaxation is shown in Figure H.2. The output converges oscillatorily towards a point, as shown in Figure H.2(a). The geometric convergence of points $\mathbf{Y}(k)$ in the phase plane is oscillatory towards a point, forming a line in the plane (Figure H.2(b)). The 2 norm, shown in Figure H.2(c) and the 1 and ∞ norm, shown in Figure H.2(d) all exhibit a relative behaviour.
- (b) For a system described by coefficients $a_1 = 0.15$, $a_2 = 0.65$, the corresponding relaxation diagrams are shown in Figure H.3. Here, $\|\mathbf{a}\|_1 < 1$, and the elements of \mathbf{a} are positive. Relaxation in the 1 and 2 norm is aperiodic, whereas the output and phase plane convergence are oscillatory.
- (c) For a system described by coefficients $a_1 = 0.15$, $a_2 = -0.85$, the relaxation diagrams are shown in Figure H.4. Here, $\|\mathbf{a}\|_1 = 1$, but not all the elements of \mathbf{a} are positive. There is a clear difference between the relaxation behaviour of the output and the phase space diagram (Figure H.4(a) and H.4(b)) and the convergence in the norm (Figure H.4(c) and H.4(d)).
- (d) For a system described by coefficients $a_1 = -0.15$, $a_2 = 0.85$, the convergence is shown in Figure H.5. Here, $\|\mathbf{a}\|_1 = 1$, but not all elements of \mathbf{a} are nonnegative. The process (H.1) converges in all the norms (Figure H.5(c) and H.5(d)), but not in either the output or the geometric sense in the phase plane (Figure H.5(a) and H.5(b)), where it achieves its limit cycle, for there are two distinct points, with the same norm, to which the process converges.

Hence the following observation.

Observation H.2.1. *For the system (H.1), with $\|\mathbf{a}\|_1 = 1$, convergence in the norm does not imply convergence in the geometric sense.*

These situations depict the need for the use of alternative techniques for the analysis of relaxation and learning algorithms for nonlinear systems, such as the use of CMT and FPI as introduced in Chapter 7 and Appendix G (Mandic 2000b,c).

Appendix I

The Main Notions in Stability Theory

The roots of stability theory in physics and engineering can be traced down to Alexander Mikhailovitch Lyapunov (1857–1918). He developed the so-called ‘Lyapunov’s second method’ in his PhD thesis ‘*The General Problem of the Stability of Motion*’ in 1892¹. The following are definitions of notions from the stability theory. They are similar to those given in Sastry and Bodson (1989), but rearranged and updated due to the specific requirements of this work. Let us consider the following differential equation:

$$\dot{\mathbf{x}} = f(t, \mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0. \quad (\text{I.1})$$

Definition I.1.2 (autonomous systems). The system defined by (I.1) is called autonomous or time-invariant, if f does not depend on t .

Definition I.1.3 (linearity). The system is said to be linear if $f(t, \mathbf{x}) = A(t)\mathbf{x}$ for some $A : \mathbb{R}^+ \rightarrow \mathbb{R}^{n \times n}$.

The properties of systems defined on a closed ball B_h with radius h centred at $\mathbf{0} \in \mathbb{R}^n$, are considered

- (i) *locally*, if true for all $\mathbf{x}_0 \in B_h$,
- (ii) *globally*, if true for all $\mathbf{x}_0 \in \mathbb{R}^n$,
- (iii) *in any closed ball*, if true for all $\mathbf{x}_0 \in B_h$, with h arbitrary,
- (iv) *uniformly*, if true for all $t_0 > 0$.

Definition I.1.4 (Lipschitz function). The function f is said to be Lipschitz in \mathbf{x} if, for some $h > 0$, there exists $l \geq 0$ such that

$$\|f(t, \mathbf{x}_1) - f(t, \mathbf{x}_2)\| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\| \quad (\text{I.2})$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in B_h$, $t \geq 0$. The constant L is called the *Lipschitz constant*.

¹ Lyapunov was interested in the problem of equilibrium figures of a rotating liquid, which was also treated by Maclaurin, Jacobi and Laplace.

Definition I.1.5 (equilibrium). Point \mathbf{x} is called an equilibrium point of (I.1), if $f(t, \mathbf{x}) = 0$ for all $t \geq 0$.

Definition I.1.6 (stability). Point $\mathbf{x} = 0$ is called a stable equilibrium point of (I.1), if, for all $t_0 \geq 0$ and $\varepsilon > 0$, there exists $\delta(t_0, \varepsilon)$ such that

$$\|\mathbf{x}_0\| < \delta(t_0, \varepsilon) \Rightarrow \|\mathbf{x}(t)\| < \varepsilon, \quad \text{for all } t \geq t_0, \quad (\text{I.3})$$

where $\mathbf{x}(t)$ is the solution of (I.1) with initial conditions \mathbf{x}_0 and t_0 .

Definition I.1.7 (uniform stability). Point $\mathbf{x} = 0$ is called a uniformly stable equilibrium point of (I.1) if, in the preceding definition, δ can be chosen independent of t_0 .

In other words, the equilibrium point is not growing progressively less stable with time.

Definition I.1.8 (asymptotic stability). Point $\mathbf{x} = 0$ is called an asymptotically stable equilibrium point if

- (a) $\mathbf{x} = 0$ is a stable equilibrium point,
- (b) $\mathbf{x} = 0$ is attractive, i.e. for all $t_0 \geq 0$, there exists $\delta(t_0)$, such that

$$\|\mathbf{x}_0\| < \delta \Rightarrow \lim_{t \rightarrow \infty} \|\mathbf{x}(t)\| = 0. \quad (\text{I.4})$$

Definition I.1.9 (uniform asymptotic stability). Point $\mathbf{x} = 0$ is called a uniformly asymptotically stable equilibrium point of (I.1) if

- (a) $\mathbf{x} = 0$ uniformly stable,
- (b) trajectory $\mathbf{x}(t)$ converges to 0 uniformly in t_0 .

The previous definitions are local, since they concern neighbourhoods of the equilibrium point. The following definition treats global asymptotic stability (GAS).

Definition I.1.10 (global asymptotic stability). Point $\mathbf{x} = 0$ is called a globally asymptotically stable equilibrium point of (I.1), if it is asymptotically stable and $\lim_{t \rightarrow \infty} \|\mathbf{x}(t)\| = 0$, for all $\mathbf{x}_0 \in \mathbb{R}^n$.

Global uniform asymptotic stability is defined likewise.

Definition I.1.11 (exponential stability). Point $\mathbf{x} = 0$ is called an exponentially stable equilibrium point of (I.1) if there exist $m, \alpha > 0$, such that the solution $\mathbf{x}(t)$ satisfies

$$\|\mathbf{x}(t)\| \leq m e^{-\alpha(t-t_0)} \|\mathbf{x}_0\| \quad (\text{I.5})$$

for all $\mathbf{x}_0 \in B_h, t \geq t_0 \geq 0$. The constant α is called the rate of convergence.

By GAS we mean that the system is stable for any $\mathbf{x}_0 \in \mathbb{R}^n$. Exponential stability in any closed ball is similar except that m and α may be functions of h . Exponential stability is uniform with respect to t_0 .

Appendix J

Deseasonalising Time Series

Forecasting methods which do not account for seasonal characteristics are likely to perform poorly when applied to time series that contain such components. We present three different methods of deseasonalising time series – each method being appropriate for different forms of seasonal behaviour.

The first method assumes that the seasonal component of the time series is constant. The series is written as

$$x_k = m_k + s_k + y_k, \quad (\text{J.1})$$

where m_k is the trend component, s_k is the seasonal component and y_k is the random noise component. It is desirable to remove the trend and seasonal components and then, if the remaining random noise component is linear and stationary, it can be fitted by an ARIMA model. Denote the length of the time series by N and the period of the seasonal component by d . In the air pollutant series, for instance, two seasonal effects would be expected; one with a period of a day (24 hours), the other with period of a year ($365 \times 24 = 8760$ hours).

A moving average is applied to the series to eliminate the seasonal component. If d is even let $d = 2q$, otherwise let $d = 2q + 1$. Let

$$\begin{aligned} \hat{m}_k &= (0.5x_{k-q} + x_{k-q+1} + \cdots + x_{k+q-1} + 0.5x_{k+q}) && \text{if } d \text{ is even,} \\ \hat{m}_k &= (x_{k-q} + x_{k-q+1} + \cdots + x_{k+q-1} + x_{k+q}) && \text{if } d \text{ is odd.} \end{aligned}$$

For each hour, the seasonal component can be estimated by the average deviation

$$w_c = \frac{1}{\sum_{k=1}^N I((k-c) \bmod d = 0)} \sum_{k=1}^N I((k-c) \bmod d = 0)(x_k - \hat{m}_k),$$

where $c = 1, \dots, d$ and $I(\cdot)$ is the indicator function, $I(x) = 1$ if x is true, otherwise $I(x) = 0$. To ensure the seasonal components s_c sum to zero, they are calculated as

$$\hat{s}_c = w_c - \frac{1}{d} \sum_{i=1}^d w_i$$

for $c = 1, \dots, d$. The series \hat{s}_c is replicated approximately N/d times, i.e. until it is of the same length as x_k , and the deseasonalised series is defined to be

$$d_k = x_k - \hat{s}_k.$$

The trend of the deseasonalised series d_k can now be estimated simply with a polynomial fit via least squares.

