INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Architectures and Algorithms for Stable and Constructive Learning in Discrete Time Recurrent Neural Networks

by

Shyamala C. Sivakumar

A Thesis Submitted to the

Faculty of Engineering

in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

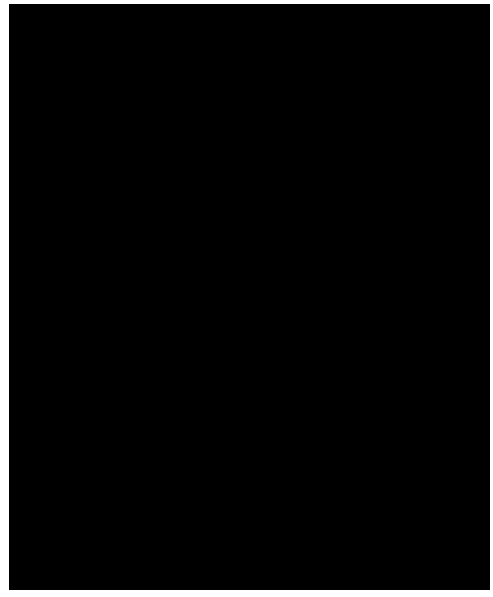
APPROVED:

W. Robertson

W.P. Phillips

C.R. Baird

M. Stevenson



TECHNICAL UNIVERSITY OF NOVA SCOTIA

Halifax, Nova Scotia

1997

TECHNICAL UNIVERSITY OF NOVA SCOTIA LIBRARY

"AUTHORITY TO DISTRIBUTE MANUSCRIPT THESIS"

Title:

Architectures and Algorithms for Stable and Constructive Learning in Discrete Time Recurrent Neural Networks

The above library may make available or authorize another library to make available individual photo/microfilm copies of this thesis without restrictions.

Full Name of Author: Shyamala Chandramouli Sivakumar

Signature of Author [redacted]

Date 16 APRIL - 1997

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

Notation	Description
BDRNN	Block Diagonal Recurrent Neural Network
BPTT	BackPropagation Through Time
CFNN	Constraining Feedforward Neural Network
DRM	Digit Recognition Module
DRNN	Diagonally Recurrent Neural Network
DTRNN	Discrete Time Recurrent Neural Network
FF-BDRNN	Feedforward Block Diagonal Recurrent Neural Network
LRGF	Locally Recurrent Globally Feedforward
NRMSE	Normalized Root Mean Square Error
RNN	Recurrent Neural Network
RTRL	Real Time Recurrent Learning
SFNN	Stabilizing Feedforward Neural Network
a_n	real coefficients of the characteristic polynomial of a linear time-invariant system
b_n	real coefficients of a second order polynomial
B	N x M input weight matrix
B^f	input weight matrix associated with a fully recurrent DTRNN
b_{ij}	weight from the j-th state unit to the i-th external input unit
b^f_{ij}	weight from the j-th state unit to the i-th external input unit associated with a fully recurrent DTRNN
c_n	real coefficients of a first order polynomial
C	N_o x N output weight matrix
C_i	sum of the magnitude of all elements other than the diagonal element in the i-th column of W

c_j^s	constant shadow-output weight matrix
$c_{i,j}$	weight from the j-th state unit to the i-th output unit
$d_{i,j}^{\ l}$	feedforward weight that connects the j-th node of layer l to the i-th node of layer l+1
$e_h(k)$	output error of the h-th output unit at instant k
$\underline{e}^s$	output error vector of the SFNN
$e^s(k)$	scalar shadow error at instant k
E_t	cost function for the t-th training sequence
$E_s(X,V)$	an energy function used by Simard et al
f	symmetric sigmoidal squashing function
f^I	inverse of the symmetric sigmoidal squashing function f
$f'(x)$	derivative of $f(x) = 1 - [f(x)]^2$ for symmetric sigmoidal squashing function
g	sigmoidal squashing function in the output layer
J_t	squared output error function for the t-th training sequence
$J_p(t)$	total squared error over all the N_o output units for the p-th training pattern
k	index on the time scale of input sequences
k_s	constant associated with the ideal stability function
K_t	length of the t-th training sequence
K_p	total number of steps in the p-th training pattern
L	total number of layers
m	index on the time scale of weight updates
M	total number of input units
N	total number of states in the DTRNN
N_l	number of nodes in layer l
N_L	number of nodes in the output layer
N_o	total number of output units
N_s	total number of state vectors to be stored
$p_{i,j}^{\ h}(k)$	partial derivative of $x_h(k)$ with respect to $w_{i,j}$