The second method of deseasonalising differs from the previous one in that it is no longer assumed that the seasonality component is constant. This is known as SARIMA modelling, or seasonal ARIMA (Box and Jenkins 1976). In terms of a season with period 24 hours, it is reasonable to expect that the behaviour of the series at a particular time is closely related to the behaviour of the series at the same time during the previous day, as well as being related to the behaviour of the series during the hour immediately preceding. Thus, a multiplicative model is used, involving a component which is a standard ARIMA model with parameters p , d and q , and a seasonal component, given by

$$z_k = \sum_{i=1}^P a_i z_{k-i*S} + \sum_{i=1}^Q b_i \epsilon_{k-i*S} + \epsilon_k, \quad (\text{J.2})$$

where $z_k = \nabla^D x_k$. This model is written as

$$(p, d, q) \times (P, D, Q)_S,$$

where S is the period of seasonality.

Finally, when it is suspected that there is an approximately sinusoidal seasonal behaviour present in the time series – a likely scenario in air pollutant measurements given the interaction between climate and air pollutants and the known result that temperature approximately follows a sine wave with period one year – the sinusoidal component can be removed directly. This is done via the following filter

$$y_k = (1 - aB + B^2)x_k, \quad (\text{J.3})$$

where $a = 2 \cos(2\pi/b)$ and $b = 24 \times 365$, the number of measurements taken in one year; B is the *backshift* operator, defined by

$$B^j(x_k) = x_{k-j}.$$

The result (J.3) follows from the use of the \mathcal{Z} -transform to model the sine wave and find its inverse, details of which can be found in many signal processing texts (see, for example, Vaidyanathan and Mitra 1987).

References

- Agarwal A and Mammone RJ 1993 Long-term memory for neural networks. In *Artificial Neural Networks for Speech and Vision* (ed. Mammone RJ), pp. 171–193. Chapman & Hall.
- Aguirre LA 2000 A nonlinear dynamical approach to system identification. *IEEE Circuits Systems Soc. Newslett.* **1**(2), 10–23.
- Ahmad S, Tesauro G and He Y 1990 Asymptotic convergence of backpropagation: numerical experiments. In *Advances in Neural Information Processing Systems 2* (ed. Touretzky DS), pp. 606–613. Morgan Kaufmann.
- Alippi C and Piuri V 1996 Experimental neural networks for prediction and identification. *IEEE Trans. Instrument. Measurement* **45**, 670–676.
- Al-Ruwaihi KM 1997 CMOS analogue neurone circuit with programmable activation functions utilising MOS transistors with optimised process/device parameters. *IEE Proc. Circuits Devices Systems* **144**, 318–322.
- An PE, Brown M and Harris CJ 1994 Aspects of instantaneous on-line learning rules. In *Proc. Int. Conf. on Control*, vol. I, pp. 646–651.
- Anderson BDO, Bitmead RR, Johnson Jr CR, Kokotovic PV, Kosut RL, Mareels IMY, Praly L and Riedle BD 1986 *Stability of Adaptive Systems: Passivity and Averaging Analysis*. MIT Press.
- Apostol TM 1997 *Modular Functions and Dirichlet Series in Number Theory*, 2nd edn. Springer.
- Arena P, Fortuna L, Muscato G and Xibilia MG 1998a *Neural Networks in Multidimensional Domains: Fundamentals and New Trends in Modelling and Control*. Springer.
- Arena P, Fortuna L and Xibilia MG 1998b On the capability of neural networks. In *Proc. IEEE Int. Conf. on Circuits and Systems*, vol. 4, pp. 1271–1274.
- Aristotle 1975 *Posterior Analytics* (translated in English by J Barnes). Clarendon Press, Oxford.
- Attali JG and Pages G 1997 Approximations of functions by a multilayer perceptron: a new approach. *Neural Networks* **10**, 1069–1081.
- Bailer-Jones CA, MacKay DJC and Whithers PJ 1998 A recurrent neural network for modelling dynamical systems. *Network: Computation in Neural Systems* **9**, 531–547.
- Baldi P and Atiya AF 1994 How delays affect neural dynamics and learning. *IEEE Trans. Neural Networks* **5**, 612–621.
- Baltersee J and Chambers JA 1998 Non-linear adaptive prediction of speech signals using a pipelined recurrent neural network. *IEEE Trans. Signal Processing* **46**, 2207–2216.
- Barnett S and Storey C 1970 *Matrix Methods in Stability Theory*. Nelson.
- Barron AR 1993 Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Information Theory* **39**, 930–945.
- Basaglia A, Fornaciari W and Salice F 1995 Behaviour-driven minimal implementation of digital ANNs. In *Proc. Int. Conf. on Neural Networks*, vol. 4, pp. 1644–1649.

- Basar T and Bernhard P 1995 H^∞ -Optimal Control and Relaxed Minimax Design Problems. Birkhäuser.
- Bauer P, Mansour M and Duran J 1993 Stability of polynomials with time-varying coefficients. *IEEE Trans. Circuits and Systems. I. Fundamental Theory and Applications* **40**, 423–426.
- Beiu V 1998 On Kolmogorov's superpositions and Boolean functions. In *Proc. 5th Brazilian Symp. on Neural Networks*, pp. 55–60.
- Bellman R 1961 *Adaptive Control Processes: A Guided Tour*. Oxford University Press.
- Bengio Y 1995 *Neural Networks for Speech and Sequence Recognition*. International Thomson Publishing.
- Bengio Y, Simard P and Frasconi P 1994 Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks* **5**, 157–166.
- Benvenuto N and Piazza F 1992 On the complex backpropagation algorithm. *IEEE Trans. Signal Processing* **40**, 967–969.
- Bershad NJ 1986 Analysis of the normalised LMS algorithm with Gaussian inputs. *IEEE Trans. Acoustics Speech Signal Processing ASSP-34*:793–806.
- Bershad NJ, McLaughlin S and Cowan CFN 1990 Performance comparison of RLS and LMS algorithms for tracking a first order Markov communications channel. In *Proc. Int. Symp. on Circuits and Systems*, vol. 1, pp. 266–270.
- Bershad NJ, Shynk JJ and Feintuch PL 1993a Statistical analysis of the single-layer backpropagation algorithm. Part I. Mean weight behaviour. *IEEE Trans. Signal Processing* **41**, 573–582.
- Bershad NJ, Shynk JJ and Feintuch PL 1993b Statistical analysis of the single-layer backpropagation algorithm. Part II. MSE and classification performance. *IEEE Trans. Signal Processing* **41**, 583–591.
- Beule D, Herzel H, Uhlmann E, Kruger J and Becker F 1999 Detecting nonlinearities in time series of machining processes. *Proc. American Control Conf.*, pp. 694–698.
- Bharucha-Reid AT 1976 Fixed point theorems in probabilistic analysis. *Bull. Am. Math. Soc.* **82**, 641–657.
- Billings SA 1980 Identification of nonlinear systems—a survey. *IEE Proc. D* **127**(6), 272–285.
- Billings SA and Voon WSF 1986 Correlation based model validity tests for non-linear models. *Int. J. Control* **44**, 235–244.
- Billings SA, Jamaluddin HB and Chen S 1992 Properties of neural networks with applications to modelling non-linear dynamical systems. *Int. J. Control* **55**, 193–224.
- Birkett AN and Goubran RA 1997 Nonlinear adaptive filtering with FIR synapses and adaptive activation functions. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP'97)*, pp. 3321–3324.
- Blondel VD and Tsitsiklis JN 2000 A survey of computational complexity results in systems and control. *Automatica* **36**, 1249–1274.
- Box GEP and Jenkins GM 1970 *Time Series Analysis: Forecasting and Control*. Holden-Day.
- Box GEP and Jenkins GM 1976 *Time Series Analysis: Forecasting and Control*, 2nd edn. Holden-Day.
- Brockwell PJ and Davis RA 1991 *Time Series: Theory and Methods*. Springer, New York.
- Burr DJ 1993 Artificial neural networks: a decade of progress. In *Artificial Neural Networks for Speech and Vision* (ed. Mammone RJ). Chapman & Hall.
- Casdagli MC and Weigend AS 1993 Exploring the continuum between deterministic and stochastic modeling. In *Time Series Prediction: Forecasting the Future and Understanding the Past* (ed. Weigend AS and Gershenfeld NA), pp. 347–366. Addison Wesley.
- Chambers JA, Sherliker W and Mandic DP 2000 A normalised gradient algorithm for an adaptive recurrent perceptron. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-2000)*, vol. I, pp. 396–399.

- Chang P-R and Hu J-T 1997 Optimal nonlinear adaptive prediction and modeling of MPEG video in ATM networks using pipelined recurrent neural networks. *IEEE J. Selected Areas Commun.* **15**, 1087–1100.
- Chen C and Aihara K 1997 Chaos and asymptotical stability in discrete-time neural networks. *Physica D* **104**(3–4):286–325.
- Chen S and Billings SA 1989 Representations of non-linear systems: the NARMAX model. *Int. J. Control* **49**, 1013–1032.
- Chen T 1997 The past, present, and future of neural networks for signal processing. *IEEE Signal Processing Mag.* **14**(6), 28–48.
- Chon KH and Cohen RJ 1997 Linear and nonlinear ARMA model parameter estimation using an artificial neural network. *IEEE Trans. Biomedical Engng* **44**(3), 168–174.
- Chon KH, Holstein-Rathlou N-H, Marsh DJ and Marmarelis VZ 1998 Comparative non-linear modeling of renal autoregulation in rats: Volterra approach versus artificial neural networks. *IEEE Trans. Neural Networks* **9**, 430–435.
- Chon KH, Hoyer D, Armoundas AA, Holstein-Rathlou N-H and Marsh DJ 1999 Robust non-linear autoregressive moving average model parameter estimation using stochastic recurrent artificial neural networks. *Ann. Biomed. Engng* **27**, 538–547.
- Ciarlet PG 1989 *Introduction to numerical linear algebra and optimization*. Cambridge University Press.
- Cichocki A and Unbehauen R 1993 *Neural Networks for Optimization and Signal Processing*. Wiley.
- Clarke TL 1990 Generalization of neural networks to the complex plane. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN90)*, vol. II, pp. 435–440.
- Connor J, Atlas LE and Martin DR 1992 Recurrent networks and NARMA modeling. In *Advances in Neural Information Processing Systems 4* (ed. Moody JE, Hanson SJ and Lippmann RP), pp. 301–308. Morgan Kaufmann.
- Connor JT, Martin RD and Atlas LE 1994 Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Networks* **5**, 240–254.
- Cotter NE 1990 The Stone–Weierstrass theorem and its applications to neural networks. *IEEE Trans. Neural Networks* **1**, 290–295.
- Cybenko G 1989 Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems* **2**, 303–314.
- deFigueiredo RJP 1997 Optimal neural network realisations of nonlinear FIR and IIR filters. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 709–712.
- Dennis Jr JE and Schnabel RB 1983 *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall Series in Computational Mathematics, 1983.
- DeRusso PM, Roy RJ, Close CM and Desrochers AA 1998 *State Variables for Engineers*, 2nd edn. Wiley.
- Devaney RL 1989 *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley.
- Devaney RL 1999 The Mandelbrot set and the Farey tree, and the Fibonacci sequence. *Am. Math. Mon.* **106**, 289–302.
- Diaconis P and Freedman D 1999 Iterated random functions. *SIAM Rev.* **41**, 719–741.
- Dillon RM and Manikopoulos CN 1991 Neural net nonlinear prediction for speech data. *Electron. Lett.* **27**(10), 824–826.
- Dogaru R, Murgan AT, Ortmann S and Glesner M 1996 Searching for robust chaos in discrete time neural networks using weight space exploration. In *Proc. IEEE Int. Conf. on Neural Networks*, vol. 2, pp. 688–693.
- Douglas SC 1994 A family of normalized LMS algorithms. *IEEE Signal Processing Lett.* **1**(3), 49–51.
- Douglas SC 1997 Adaptive filters employing partial updates. *IEEE Trans. Circuits and Systems. II. Analog and Digital Signal Processing* **44**, 209–216.

- Douglas SC and Pan W 1995 Exact expectation analysis of the LMS adaptive filter. *IEEE Trans. Signal Processing* **43**, 2863–2871.
- Douglas SC and Rupp M 1997 *A posteriori* updates for adaptive filters. In *Conf. Record 31st Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 1641–1645.
- Draye J-PS, Pavisic DA, Cheron GA and Libert GA 1996 Dynamic recurrent neural networks: A dynamical analysis. *IEEE Trans. Systems Man, Cybernetics. Part B. Cybernetics* **26**, 692–706.
- Dreyfus G and Idan Y 1998 The canonical form of nonlinear discrete-time models. *Neural Computation* **10**, 133–136.
- Dror G and Tsodyks M 2000 Chaos in neural networks with dynamics synapses. *Neurocomputing* **32–33**:365–370.
- Duch W and Jankowski N 1999 Survey of neural transfer functions. *Neural Computing Surv.* **2**, 163–212.
- Dumitras A and Lazarescu V 1997 On viewing the transform performed by a hidden layer in a feedforward ANN as a complex Möbius mapping. In *Proc. Int. Conf. on Neural Networks*, vol. 2, pp. 1148–1151.
- Eberhart RC 1990 Standardization of neural network terminology. *IEEE Trans. Neural Networks* **1**, 244–245.
- Elliott DL 1993 A better activation function for artificial neural networks. Tech. Rep. TR 93-8. Institute for Systems Research, University of Maryland.
- Elman JL 1990 Finding structure in time. *Cognitive Sci.* **14**, 179–211.
- Elsken T 1999 Smaller nets may perform better: special transfer functions. *Neural Networks* **12**, 627–645.
- Ermentrout B 1998 Neural systems as spatio-temporal pattern-forming systems. *Rep. Prog. Phys.* **61**, 353–430.
- Fiesler E 1994 Neural network formalization. *Computer Standards Interfaces* **16**(3), 231–239.
- Fiesler E and Beale R (eds) 1997 *Handbook of Neural Computation*. Institute of Physics Publishing and Oxford University Press.
- Forti M and Tesi A 1994 Conditions for global stability of some classes of nonsymmetric neural networks. In *Proc. 33rd Conf. on Decision and Control*, pp. 2488–2493.
- Foxall R, Krcmar I, Cawley G, Dorling S and Mandic DP 2001 On nonlinear processing of air pollution time series. In *Proc. Int. Conf. on Artificial Neural Nets and Genetic Algorithms, ICANNGA-2001*, pp. 477–480.
- Frasconi P, Gori M and Soda G 1992 Local feedback multilayered networks. *Neural Computation* **4**, 120–130.
- Fukushima K 1975 Cognitron: a self-organized multilayered neural network. *Biol. Cybernet.* **23**, 121–136.
- Funahashi K 1989 On the approximate realisation of continuous mappings by neural networks. *Neural Networks* **2**, 183–192.
- Funahashi K and Nakamura Y 1993 Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks* **6**, 801–806.
- Gemon S, Bienenstock E and Doursat R 1992 Neural networks and the bias/variance dilemma. *Neural Computation* **4**(1), 1–58.
- Gent CR and Sheppard CP 1992 Predicting time series by a fully connected neural network trained by backpropagation. *Computing Control Engng J.*, pp. 109–112.
- Georgiou GM and Koutsougeras C 1992 Complex domain backpropagation. *IEEE Trans. Circuits and Systems. II. Analog and Digital Signal Processing* **39**, 330–334.
- Gershenfeld NA and Weigend AS 1993 The future of time series: learning and understanding. In *Time Series Prediction: Forecasting the Future and Understanding the Past* (ed. Weigend AS and Gershenfeld NA), pp. 1–70. Addison-Wesley.
- Gholkar VA 1990 Mean square convergence analysis of LMS algorithm. *Electron. Lett.* **26**, 1705–1706.