P_t	penalty function for the t-th training sequence
$q_{ij}^{k}(k)$	partial derivative of $y_k(k)$ with respect to w_{ij}
$r_j(k)$	desired response of the j-th node of the output layer at time instant k
r^s	required value of the stability function y^s
$r^{s1}{}_i$	required value of the first local stability function $y^{s1}{}_i$, of the freeform BDRNN
$r^{s2}{}_i$	required value of the second local stability function $y^{s2}{}_i$, of the freeform BDRNN
$\underline{r}^s$	desired output vector of the SFNN
$r^{\epsilon}{}_i$	required value of the invertibility function $y^{\epsilon}{}_i$
R_i	sum of the magnitude of all elements other than the diagonal element in the i-th row of W
s_i	instants at which intermediate state vectors are stored
t	t-th training sequence
T	total number of training sequences
T	transformation matrix
$u_i(k)$	i-th component of external input vector and is the same as $u_i^1(k)$
$\underline{u}(k)$	external input vector
$u_j^l(k)$	output activation of the j-th unit in layer l at time instant k
u_i^l	i-th component of the input vector
$\underline{u}^s$	input vector to the SFNN
V	unitary vector
W	N x N state feedback weight matrix of a BDRNN
W^T	transpose of the state feedback weight matrix W
W^{ϵ}	inverse of the state feedback weight matrix W
W^+	equals W for local stability, and equals W^TW for global stability
$\lvert W \rvert$	a suitably defined norm of W
$\lVert W \rVert_1$	1-norm of W

$\|W\|_2$	2-norm of W
$\|W\|_\infty$	∞-norm of W
W^f	N x N state feedback weight matrix of a fully recurrent DTRNN
W_o	scaled orthogonal 2 x 2 weight matrix
$W'_{n/2}$	$n/2$-th 2 x 2 scaled orthogonal submatrix obtained by extracting rows n-1,n and columns n-1,n of the block-diagonal W
$W''_{n/2}$	$n/2$-th 2 x 2 freeform submatrix obtained by extracting rows n-1,n and columns n-1,n of the block-diagonal freeform W
$w_{i,j}$	weight from the i-th to the j-th state unit
$w^{\epsilon}_{i,j}$	weight from the i-th to the j-th unit of the inverse matrix of W
$w^T_{i,j}$	element in the i-th row, j-th column of the transpose of the feedback weight matrix
w^{δ}_{min}	constant associated with the invertibility of $W'_{n/2}$ or $W''_{n/2}$
$\underline{x}$	equilibrium state
$x_i(0)$	initial states of the system
$x_i(k)$	i-th component of the state vector
$x^n_i(k)$	i-th component of the state vector of the neural network
$x^p_i(k)$	i-th component of the state vector of the plant
$x^r_i(k)$	i-th component of the recomputed state vector
$\underline{x}(k)$	state vector $\{x_i(k)\}$
$\underline{x}'(k)$	state vector associated with a fully recurrent DTRNN after transformation
$\underline{x}''(k)$	state vector associated with the BDRNN after transformation
$y_j(k)$	actual response of the j-th node of the output layer at instant k
$y^n_j(k)$	j-th output of the neural network at instant k
$y^p_j(k)$	j-th output of the plant at instant k
$\underline{y}(k)$	output vector
$y^{fs}(k)$	scalar shadow output computed in the forward pass at time instant k
$y^{rs}(k)$	scalar shadow output recomputed in the backward pass for time instant k