- Gill PE, Murray W and Wright MH 1981 *Practical Optimization*. Academic Press, London.
- Girosi F and Poggio T 1989a Networks and the best approximation property. Tech. Rep. TR 1164. Artificial Intelligence Laboratory, Massachusetts Institute of Technology.
- Girosi F and Poggio T 1989b Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation* **1**, 465–469.
- Golub GH and Van Loan CF 1996 *Matrix Computation*, 3rd edn. The Johns Hopkins University Press.
- Gorban AN and Wunsch II DC 1998 The general approximation theorem. In *Proc. IEEE World Congress on Computational Intelligence*, vol. 2, pp. 1271–1274.
- Grossberg S 1974 Classical and instrumental learning by neural networks. *Prog. Theoret. Biol.* **3**, 51–141.
- Guarnieri S, Piazza F and Uncini A 1999 Multilayered feedforward networks with adaptive spline activation function. *IEEE Trans. Neural Networks* **10**, 672–683.
- Guo L and Ljung L 1995 Exponential stability of general tracking algorithms. *IEEE Trans. Automatic Control* **40**, 1376–1387.
- Guo L, Ljung L and Wang G-J 1997 Necessary and sufficient conditions for stability of LMS. *IEEE Trans. Automatic Control* **42**, 761–770.
- Haber R and Unbehauen H 1990 Structure identification of nonlinear dynamic systems—a survey on input/output approaches. *Automatica* **26**, 651–677.
- Hakim NZ, Kaufman JJ, Cerf G and Meadows HE 1991 Volterra characterization of neural networks. In *Conf. Record 25th Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 1128–1132.
- Han JY, Sayeh MR and Zhang J 1989 Convergence and limit points of neural network and its applications to pattern recognition. *IEEE Trans. Systems Man. Cybernetics* **19**, 1217–1222.
- Hassibi B, Sayed AH and Kailath T 1996 H^∞ optimality of the LMS algorithm. *IEEE Trans. Signal Processing* **44**, 267–280.
- Hayes M 1997 *Statistical Signal Processing*. Wiley.
- Haykin S 1994 *Neural Networks: A Comprehensive Foundation*. Prentice-Hall.
- Haykin S 1996a *Adaptive Filter Theory*, 3rd edn. Prentice-Hall.
- Haykin S 1996b Neural networks expand SP's horizons. *IEEE Signal Processing Mag.* **13**(2), 24–49.
- Haykin S 1999a Lessons on adaptive systems for signal processing, communications, and control. *IEEE Signal Processing Mag.* **16**(5), 39–48.
- Haykin S 1999b *Neural Networks: A Comprehensive Foundation*, 2nd edn. Prentice-Hall.
- Haykin S (ed.) 2000 *Unsupervised Adaptive Filtering*, vol. 1: *Blind Separation*. Wiley.
- Haykin S and Li L 1995 Nonlinear adaptive prediction of nonstationary signals. *IEEE Trans. Signal Processing* **43**, 526–535.
- Haykin S and Principe J 1998 Using neural networks to dynamically model chaotic events such as sea clutter. *IEEE Signal Processing Mag.* **15**(3), 66–81.
- Haykin S, Sayed AH, Zeidler JR, Yee P and Wei PC 1997 Adaptive tracking of linear time-variant systems by extended RLS algorithms. *IEEE Trans. Signal Processing* **45**, 1118–1128.
- Hebb DO 1949 *The Organization of Behaviour*. Wiley.
- Hecht-Nielsen R 1987 Kolmogorov's mapping neural networks existence theorem. In *Proc. Int. Joint Conf. on Neural Networks*, vol. 3, pp. 11–14.
- Helmke U and Williamson RC 1995 Neural networks, rational functions, and realisation theory. *Math. Control Signals Systems* **8**, 27–49.
- Hertz J, Krogh A and Palmer RG 1991 *Introduction to the Theory of Neural Computation*. Addison-Wesley.
- Hilbert D 1901–1902 Mathematical problems. *Bull. Am. Math. Soc.* **8**, 437–479.

- Hlavacek I and Krizek M 1998 *A posteriori* error estimates for three-dimensional axisymmetric elliptic problems. In *Finite Element Methods* (ed. Krizek M, Neittaanmaki P and Stenberg R), pp. 147–154. Marcel Dekker.
- Hoehfeld M and Fahlman SE 1992 Learning with limited numerical precision using the cascade-correlation algorithm. *IEEE Trans. Neural Networks* **3**, 602–611.
- Hopfield JJ and Tank DW 1985 ‘Neural’ computation of decisions in optimisation problems. *Biol. Cybernetics* **52**, 141–152.
- Horn RA and Johnson CA 1985 *Matrix Analysis*. Cambridge University Press.
- Hornik K 1990 Approximation capabilities of multilayer feedforward networks. *Neural Networks* **4**, 251–257.
- Hornik K 1993 Some new results on neural network approximation. *Neural Networks* **8**, 1069–1072.
- Hornik K, Stinchcombe M and White H 1989 Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359–366.
- Hornik K, Stinchcombe M and White H 1990 Universal approximation of an unknown mapping and its derivatives using multilayer feedforward network. *Neural Networks* **3**, 551–560.
- Hunt KJ, Sbarbaro D, Zbikowski R and Gawthrop PJ 1992 Neural networks for control systems—a survey. *Automatica* **28**, 1083–1112.
- Hwang K and Briggs FA 1986 *Computer Architectures and Parallel Processing*. McGraw-Hill.
- Ibnkahla M, Bershada NJ, Sombrin J and Castanie F 1998 Neural network modelling and identification of nonlinear channels with memory: algorithms, applications, and analytic models. *IEEE Trans. Signal Processing* **46**, 1208–1220.
- Iiguni Y and Sakai H 1992 A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter. *IEEE Trans. Signal Processing* **40**, 959–966.
- Jacobs RA 1988 Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**, 295–307.
- Jenkins WK, Hull AW, Strait JC, Schnauffer BA and Li X 1996 *Advanced Concepts in Adaptive Signal Processing*. Kluwer.
- Jin L and Gupta MM 1996 Globally asymptotical stability of discrete-time analog neural networks. *IEEE Trans. Neural Networks* **7**, 1024–1031.
- Jin L, Nikiforuk PN and Gupta MM 1994 Absolute stability conditions for discrete-time recurrent neural networks. *IEEE Trans. Neural Networks* **5**, 954–964.
- Jones LK 1990 Constructive approximations for neural networks by sigmoidal functions. *Proc. IEEE* **78**, 1586–1589.
- Jordan MI 1986 Serial order: a parallel distributed processing approach. Tech. Rep. TR-8604, UC San Diego, Institute for Cognitive Science.
- Jury EI 1978 Stability of multidimensional scalar and matrix polynomials. *Proc. IEEE* **66**, 1018–1047.
- Kailath T 1980 *Linear Systems*. Prentice-Hall.
- Kailath T, Sayed AH and Hassibi B 2000 *Linear Estimation*. Prentice-Hall.
- Kainen PC, Kurkova V and Vogt A 1999 Approximation by neural networks is not continuous. *Neurocomputing* **29**, 47–56.
- Kang HW, Cho YS and Youn DH 1998 Adaptive precompensation of Wiener systems. *IEEE Trans. Signal Processing* **46**, 2825–2829.
- Katsuura H and Sprecher DA 1994 Computational aspects of Kolmogorov’s superposition theorem. *Neural Networks* **115**, 455–461.
- Kay SM 1993 *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice-Hall International.
- Khalaf AAM and Nakayama K 1998 A cascade form predictor of neural and FIR filters and its minimum size estimation based on nonlinearity analysis of time series. *IEICE Trans. Fundamentals E* **81A**(3), 364–373.

- Khalaf AAM and Nakayama K 1999 A hybrid nonlinear predictor: analysis of learning process and predictability for noisy time series. *IEICE Trans. Fundamentals E* **82A**(8), 1420–1427.
- Khotanzad A and Lu J 1990 Non-parametric prediction of AR processes using neural networks. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-90)*, pp. 2551–2555.
- Kim MS and Guest CC 1990 Modification of backpropagation networks for complex-valued signal processing in frequency domain. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN90)*, vol. III, pp. 27–31.
- Kim S-S 1998 Time-delay recurrent neural network for temporal correlations and prediction. *Neurocomputing* **20**, 253–263.
- Kirkpatrick S, Gelatt CD and Vecchi MP 1983 Optimization by simulated annealing. *Science* **220**, 671–680.
- Kohonen T 1982 Self-organized formation of topologically correct feature maps. *Biol. Cybernetics* **43**, 59–69.
- Kolmogorov AN 1941 Interpolation and extrapolation von stationären zufällen. *Bull. Acad. Sci. (Nauk)* **5**, 3–14.
- Kolmogorov AN 1957 On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR* **114**, 953–956.
- Kosmatopoulos EB, Polycarpou MM, Christodoulou MA and Ioannou PA 1995 High-order neural network structures for identification of dynamical systems. *IEEE Trans. Neural Networks* **6**, 422–431.
- Krcmar IR and Mandic DP 2001 A fully adaptive NNGD algorithm. Accepted for ICASSP-2001.
- Krcmar IR, Bozic MM and Mandic DP 2000 Global asymptotic stability of RNNs with bipolar activation functions. In *Proc. 5th IEEE Seminar on Neural Networks Applications in Electrical Engineering (NEUREL2000)*, pp. 33–36.
- Kreinovich V, Nguyen HT and Yam Y 2000 Fuzzy systems are universal approximators for a smooth function and its derivatives. *Int. J. Intellig. Syst.* **15**, 565–574.
- Kreith K and Chakerian D 1999 *Iterative Algebra and Dynamic Modeling*. Springer.
- Kruschke JK and Movellan JR 1991 Benefits of gain: speeded learning and minimal hidden layers in back-propagation networks. *IEEE Trans. Syst. Man, Cybernetics* **21**(1), 273–280.
- Kuan C-M and Hornik K 1991 Convergence of learning algorithms with constant learning rates. *IEEE Trans. Neural Networks* **2**, 484–489.
- Kung S-Y and Hwang J-N 1998 Neural networks for intelligent multimedia processing. *Proc. IEEE* **86**, 1244–1272.
- Kurkova V 1992 Kolmogorov's theorem and multilayer neural networks. *Neural Networks* **5**, 501–506.
- Kushner HJ 1984 *Approximation and Weak Convergence Methods for Random Processes with Applications to Stochastic System Theory*. MIT Press.
- LaSalle JP 1986 *The Stability and Control of Discrete Processes*. Springer.
- Leontaritis IJ and Billings SA 1985 Input–output parametric models for nonlinear systems. *Int. J. Control* **41**, 303–344.
- Leshno M, Lin VY, Pinkus A and Schocken S 1993 Multilayer feedforward neural networks with a nonpolynomial activation function can approximate any function. *Neural Networks* **6**, 861–867.
- Leung H and Haykin S 1991 The complex backpropagation algorithm. *IEEE Trans. Signal Processing* **39**, 2101–2104.
- Leung H and Haykin S 1993 Rational neural networks. *Neural Computation* **5**, 928–938.
- Li LK 1992 Approximation theory and recurrent networks. In *Proc. Int. Joint Conf. on Neural Networks*, vol. II, pp. 266–271.

- Li L and Haykin S 1993 A cascaded neural networks for real-time nonlinear adaptive filtering. In *Proc. IEEE Int. Conf. on Neural Networks (ICNN'93)*, pp. 857–862.
- Liang X-B and Yamaguchi T 1997 Necessary and sufficient conditions for absolute exponential stability of a class of nonsymmetric neural networks. *IEICE Trans. Inform. Syst.* **E80D**(8), 802–807.
- Lin T, Horne BG, Tino P and Lee Giles C 1996 Learning long-term dependencies in NARX recurrent neural networks. *IEEE Trans. Neural Networks* **7**, 1329–1338.
- Lin T-N, Lee Giles C, Horne BG and Kung S-Y 1997 A delay damage model selection algorithm for NARX neural networks. *IEEE Trans. Signal Processing* **45**, 2719–2730.
- Lippmann RP 1987 An introduction of computing with neural nets. *IEEE Acoust. Speech Signal Processing Mag.* **4**(2), 4–22.
- Lipschutz S 1965 *Theory and Problems of Probability*. Schaum's Outline Series. McGraw-Hill.
- Ljung L 1984 Analysis of stochastic gradient algorithms for linear regression problems. *IEEE Trans. Information Theory* **IT-30**(2):151–160.
- Ljung L and Sjöberg J 1992 A system identification perspective of neural nets. In *Proc. II IEEE Workshop on Neural Networks for Signal Processing (NNSP92)*, pp. 423–435.
- Ljung L and Soderstrom T 1983 *Theory and Practice of Recursive Identification*. MIT Press.
- Lo JT-H 1994 Synthetic approach to optimal filtering. *IEEE Trans. Neural Networks* **5**, 803–811.
- Lorentz GG 1976 The 13th problem of Hilbert. In *Mathematical Developments Arising from Hilbert Problems* (ed. Browder FE). American Mathematical Society.
- Lorenz EN 1963 Deterministic nonperiodic flow. *J. Atmos. Sci.* **20**, 130–141.
- Luenberger DG 1969 *Optimization by Vector Space Methods*. Wiley.
- Luh PB, Zhao X and Wang Y 1998 Lagrangian relaxation neural networks for job shop scheduling. In *Proc. 1998 IEEE Int. Conf. on Robotics and Automation*, pp. 1799–1804.
- Maass W and Sontag ED 2000 Neural systems as nonlinear filters. *Neural Computation* **12**, 1743–1772.
- McCulloch WS and Pitts W 1943 A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133.
- McDonnell JR and Waagen D 1994 Evolving recurrent perceptrons for time-series modelling. *IEEE Trans. Neural Networks* **5**, 24–38.
- Maierov V and Pinkus A 1999 Lower bounds for approximation by MLP neural networks. *Neurocomputing* **25**, 81–91.
- Makhoul J 1975 Linear prediction: a tutorial overview. *Proc. IEEE* **63**, 561–580.
- Mandic DP 2000a NNGD algorithm for neural adaptive filters. *Electron. Lett.* **36**, 845–846.
- Mandic DP 2000b On fixed points of a general neural network via Möbius transformations. In *Proc. 5th Int. Conf. on Mathematics in Signal Processing*.
- Mandic DP 2000c The use of Möbius transformations in neural networks and signal processing. In *Proc. Xth IEEE Workshop on Neural Networks for Signal Processing (NNSP2000)*, pp. 185–194.
- Mandic DP and Chambers JA 1998a Advanced PRNN based nonlinear prediction/System identification. In *Digest of the IEE Colloquium on Non-Linear Signal and Image Processing*, pp. 11/1–11/6.
- Mandic DP and Chambers JA 1998b From an *a priori* RNN to an *a posteriori* PRNN nonlinear predictor. In *Proc. VIII IEEE Workshop on Neural Networks for Signal Processing (NNSP98)*, pp. 174–183.
- Mandic DP and Chambers JA 1998c *A posteriori* real time recurrent learning schemes for a recurrent neural network based non-linear predictor. *IEE Proc. Vision Image Signal Processing* **145**, 365–370.
- Mandic DP and Chambers JA 1999a Exploiting inherent DP relationships in RNN architectures. *Neural Networks* **12**, 1341–1345.

- Mandic DP and Chambers JA 1999b Global asymptotic stability of nonlinear relaxation equations realised through a recurrent perceptron. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-99)*, vol. 2, pp. 1037–1040.
- Mandic DP and Chambers JA 1999c A nonlinear adaptive predictor realised via recurrent neural networks with annealing. In *Digest of the IEE Colloquium Statistical Signal Processing*, pp. 2/1–2/6.
- Mandic DP and Chambers JA 1999d *A posteriori* error learning in nonlinear adaptive filters. *IEE Proc. Vision, Image and Signal Processing* **146**(6), 293–296.
- Mandic DP and Chambers JA 1999e Relationship between the slope of the activation function and the learning rate for the RNN. *Neural Computation* **11**, 1069–1077.
- Mandic DP and Chambers JA 1999f Toward an optimal PRNN based nonlinear predictor. *IEEE Trans. Neural Networks* **10**, 1435–1442.
- Mandic DP and Chambers JA 2000a Advanced RNN based NARMA predictors. *J. VLSI Signal Processing Syst. Signal Image Video Technol.* **26**, 105–117.
- Mandic DP and Chambers JA 2000b A normalised real time recurrent learning algorithm. *Signal Processing* **80**, 1909–1916.
- Mandic DP and Chambers JA 2000c On robust stability of time-variant discrete-time nonlinear systems with bounded parameter perturbations. *IEEE Trans. Circuits Systems. I. Fundamental Theory and Applications* **47**, 185–188.
- Mandic DP and Chambers JA 2000d On stability of relaxive systems described by polynomials with time-variant coefficients. *IEEE Trans. Circuits Systems. I. Fundamental Theory and Applications* **47**, 1534–1537.
- Mandic DP and Chambers JA 2000e Relations between the *a priori* and *a posteriori* errors in nonlinear adaptive neural filters. *Neural Computation* **12**, 1285–1292.
- Mandic DP and Chambers JA 2000f Towards an optimal learning rate for backpropagation. *Neural Processing Lett.* **11**(1), 1–5.
- Mandic DP and Krcmar IR 2000 On training with slope adaptation for feedforward neural networks. In *Proc. 5th IEEE Seminar on Neural Networks Applications in Electrical Engineering (NEUREL2000)*, pp. 42–45.
- Mandic DP and Krcmar IR 2001 Stability of the NNGD algorithm for nonlinear system identification. *Electron. Lett.* **37**, 200–202.
- Mandic DP, Baltersee J and Chambers JA 1998 Nonlinear prediction of speech with a pipelined recurrent neural network and advanced learning algorithms. In *Signal Analysis and Prediction* (ed. Prochazka A, Uhler J, Rayner PJW and Kingsbury NG), pp. 291–309. Birkhäuser, Boston.
- Mandic DP, Chambers JA and Bozic MM 2000 On global asymptotic stability of fully connected recurrent neural networks. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-2000)*, vol. VI, pp. 3406–3409.
- Mane R 1981 On the dimension of the compact invariant sets of certain nonlinear maps. In *Dynamical Systems and Turbulence* (ed. Rand D and Young LS). Warwick 1980 Lecture Notes in Mathematics, vol. 898. Springer.
- Marmarelis VZ 1989 Signal transformation and coding in neural systems. *IEEE Trans. Biomed. Engng* **36**(1), 15–24.
- Mathews VJ 1991 Adaptive polynomial filters. *IEEE Signal Processing Mag.* **8**(3), 10–26.
- Medler DA 1998 A brief history of connectionism. *Neural Computing Surveys* **1**, 61–101.
- Mathews JH and Howell RW 1997 *Complex Analysis: for Mathematics and Engineering*, 3rd edn. Jones and Bartlett.
- Mathews VJ and Xie Z 1993 A stochastic gradient adaptive filter with gradient adaptive step size. *IEEE Trans. Signal Processing* **41**, 2075–2087.
- Mitaim S and Kosko B 1996 What is the best shape for a fuzzy set in function approximation? In *Proc. 5th IEEE Int. Conf. on Fuzzy Systems (FUZZ-96)*, pp. 542–560.
- Mitaim S and Kosko B 1997 Adaptive joint fuzzy sets for function approximation. In *Proc. Int. Conf. on Neural Networks (ICNN-97)*, pp. 537–542.