y^{sg}_i	actual value of the global stability function associated with the i-th submatrix of the freeform BDRNN
y^{s1}_i	actual value of the first local stability function associated with the i-th submatrix of the freeform BDRNN
y^{s2}_i	actual value of the second local stability function associated with the i-th submatrix of the freeform BDRNN
y^{s}_i	actual value of the stability function associated with the i-th submatrix of W
y^{e}_i	actual value of the invertibility function associated with the i-th submatrix of the scaled orthogonal BDRNN
y^{e2}_i	actual value of the invertibility function associated with the i-th submatrix of the freeform BDRNN
z^s	output vector of the SFNN
α	gain of the symmetric sigmoidal function
$\alpha/2$	maximum bound on $\|f_\alpha'(.)\|$ i.e, the maximum bound on the magnitude of the slope of the symmetric sigmoidal function
B	decay parameter associated with each self recurrent neuron in the DRNN proposed by Gori et al
δ	*kronecker* delta function
Δb_{ij}	accumulation of the instantaneous error gradient w.r.t b_{ij} over the K_p steps of the training pattern
Δc_{ij}	accumulation of the instantaneous error gradient w.r.t c_{ij} over the K_p steps of the training pattern
Δw_{ij}	accumulation of the instantaneous error gradient w.r.t w_{ij} over the K_p steps of the training pattern
$\Delta W(t)$	Jacobian matrix of W
$\Delta \chi_i$	accumulation of the instantaneous error gradient w.r.t χ_i over the K_p steps of the training pattern

E	threshold value of scalar shadow error $e^s_j(k)$
$\varepsilon_i(k)$	$\partial J_p(k)/\partial x_i(k)$; partial derivative of the $J_p(k)$ w.r.t $x_i(k)$
$\lambda_i(A)$	i-th eigenvalue of the square matrix A
$\lambda_{max}(A)$	maximum eigenvalue of the square matrix A
Λ	a block diagonal matrix
K	contracting mapping used by Simard et al in the direction of the unitary vector V
μ_l	learning rate parameter
μ_{max}	maximum limiting value that the learning rate can take
$\zeta_{ij}^{h}(k)$	partial derivative of $x_h(k)$ with respect to b_{ij}
χ_i	parameter associated with each self recurrent neuron in a DRNN

ACKNOWLEDGMENTS

I am grateful to Dr. W.Robertson and Dr. W.J.Phillips for their supervision during the course of this work. I am also grateful to Dr. C.R.Baird for his encouragement.

I am thankful to Dr. M.Stevenson, my external examiner, for her insightful comments that helped to improve this thesis.

The financial support provided by the Natural Sciences and Engineering Research Council of Canada and by the Technical University of Nova Scotia is appreciated.

Thanks are due to my husband, Kumar, for the enlightening and lively discussions that I had with him on various issues of this thesis. Thanks are also due to my young son, Kiran, for being patient with his mother during the course of this work. I am deeply indebted to my parents, Mr. Chandramouli and Mrs. Vijaya, for their unwavering faith in me.

ABSTRACT

This thesis deals with a discrete time recurrent neural network (DTRNN) with a block-diagonal feedback weight matrix, called the block-diagonal recurrent neural network (BDRNN), that allows a simplified approach to on-line training and addresses stability issues. It is known that backpropagation-through-time (BPTT) is the algorithm of choice for training DTRNN due to the exact and local nature of gradient computation. However, the BPTT algorithm stores the network state variables at each time instant and hence requires large storage for long training sequences. Here, the block-diagonal structure of the BDRNN is exploited to modify the BPTT algorithm to reduce the storage requirements while still maintaining exactness and locality of gradient computation. To achieve this, a numerically stable method for recomputing the state variables in the backward pass of the BPTT algorithm is proposed. It is also known that the local or global stability of DTRNN during training is guaranteed if a suitably defined norm of the updated weight matrix is less than a bound determined by the slope of the sigmoidal limiter. The determination of this norm at each weight update requires eigenvalue computations and is computationally expensive. In linear systems, this is overcome by using special sparser structures which facilitate direct monitoring of stability during weight updates by examining appropriate matrix elements. In this thesis, this approach is extended by exploiting the sparse structure of the BDRNN to monitor and maintain stability. This is addressed, first, by developing a suitable stability function that provides a measure of the system eigenvalues with reference to the unit circle; next, a penalty term based on this stability function is incorporated as part of the cost function being minimized during training. It is shown that the stability function can be suitably tailored and formulated as a constrained feedforward neural network. This allows the stabilization to be addressed in a feedforward BDRNN framework which can be trained using conventional gradient descent techniques. Finally, a modular construction method is presented that is suitable for modelling dynamic trajectories which can be decomposed

into several subdynamics. The performance of the FF-BDRNN architecture, the new learning algorithm, the stabilization technique and the construction method are demonstrated using several simulation examples.