- Mhaskar HN and Micchelli C 1992 Approximation by superposition of sigmoidal and radial basis functions. *Adv. Appl. Math.* **13**, 350–373.
- Minsky M and Papert SA 1969 *Perceptrons: An Introduction to Computational Geometry*. MIT Press.
- Mozer MC 1993 Neural net architectures for temporal sequence processing. In *Time Series Prediction: Forecasting the Future and Understanding the Past* (ed. Weigend AS and Gershenfeld NA). Addison-Wesley.
- Murtagh P and Tsoi AC 1992 Implementation issues of sigmoid function and its derivative for VLSI digital neural networks. *IEE Proc.* E **139**(3).
- Nakagawa M 1996 An autonomously controlled chaos neural network. In *Proc. IEEE Int. Conf. on Neural Networks*, vol. 2, pp. 862–867.
- Narendra KS 1996 Neural networks for control: theory and practice. *Proc. IEEE* **84**, 1385–1406.
- Narendra KS and Parthasarathy K 1990 Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks* **1**, 4–27.
- Nerrand O, Roussel-Ragot P, Personnaz L and Dreyfus G 1991 Neural network training schemes for non-linear adaptive filtering and modelling. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN-91)*, vol. I, pp. 61–66.
- Nerrand O, Roussel-Ragot P, Personnaz L and Dreyfus G 1993 Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms. *Neural Computation* **5**, 165–199.
- Nerrand O, Roussel-Ragot P, Urbani D, Personnaz L and Dreyfus G 1994 Training recurrent neural networks: Why and how? An illustration in dynamical process modelling. *IEEE Trans. Neural Networks* **5**, 178–184.
- Niranjan M and Kadirkamanathan V 1991 A nonlinear model for time series prediction and signal interpolation. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-91)*, pp. 1713–1716.
- Nitzberg R 1985 Application of the normalized LMS algorithm to MSLC. *IEEE Trans. Aerospace Electron. Syst.* **AES-21**(1):79–91.
- Oppenheim AV, Buck JR and Schafer RW 1999 *Discrete-Time Signal Processing*. Prentice-Hall.
- Papoulis A 1984 *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill.
- Pearson RK 1995 Nonlinear input/output modelling. *J. Process Control* **5**(4), 197–211.
- Peng HC, Sha LF, Gan Q and Wei Y 1998 Combining adaptive sigmoid packet and trace neural networks for fast invariance learning. *Electron. Lett.* **34**, 898–899.
- Personnaz L and Dreyfus G 1998 Comment on ‘Discrete-time recurrent neural network architectures: a unifying review’. *Neurocomputing* **20**, 325–331.
- Piazza F, Uncini A and Zenobi M 1992 Artificial neural networks with adaptive polynomial activation function. In *Proc. Int. Joint Conf. on Neural Networks*, vol. II, pp. 343–349.
- Piazza F, Uncini A and Zenobi M 1993 Neural networks with digital LUT activation functions. In *Proc. 1993 Int. Joint Conf. on Neural Networks*, pp. 1401–1404.
- Pineda FJ 1987 Generalization of backpropagation to recurrent neural networks. *Phys. Rev. Lett.* **59**, 2229–2232.
- Pineda FJ 1989 Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation* **1**, 161–172.
- Poddar P and Unnikrishnan KP 1991 Non-linear prediction of speech signals using memory neuron networks. In *Proc. IEEE Workshop NNSP I*, pp. 395–404.
- Poggio T and Girosi F 1990 Networks for approximation and learning. *Proc. IEEE* **78**, 1481–1497.
- Premaratne K and Mansour M 1995 Robust stability of time-variant discrete-time systems with bounded parameter perturbations. *IEEE Trans. Circuits and Systems. I. Fundamental Theory and Applications* **42**, 40–45.

- Priestley MB 1991 *Non-Linear and Non-Stationary Time Series Analysis*. Academic Press, London.
- Principe JC, deVries B and deOliveira PG 1993 The Gamma filter—a new class of adaptive IIR filters with restricted feedback. *IEEE Trans. Signal Processing* **41**, 649–656.
- Principe JC, Euliano NR and Lefebvre WC 2000 *Neural and Adaptive Systems*. Wiley.
- Puskoris GA and Feldkamp LA 1994 Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Trans. Neural Networks* **5**, 279–297.
- Qin SZ, Su HT and Mc-Avoy TJ 1992 Comparison of four neural net learning methods for dynamic system identification. *IEEE Trans. Neural Networks* **3**, 122–130.
- Rao DH and Gupta MM 1993 Dynamic neural units and function approximation. In *Proc. IEEE Int. Conf. on Neural Networks*, vol. 2, pp. 743–748.
- Reed R 1993 Pruning algorithms—a survey. *IEEE Trans. Neural Networks* **4**, 740–747.
- Regalia PA 1994 *Adaptive IIR Filtering in Signal Processing and Control*. Marcel Dekker.
- Ridella S, Rovetta S and Zunino R 1997 Circular backpropagation networks for classification. *IEEE Trans. Neural Networks* **8**, 84–97.
- Robinson AJ and Fallside F 1987 The utility driven dynamic error propagation network. Tech. Rep. CUED/F-INFENG/TR.1, Cambridge University Engineering Department.
- Rose K 1998 Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proc. IEEE* **86**, 2210–2239.
- Rosenblatt F 1958 The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**, 386–408.
- Rosenblatt F 1962 *Principles of Neuro-Dynamics*. Washington, DC, Spartan.
- Roy S and Shynk JJ 1989 Analysis of the data-reusing LMS algorithm. In *Proc. 32nd Midwest Symp. on Circuits and Systems*, vol. 2, pp. 1127–1130.
- Ruck DW, Rogers SK, Kabrisky M, Maybeck PS and Oxley ME 1992 Comparative analysis of backpropagation and the extended Kalman filter for training multilayer perceptrons. *IEEE Trans. Pattern Analysis and Machine Intelligence* **14**, 686–691.
- Rumelhart DE, Hinton GE and Williams R 1986 Learning internal representation by error propagation. *Nature* **323**, 533–536.
- Sastry S and Bodson M 1989 *Adaptive Control: Stability, Convergence, and Robustness*. Prentice-Hall International.
- Schetzen M 1981 Nonlinear system modelling based on the Wiener theory. *Proc. IEEE* **69**, 1557–1574.
- Schnauffer BA and Jenkins WK 1993 New data-reusing LMS algorithms for improved convergence. In *Conf. Record 27th Asilomar Conf. on Signals and Systems*, vol. 2, pp. 1584–1588.
- Sethares WA 1992 Adaptive algorithms with nonlinear data and error functions. *IEEE Trans. Signal Processing* **40**, 2199–2206.
- Shadafan RS and Niranjana M 1993 A dynamic neural network architecture by sequential partitioning of the input space. In *Proc. IEEE Int. Conf. on Neural Networks*, pp. 226–231.
- Sheu M-H, Wang J-F, Chen J-S, Suen A-N, Jeang Y-L and Lee J-Y 1992 A data-reuse architecture for gray-scale morphologic operations. *IEEE Trans. Circuits and Systems. II. Analog and Digital Signal Processing* **39**, 753–756.
- Shynk J 1989 Adaptive IIR filtering. *IEEE Acoust. Speech and Signal Processing (ASSP) Mag.* **6**(2), 4–21.
- Shynk JJ and Roy S 1990 Convergence properties and stationary points of a perceptron learning algorithm. *Proc. IEEE* **78**, 1599–1604.
- Siegelmann H and Sontag ED 1995 On the computational power of neural networks. *J. Computat. Syst. Sci.* **45**, 132–150.
- Siegelmann HT, Horne BG and Giles CL 1997 Computational capabilities of recurrent NARX neural networks. *IEEE Trans. Systems Man. Cybernetics. Part B. Cybernetics* **27**(2), 208–215.

- Sjöberg J, Zhang Q, Ljung L, Benveniste A, Delyon B, Glorennec P-Y, Hjalmarsson H and Juditsky A 1995 Nonlinear black-box modelling in system identification: a unified overview. *Automatica* **31**, 1691–1724.
- Slock DTM 1993 On the convergence properties of the LMS and the normalized LMS algorithms. *IEEE Trans. Signal Processing* **41**, 2811–2825.
- Solo V and Kong X 1994 *Adaptive Signal Processing Algorithms: Stability and Performance*. Prentice-Hall.
- Song M and Manry MT 1993 Conventional modeling of the multilayer perceptron using polynomial basis functions. *IEEE Trans. Neural Networks* **4**, 164–166.
- Soria-Olivas E, Calpe-Maravilla J, Guerrero-Martinez JF, Martinez-Sober M and Espi-Lopez J 1998 An easy demonstration of the optimum value of the adaptation constant in the LMS algorithm. *IEEE Trans. Education* **41**(1), 81.
- Sprecher DA 1965 On the structure of continuous functions of several variables. *Trans. Am. Math. Soc.* **115**, 340–355.
- Sprecher DA 1993 A universal mapping for Kolmogorov's superposition theorem. *Neural Networks* **6**, 1089–1094.
- Stark H and Woods JW 1986 *Probability, Random Processes and Estimation Theory for Engineers*. Prentice-Hall.
- Strogatz SH 1994 *Nonlinear Dynamics and Chaos With Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books.
- Sum J, Leung C-S, Young GH and Kan W-K 1999 On the Kalman filtering method in neural network training and pruning. *IEEE Trans. Neural Networks* **10**, 161–166.
- Szu H and Hartley R 1987 Nonconvex optimization by fast simulated annealing. *Proc. IEEE* **75**, 1538–1540.
- Takens F 1981 On the numerical determination of the dimension of an attractor. In *Dynamical Systems and Turbulence* (ed. Rand D and Young LS). Warwick 1980 Lecture Notes in Mathematics, vol. 898. Springer.
- Tanaka K 1996 Stability analysis of neural networks via Lyapunov approach. In *Proc. Int. Conf. on Neural Networks*, vol. 6, pp. 3192–3197.
- Tarrab M and Feuer A 1988 Convergence and performance analysis of the normalized LMS algorithm with uncorrelated Gaussian data. *IEEE Trans. Information Theory* **34**, 680–691.
- Theiler J, Lindsay PS and Rubin DM 1993 Detecting nonlinearity in data with long coherence times with internal delays. In *Time Series Prediction: Forecasting the Future and Understanding the Past* (ed. Weigend AS and Gershenfeld NA). Addison-Wesley.
- Thimm G and Fiesler E 1997a High order and multilayer perceptron initialisation. *IEEE Trans. Neural Networks* **8**, 349–359.
- Thimm G and Fiesler E 1997b Optimal setting of the weights, learning rate, and gain. Tech. Rep. IDIAP-RR-97-04, Institut Dalle Molle D'Intelligence Artificielle Perceptive, Martigny Valais, Switzerland.
- Thimm G, Moerland P and Fiesler E The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation* **8**, 451–460, 1996.
- Tikhonov AN, Leonov AS and Yagola AG 1998 *Nonlinear ill-posed problems*. Applied Mathematics and Mathematical Computation. Chapman & Hall, London.
- Townshend B 1991 Nonlinear prediction of speech. In *Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-91)*, pp. 425–428.
- Treichler JR, Johnson Jr CR and Larimore MG 1987 *Theory and Design of Adaptive Filters*. Wiley.
- Tsoi CA and Back A 1997 Discrete time neural network architectures: a unifying review. *Neurocomputing* **15**(3), 183–223.
- Vaidyanathan PP and Mitra SK 1987 A unified structural interpretation of some well-known stability-test procedures for linear systems. *Proc. IEEE* **75**, 478–497.

- Vecci L, Campolucci P, Piazza F and Uncini A 1997 Approximation capabilities of adaptive spline neural networks. In *Proc. Int. Conf. on Neural Networks*, vol. 1, pp. 260–265.
- Vitushkin AG 1954 On Hilbert's thirteenth problem. *Dokl. Akad. Nauk SSSR* **95**, 701–704.
- Waibel A, Hanazawa T, Hinton G, Shikano K and Lang K 1989 Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoustics, Speech and Signal Processing* **37**, 328–339.
- Wallace WA 1992 *Galileo's Logical Treatises (A Translation, with Notes and Commentary, of His Appropriated Latin Questions on Aristotle's Posteriori Analytics)*. Kluwer.
- Wan EA 1993 Time series prediction by using a connectionist network with internal delay lines. In *Time Series Prediction: Forecasting the Future and Understanding the Past* (ed. Weigend AS and Gershenfeld NA). Addison-Wesley.
- Wang K and Michel AN 1994 On Lyapunov stability of a family of nonlinear time-varying systems. In *Proc. 33rd Conf. on Decision and Control*, pp. 2131–2136.
- Weigend AS and Gershenfeld NA (eds) 1994 *Time Series Prediction: Forecasting the Future and Understanding the Past*. Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley.
- Werbos P 1974 Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science. PhD thesis, Harvard University, Cambridge, MA.
- Werbos P 1990 Backpropagation through time: what it does and how to do it. *Proc. IEEE* **78**, 1550–1560.
- Widrow B and Hoff ME 1960 Adaptive switching circuits. In *Proc. WESCON Convention*, vol. IV, pp. 96–104.
- Widrow B and Lehr MA 1990 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proc. IEEE* **78**, 1415–1442.
- Widrow B and Stearns SD 1985 *Adaptive Signal Processing*. Prentice-Hall.
- Wiener N 1949 *The Extrapolation, Interpolation and Smoothing of Stationary Time Series with Engineering Applications*. Wiley.
- Williams RJ 1992 Training recurrent networks using the extended Kalman filter. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN'92)*, vol. IV, pp. 241–246.
- Williams R and Zipser D 1989a A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* **1**, 270–280.
- Williams RJ and Zipser D 1989b Experimental analysis of the real-time recurrent learning algorithm. *Connection Sci.* **1**(1), 87–111.
- Williams RJ and Zipser D 1995 Gradient-based algorithms for recurrent networks and their computational complexity. In *Backpropagation: Theory, Architectures, and Applications* (ed. Chauvin Y and Rumelhart DE). Lawrence Erlbaum Associates.
- Williamson RC and Helmke U 1995 Existence and uniqueness results for neural network approximations. *IEEE Trans. Neural Networks* **6**, 2–13.
- Wilkinson JH 1965 *The Algebraic Eigenvalue Problem*. Oxford University Press.
- Wu L and Niranjan M 1994 On the design of nonlinear speech predictors with recurrent nets. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-94)*, pp. 529–532.
- Wu S-I 1995 Mirroring our thought processes. *IEEE Potentials* **14**, 36–41.
- Yang J, Ahmadi M, Jullien GA and Miller WC 1998 Model validation and determination for neural network activation function modelling. In *Proc. 1998 Midwest Symp. on Circuits and Systems*, pp. 548–551.
- Yee P and Haykin S 1999 A dynamic regularized radial basis function network for nonlinear, nonstationary time series prediction. *IEEE Trans. Signal Processing* **47**, 2503–2521.
- Yule GU 1927 On a method of investigating periodicities in disturbed series, with special reference to Wölfer's sunspot numbers. *Phil. Trans. R. Soc. Lond. A* **226**, 267–298.
- Zakeri S 1996 On critical points of proper holomorphic maps on the unit disc. *Bull. Lond. Math. Soc.* **30**, 62–66.

- Zeidler E 1986 *Nonlinear Functional Analysis and its Applications*, vol. 1: *Fixed-Point Theorems*. Springer.
- Zhang J, Walter GG, Miao Y and Lee WNW 1995 Wavelet neural networks for function learning. *IEEE Trans. Signal Processing* **43**, 1485–1497.
- Zhang S and Constantinides AG 1992 Lagrange programming neural networks. *IEEE Trans. Circuits and Systems. II. Analog and Digital Signal Processing* **37**, 441–452.
- Zhang Z and Sarhadi M 1993 A modified neuron activation function which enables single layer perceptrons to solve some linearly inseparable problems. In *Proc. 1993 Int. Conf. on Neural Networks*, pp. 2723–2726.
- Zhang Z and Sarhadi M 1993 Performance aspects of a novel neuron activation function in multilayer feedforward networks. In *Proc. 1993 Int. Joint Conf. on Neural Networks*, pp. 2727–2730.
- Zurada JM and Shen W 1990 Sufficient condition for convergence of a relaxation algorithm in actual single-layer neural networks. *IEEE Trans. Neural Networks* **1**, 300–303.