1 INTRODUCTION

1.1 CURRENT TECHNIQUES AND PROBLEM DEFINITION

Artificial neural networks can be classified broadly into two categories: static networks and dynamic networks. Static networks are memoryless and generally consist of several layers of neurons/units that are cascaded and are used to model systems whose outputs are a complex nonlinear function of current external inputs to the system. Introducing time delays or feedback (recurrence) in the neurons produces dynamic networks that have memory. Dynamic networks with feedforward dynamics with discrete time delays [Waibel et al] have been successfully used for recognition of voiced stop consonants in speech, while, networks with continuous adaptive time delays have been applied to the prediction of chaotic time series [Day and Davenport]. Dynamic networks with network output feedback [Narendra and Parthasarathy 1990] have been used for nonlinear identification and control problems. Dynamic networks with state feedback, also known as recurrent neural networks (RNN), are gaining popularity due to their feedback structure which can be harnessed to model dynamic characteristics of time varying signals [Williams and Zipser 1989, Almeida, Pineda, Pearlmutter 1989, Narendra and Parthasarathy 1990 and 1991, Nerrand et al, Sato, Werbos, Tsoi and Back, Gori et al, Uchiyama et al]. The system equations of such networks are generally described by differential or difference equations for continuous and discrete recurrent networks respectively. Recurrent networks with state feedback can be further subdivided into fully recurrent and locally recurrent globally-feedforward (LRGF) networks (see [Tsoi and Back] for a survey). The fully recurrent architecture [Williams and Zipser 1989] is generally one-layered in which the output of every neuron is fed back with varying gains to the inputs of all neurons. The LRGF architecture is a hybrid architecture that incorporates some feedback elements together with a multi-layer feedforward network. DTRNNs having the same architecture can be trained with two different distinct learning algorithms [Hunt et al], viz., fixed point learning and trajectory learning to model different dynamic behaviour. Therefore, a

network is defined by its architecture together with its learning rule and the overall input-output behaviour is the result of the interaction between the two. Fixed-point learning is employed in DTRNNs to model finite state machines, while, trajectory learning is employed in identification and control, to model or predict desired temporal trajectories. Simulation studies on certain tasks involving trajectory learning have shown that locally recurrent architectures can perform better and converge faster than fully recurrent networks [Tsoi and Back]. Also networks with sparse feedback connections such as locally recurrent architectures can be advantageous in terms of stability of learning and computational and storage requirements.

1.2 THESIS OBJECTIVES AND MAJOR CONTRIBUTIONS

In discrete time recurrent neural networks (DTRNN) the learning algorithm, stability of the network during learning and the network architecture play dominant roles in the generalization capability and the optimal training of the network.

This thesis considers a sparse but structured architecture in which the feedback connections are restricted to between pairs of state variables. This network has two layers, the first of which is a DTRNN feedback layer made up of a block-diagonal structure and the second an interconnecting output layer that combines the state variables of the feedback layer to generate the network output. This architecture is referred to as the block-diagonal recurrent neural network (BDRNN). The BDRNN structure is then extended to include a multi-layer feedforward network in order to model any direct complex nonlinear mapping between the external inputs and outputs of the system. The resultant structure, referred to as the feedforward block-diagonal recurrent neural network (FF-BDRNN), provides a framework to model both static and dynamic characteristics in a unified fashion. The importance of such an approach to practical applications has been widely recognized [Narendra and Parthasarathy 1990 and 1991, Day and Davenport, Tsoi

and Back]. The motivation for the choice of the BDRNN architecture is threefold: first, a block-diagonal feedback weight matrix is capable of modelling plants with complex eigenvalues; second, this structure is conducive to devising a training algorithm in which the computation of the gradient is "local" in both space and time; third, the use of the sparser but structured feedback matrix eases the problem of monitoring and maintaining network stability at each weight update [Kung]. From linear time-invariant systems theory it is known that an n-th order dynamics can be represented by a combination of several first or second order dynamics, each of which can be modelled by a 2 x 2 feedback weight matrix. Extending this concept to a nonlinear process, if its dynamics can be "decoupled" into several lower order dynamics, then it may be feasible to model it with a DTRNN having a block diagonal feedback weight matrix with blocks of size 2 with each submatrix modelling a low order dynamic of the process (note that a purely diagonal feedback matrix [Tsoi and Back, Gori et al, Ku and Lee] cannot model a second order system with complex conjugate roots). Note that this structure is especially useful in modelling the oscillating modes of a dynamic process.

Training of DTRNN is generally accomplished by using either real time recurrent learning (RTRL) [Williams and Zipser 1989] or backpropagation through time (BPTT) [Werbos] algorithms or variations thereof (see [Pearlmutter 1995] for a survey). Both the RTRL and BPTT algorithms compute the exact error gradient [Beaufays and Wan]. RTRL is computationally expensive due to the spatially nonlocal nature of its error gradient computation. In BPTT, the computation of the error gradient in the backward pass requires that the state, input and error vectors be stored at each time instant in the forward pass and hence, the storage requirement for BPTT increases with the length of the training pattern. The storage requirement for the state vectors can be eliminated by recalculating the states vectors in the backward pass. This, however, requires that the feedback weight matrix be inverted after each weight update. More importantly, this technique of recalculating the state vectors is susceptible to numerical instability problems [Pearlmutter 1995]. The block diagonal structure of the BDRNN lends itself to ensuring invertibility

as the feedback weight matrix is easily inverted by simple manipulation and scaling of its elements. Also the reduced interaction between the state variables of the BDRNN helps in reducing the numerical instability problem. This thesis proposes a modified online BPTT algorithm for the BDRNN that recalculates the state vectors and computes the exact error gradient. The question of ensuring numerical stability of the algorithm is addressed by the following: first, state vectors at evenly spaced intermediate time intervals are stored in the forward pass; second, these intermediate stored values are used as initial values to recompute the state vectors during the backward pass while simultaneously monitoring for signs of numerical instability; when numerical instability is detected, the state vector for that time instant is recomputed by performing a forward pass using the nearest stored intermediate state vector and the process of recomputing the state vectors is continued. The proposed algorithm results in reduced storage compared to conventional BPTT by trading off some of the later's computational advantage.

The feedback structure of the DTRNN necessarily raises the question of stability, both of the network and during its training. While, for a stable network, the stability of training is well understood and can be ensured by suitably choosing the learning rate [Almeida], there are several unresolved issues in monitoring and maintaining the stability of the network itself at each weight update. A major difficulty in monitoring the stability of a fully connected DTRNN relates to the computation of a suitably selected norm of its feedback matrix at each weight update during training. Conditions for the stability of DTRNN, given a specific architecture, have been studied by a number of authors [e.g., Gori et al, Li, Jin et al 1994 and 1996, Simard et al]. For example, Jin et al (1994 and 1996) used Ostrowski and Gerschgorin theorems and the similarity transformation approach to describe a region of absolute stability [Jin et al 1994] and globally asymptotical stability [Jin et al 1996] that depends on the parameters and connection weights of the network. However, using these conditions to ensure the continued stability of the network during the training process has not been explored. Simard et al have proposed a general, but complex, approach to the global stabilization of DTRNN using

the contraction mapping theorem. DTRNNs with constrained or special network structures that inherently ensure stability have been studied by several researchers [Sato, Gori et al, Bruck, Vidyasagar, Ku and Lee]. For example, Gori et al considers a multi-layer feedforward architecture which contains some self-recurrent neurons in which the stability is ensured by restricting the magnitude of the self-recurrent weights to less than unity. Ku et al have proposed a method to ensure stability and convergence of training of a diagonally recurrent neural network by an adaptive learning rate algorithm. In this thesis, the block diagonal structure of the BDRNN is exploited to find conditions that lead the network towards stability at each weight update during training. This problem is addressed in two steps: first, by devising a cost function that includes the desired stability margin as one of its components; next, by ingraining the stability margin component as part of the neural network architecture. For analytical tractability of the stability margin component of the cost function, some constraints are imposed on the values that the block diagonal weights can assume. It is shown that the stability margin components together with these constraints can be implemented as a multi-layer feedforward neural structure which augments the BDRNN structure.