Index

- Activation functions
 - continual adaptation 202
 - definition 239
 - desirable properties 51
 - examples 53
 - properties to map to the complex plane 61
 - steepness 200
 - temperature 200
 - why nonlinear 50
- Activation potential 36, 54
- Adaline 2
- Adaptability 9
- Adaptation gain
 - behaviour of steepest descent 16
- Adaptive algorithms
 - convergence criteria 163
 - momentum 211
 - performance criteria 161
- Adaptive learning 21
- Adaptive resonance theory 2
- Adaptive systems
 - configurations 10
 - generic structure 9
- Analytic continuation 61
- A posteriori*
 - algorithms 113, 138
 - computational complexity 138
 - error 135
 - techniques 241
- Asymptotic convergence rate 121
- Asymptotic stability 116
- Attractors 115
- Autonomous systems 263
- Autoregressive (AR) models
 - coefficients 37
- Autoregressive integrated moving average (ARIMA) model 171
- Autoregressive moving average (ARMA) models
 - filter structure 37
- Averaging methods 163
- Backpropagation 18
 - normalised algorithm 153
 - through time 209
 - through time dynamical equivalence 210
- Batch learning 20
- Bias/variance dilemma 112
- Bilinear model 93
- Black box modelling 73
- Blind equalizer 12
- Block-based estimators 15
- Bounded Input Bounded Output (BIBO) stability 38, 118
- Brower's Fixed Point Theorem 247
- Canonical state-space representation 44
- Cauchy–Riemann equations 228
- Channel equalisation 11
- Clamping functions 240
- Classes of learning algorithm 18
- Cognitive science 1
- Complex numbers
 - elementary transformations 227
- Connectionist models 1
 - dates 2
- Constructive learning 21
- Contraction mapping theorem 245
- Contractivity of relaxation 252
- Curse of dimensionality 22

- David Hilbert 47
- Data-reusing 114, 135, 142
 - stabilising features 137, 145
- Delay space embedding 41
- De-seasonalising data 265
- Deterministic learning 21
- Deterministic versus stochastic (DVS)
 - plots 172, 175
- Directed algorithms 111
- Domain of attraction 116
- Dynamic multilayer perceptron (DMLP) 79
- Efficiency index 135
- Electrocardiogram (ECG) 181
- Embedding dimension 74, 76, 174, 178
- Embedded memory 111
- Equation error 104
 - adaptation 107
- Equilibrium point 264
- Error criterion 101
- Error function 20
- Exogeneous inputs 40
- Exponentially stable 264
- Extended Kalman filter 109
- Extended recursive least squares
 - algorithm 217, 236
- Feedforward network
 - definition 239
- Fixed point
 - iteration 143
 - theory 117, 245
- Forgetting behaviour 110
- Forgetting mechanism 101
- Frobenius matrix 251
- Function definitions
 - conformal 227
 - entire 227
 - meromorphic 227
- Gamma memory 42
- Gaussian variables
 - fourth order standard factorisation 167
- Gear shifting 21
- Global asymptotic stability (GAS)
 - 116, 118, 251, 264
- Gradient-based learning 12
- Grey box modelling 73
- Hammerstein model 77
- Heart rate variability 181
- Heaviside function 55
- Hessian 15, 52
- Holomorphic function 227
- Hyperbolic attractor 110
- Incremental learning 21
- Independence assumptions 163
- Independent Identically Distributed (IID) 38
- Induced local field 36
- Infinite Impulse Response (IIR)
 - equation error adaptive filter 107
- Input transformations 23
- Kalman Filter (KF) algorithm 14
- Kolmogorov function 223
- Kolmogorov's theorem 6, 93, 223
 - universal approximation theorem 47, 48
- Kolmogorov–Sprecher Theorem 224
- Learning rate 13
 - continual adaptation 202
 - selection 202
- Learning rate adaptation 200
- Least Mean Square (LMS) algorithm
 - 14, 18
 - data-reusing form 136
- Linear filters 37
- Linear prediction
 - foundations 31
- Linear regression 14
- Liouville Theorem 61, 228
- Lipschitz function 224, 246, 263
- Logistic function 36, 53
 - a contraction 118
 - approximation 58
 - fixed points of biased form 127
- Lorenz equations 174, 195
- Lyapunov stability 116, 143, 162
 - indirect method 162

- Mandelbrot and Julia sets 61
- Markov model
 - first order 164
- Massive parallelism 6
- Misalignment 168, 169
- Möbius transformation 47, 228
 - fixed points 67
- Model reference adaptive system (MRAS) 106
- Modular group 229
 - transfer function between neurons 66
- Modular neural networks
 - dynamic equivalence 215
 - static equivalence 214
- NARMA with eXogeneous inputs (NARMAX) model
 - compact representation 71
 - validity 95
- Nearest neighbours 175
- Nesting 130
- Neural dynamics 115
- Neural network
 - bias term 50
 - free parameters 199
 - general architectures for prediction and system identification 99
 - growing and pruning 21
 - hybrid 84
 - in complex plane 60
 - invertibility 67
 - modularity 26, 199, 214
 - multilayer feedforward 41
 - nesting 27
 - node structure 2
 - ontogenic 21
 - properties 1
 - radial basis function 60
 - redundancy 113
 - specifications 2
 - spline 56
 - time-delay 42
 - topology 240
 - universal approximators 49, 54
 - wavelet 57
 - with locally distributed dynamics (LDNN) 79
- Neuron
 - biological perspective 32
 - definition 3
 - structure 32, 36
- Noise cancellation 10
- Nonlinear Autoregressive (NAR) model 40
- Nonlinear Autoregressive Moving Average (NARMA) model 39
 - recurrent perceptron 97
- Nonlinear Finite Impulse Response (FIR) filter
 - learning algorithm 18
 - normalised gradient descent, optimal step size 153
 - weight update 201
- Nonlinear gradient descent 151
- Nonlinear parallel model 103
- Nonlinearity detection 171, 173
- Nonparametric modelling 72
- Non-recursive algorithm 25
- Normalised LMS algorithm
 - learning rate 150
- \mathcal{O} notation 221
- Objective function 20
- Ontogenic functions 241
- Orthogonal condition 34
- Output error 104
 - adaptive infinite impulse response (IIR) filter 105
 - learning algorithm 108
- Parametric modelling 72
- Pattern learning 26
- Perceptron 2
- Phase space 174
- Piecewise-linear model 36
- Pipelining 131
- Polynomial equations 48
- Polynomial time 221
- Prediction
 - basic building blocks 35
 - conditional mean 39, 88
 - configuration 11

- difficulties 5
- history 4
- principle 33
- reasons for using neural networks 5
- Preservation of
 - contractivity/expansivity 218
- Principal component analysis 23
- Proximity functions 54
- Pseudolinear regression algorithm 105
- Quasi-Newton learning algorithm 15
- Rate of convergence 121
- Real time recurrent learning (RTRL)
 - 92, 108, 231
 - a posteriori* form 141
 - normalised form 159
 - teacher forcing 234
 - weight update for static and dynamic equivalence 209
- Recurrent backpropagation 109, 209
 - static and dynamic equivalence 211
- Recurrent neural filter
 - a posteriori* form 140
 - fully connected 98
 - stability bound for adaptive algorithm 166
- Recurrent neural networks (RNNs)
 - activation feedback 81
 - dynamic behaviour 69
 - dynamic equivalence 205, 207
 - Elman 82
 - fully connected, relaxation 133
 - fully connected, structure 231
 - Jordan 83
 - local or global feedback 43
 - locally recurrent–globally feedforward 82
 - nesting 130
 - output feedback 81
 - pipelined (PRNN) 85, 132, 204, 234
 - rate of convergence of relaxation 127
 - relaxation 129
 - RTRL optimal learning rate 159
 - static equivalence 205, 206
 - universal approximators 49
 - Williams–Zipser 83
- Recurrent perceptron
 - GAS relaxation 125
- Recursive algorithm 25
- Recursive Least-Squares (RLS)
 - algorithm 14
- Referent network 205
- Riccati equation 15
- Robust stability 116
- Sandwich structure 86
- Santa Fe Institute 6
- Saturated-modulus function 57
- Seasonal ARIMA model 266
- Seasonal behaviour 172
- Semiparametric modelling 72
- Sensitivities 108
- Sequential estimators 15
- Set
 - closure 224
 - compact 225
 - dense subset 224
- Sigmoid packet 56
- Sign-preserving 162
- Spin glass 2
- Spline, cubic 225
- Staircase function 55
- Standardisation 23
- Stochastic learning 21
- Stochastic matrix 253
- Stone–Weierstrass theorem 62
- Supervised learning 25
 - definition 239
- Surrogate dataset 173
- System identification 10
- System linearity 263
- Takens’ theorem 44, 71, 96
- tanh activation function
 - contraction mapping 124
- Teacher forced adaptation 108
- Threshold nonlinearity 36
- Training set construction 24
- Turing machine 22
- Unidirected algorithms 111
- Uniform approximation 51

- Uniform asymptotic stability 264
- Uniform stability 264
- Unsupervised learning 25
 - definition 239
- Uryson model 77
- Vanishing gradient 109, 166
- Vector and matrix differentiation
 - rules 221
- Volterra series
 - expansion 71
- Weierstrass Theorem 6, 92, 224
- White box modelling 73
- Wiener filter 17
- Wiener model 77
 - represented by NARMA model 80
- Wiener–Hammerstein model 78
- Wold decomposition 39
- Yule–Walker equations 34
- Zero-memory nonlinearities 31
 - examples 35