The size and architecture of neural networks is generally determined by trial and error. However, this technique results in training a number of networks and then selecting the smallest network that meets the required performance criterion. This thesis also proposes a constructive method of designing BDRNN. The motivation for such an approach can be found in the idea that a nonlinear dynamic can be decomposed into a dominant nonlinear dynamic and a series of progressively less dominant subdynamics. Blocks of BDRNNs are employed, one each, to learn the dominant dynamic and each of the subdynamics. The advantages of such an approach are that the size of the BDRNN is determined automatically and methodically without any need for guesswork, and results in a faster learning time in comparison with the larger equivalent network that may be used to learn the same dynamic without construction. The construction algorithm starts with a basic block consisting of N_l BDRNN units. The basic BDRNN block can learn a

fundamental and N_f-1 sets of harmonics in a linear sense. Learning is performed with this basic architecture until there is no improvement in the output error. At any stage, whenever a plateau in the error performance is encountered, the weights of the existing architecture are frozen and incremental construction is accomplished by cascading a N_g-BDRNN unit block to the existing architecture. Blocks are grown iteratively until the desired normalized root mean square error (NRMSE) threshold is met or a maximum number of blocks have been added. The cascading constructive trajectory learning algorithm constructs a series of BDRNNs whose combined output is required to model the desired dynamic trajectory under consideration. The basic block is directly trained on the desired trajectory being learned, while, each additional cascading block is trained on the residual error between the most recent estimate and the desired trajectory. The new estimate is a combination of the outputs of the basic block and previous blocks together with the output of the current block being trained. A similar approach for functional approximation using feedforward neural networks was arrived at independently by [Draelos et al]. Such a constructive technique is ideal for learning nonlinear "decomposable" dynamics such as the chaotic Mackey Glass time series and also practical problems such as isolated-digit speech utterances. However, this technique may not be suitable for problems where the error dynamics at any stage has greater complexity than the input dynamics.

1.3 THESIS ORGANIZATION

Chapter 2 presents a review of the architectures, algorithms currently used for trajectory learning in DTRNNs, techniques used for monitoring and maintaining DTRNN stability, and methods adopted for determining DTRNN size. Trajectory learning in DTRNN is generally accomplished using either real time recurrent learning (RTRL) [Williams and Zipser 1989] or backpropagation through time (BPTT) [Werbos] algorithms or variations thereof ([Williams and Peng 1990, Schmidhuber, Sun et al 1992], see [Pearlmutter 1995]

for a survey). The techniques that are used for monitoring DTRNN stability are the local stability approach which involves linearization of the state-space equation, and the global stability approach which involves using the Liapunov function or the contraction mapping theorem or both. The methods used in determining DTRNN size can be broadly classified into destructive and constructive methods. Destructive methods start with a fairly large network and then remove weights or units that do not contribute to the overall performance. On the other hand, constructive methods start with a "small" network and then add unit(s) to an existing or new layer(s) till the required performance is achieved.

Chapter 3 introduces a sparse but structured DTRNN architecture, viz., the block-diagonal recurrent neural network (BDRNN). The motivations for this architecture are presented; together with a plausibility argument for the potential capability of the BDRNN architecture in modelling nonlinear dynamics, by considering how it relates to a fully recurrent DTRNN architecture. Two specific architectures of the block-diagonal feedback matrix that differ from each other in terms of complexity and degree of freedom are introduced. The BDRNN structure is then extended to include a multi-layer feedforward network in order to model any complex nonlinear mapping between the external inputs and the outputs of the system. The resultant structure, referred to as the feedforward block-diagonal recurrent neural network (FF-BDRNN), provides a framework to model both static and dynamic characteristics in a unified fashion.

While RTRL requires spatially nonlocal error gradient computation which is computationally expensive, BPTT calculates the error gradients using spatially local computations with a storage requirement that increases with the length of the training pattern. Chapter 4 makes the gradient computation in BPTT local in time, by recalculating the state vectors in the backward pass thus reducing the storage requirement for the state vectors. This, however, requires that the feedback weight matrix be inverted after each weight update. The block diagonal structure of the BDRNN lends itself to ensuring invertibility as the feedback weight matrix is easily inverted by simple manipulation and

scaling of its elements. Also, this technique of recalculating the state vectors is susceptible to numerical instability problems [Pearlmutter 1995] especially, in fully recurrent DTRNN. However, the reduced interaction between the state variables of the BDRNN helps in reducing the numerical instability problem. In addition, Chapter 4 specifically addresses the question of ensuring numerical stability of the algorithm. The recalculated state vector modified BPTT algorithm is described in this chapter.

Chapter 5 addresses the question of BDRNN stability. First, the conditions for local and global stability for the two BDRNN architectures are derived; next, the block diagonal structure of the BDRNN is exploited to find conditions that guide the network towards stability at each weight update during training. This problem is addressed in two steps: first, by devising a cost function that includes the desired stability margin as one of its components; next, by ingraining the stability margin component as part of the neural network architecture. To quantify the stability of the FF-BDRNN at each weight update during the training process, a stability function that is a measure of the norm of the feedback weight matrix W is formulated. To ensure the stability of the FF-BDRNN and hence of its training process, the cost function to be minimized during the training process includes a penalty term which is a function of the stability function. It is shown that, under certain non-restricting conditions, the stability function itself is formulated as a multi-layer feedforward neural network using first order perceptrons which augments the BDRNN structure and hence is ingrained in the FF-BDRNN architecture.

Chapter 6 presents several examples to illustrate the feasibility of the FF-BDRNN architecture and its stabilization to model a wide range of nonlinear processes and to demonstrate the performance of the modified BPTT trajectory learning algorithm and its numerical stability. Several of these simulation examples are well documented in literature and are useful in providing benchmark results. Finally, isolated-digit speech utterances are recognized using speech prediction techniques.

The cascading constructive trajectory learning algorithm outlined in Chapter 7, constructs a series of BDRNNs whose combined output is required to model the desired dynamic trajectory under consideration. Constructive learning is started with a basic BDRNN block that is directly trained on the desired trajectory being learned, while, each additional cascading block is trained on the residual error between the most recent estimate and the desired trajectory. The new estimate is a combination of the outputs of the basic block and previous blocks together with the output of the current block being trained. Such a constructive technique is ideal for learning nonlinear "decomposable" dynamics such as the chaotic Mackey Glass time series.

In Chapter 8. conclusions and suggestions for future work are presented.

2 A REVIEW OF CURRENT ARCHITECTURES AND TECHNIQUES FOR TRAJECTORY LEARNING, MONITORING STABILITY AND DETERMINING NETWORK SIZE IN RECURRENT NEURAL NETWORKS

2.1 INTRODUCTION

In discrete time recurrent neural networks (DTRNN) the network architecture, learning algorithm and stability of the network during learning play dominant roles in the generalization capability and optimal training of the network. In this chapter, a review of existing architectures, algorithms used for training DTRNN that learn trajectories, techniques used for monitoring and maintaining DTRNN stability, and methods adopted for determining DTRNN size is presented.

Currently, DTRNN architectures can be subdivided into two broad classes: fully recurrent architectures and locally recurrent globally feedforward architectures. Introduced in [Williams and Zipser 1989] is the single layer fully recurrent architecture in which the output of any neuron in the network is fedback to all neurons (including itself) with varying gain. The input is fed to all the neurons while, the output neurons are a subset of the total neurons of the network. The feedback weight matrix W of the fully recurrent architecture is full. The LRGF architecture consists of feedforward networks with self recurrent connections in some of the neurons (see [Tsoi and Back] for a survey). The LRGF network is said to have state feedback if the output of each neuron after the sigmoidal squashing is fedback to itself [Ku and Lee]. The LRGF network is said to have activation feedback if the activation of each neuron, ie., the output of the neuron before sigmoidal squashing is fedback to itself [Gori et al]. The feedback weight matrix of such LRGF networks is diagonal (representing the self recurrence) with all off-diagonal elements equal to zero. In addition, feedforward networks are said to have local synaptic feedback, if each synapse (connection between neurons) is modelled by an finite impulse response (FIR) filter, or an infinite impulse response (IIR) filter [Tsoi and Back]. While

the fully recurrent DTRNN is architecturally rich, it is difficult to monitor and stabilize its feedback weight matrix and it also suffers from slow convergence and long learning times. The LRGF architectures, because of their sparser feedback weight matrices, can be easily stabilized. They have also been shown to have smaller learning times and faster convergence for time series prediction problems [Tsoi and Back] and modelling finite state machines [Fahlman 1991].

Currently, trajectory training of DTRNN is generally accomplished using either real time recurrent learning (RTRL) [Williams et al 1989] or backpropagation through time (BPTT) [Werbos] algorithms or variations thereof e.g. [Williams and Peng 1990, Schmidhuber, Sun et al], (see [Pearlmutter 1995] for a survey). Both RTRL and BPTT algorithms compute the exact error gradient [Beaufays and Wan]. While, RTRL is computationally expensive due to the spatially nonlocal nature of its error gradient computation, BPTT uses spatially local computations to compute the error and requires large storage.

Currently, two techniques are used for monitoring stability issues; the first involves linearization of the state-space equation which provides information on the local stability properties of the equilibrium state of the system; the second involves using the Liapunov function or the contraction mapping theorem or both to derive sufficient conditions for the global stability of the system. Considering the global stability of a system is more appealing than that of the local stability, because, every globally stable system is also locally stable; the converse is not true. However, since global stability constraints are more stringent than local one's, satisfying the former may impose more restrictions on weight placements than satisfying the latter.

The methods used in determining network size can be broadly classified into destructive and constructive methods. Destructive methods start with a fairly large network and then remove weights or units that do not contribute to the overall performance. This is accomplished by removing the weights with smallest magnitude or by including a penalty

term in the output error function that encourages weight decay. On the other hand, constructive methods start with a "small" network and then add unit(s) to existing or new layer(s) till the required performance is achieved.

In the following sections, a detailed review of the above issues is presented.

2.2 REVIEW OF CURRENT DISCRETE TIME RECURRENT NEURAL NETWORK LEARNING ALGORITHMS

2.2.1 Fully recurrent discrete time neural networks

The Williams and Zipser DTRNN structure [Williams et al 1989] is a completely connected (fully recurrent) state-feedback system where the output variables are a subset of the state variables. The state and output equations describing a DTRNN with N state variables, M input variables and N_o output variables is given by:

$$\begin{aligned} x_i(k+1) &= f_a\left(\sum_{j=1}^{N} w_{ij}x_j(k) + \sum_{j=1}^{M} b_{ij}u_j(k)\right), \quad i = 1,\ldots,N \\ y_h(k) &= \sum_{j=1}^{N} \delta_{hj}x_j(k), \qquad\qquad h = 1,\ldots,N_o \\ \text{where} \quad \delta_{hj} &= 1, \quad \text{if } h = j \\ &= 0, \quad \text{if } h \neq j \end{aligned} \tag{1}$$

where $x_i(k)$, $u_j(k)$ and $y_h(k)$ are the i-th state variable, j-th input and h-th output elements respectively, at instant k, w_{ij}'s are the state-feedback weight parameters and b_{ij}'s are the external input weight parameters; δ_{hj}'s are the output weight parameters. $f_a(.)$ is a symmetric sigmoidal squashing function of the form:

$$f_a(x) = \frac{1-e^{-ax}}{1+e^{-ax}}, \quad a \geq 0 \tag{2}$$

with $\alpha = a$.

The training of the DTRNN described by (1) and (2) is based on the minimization of $J_p(k)$ which is the total squared output error over all the N_o output units for the p-th training pattern and is given by:

$$J_p(t) = \sum_{k=1}^{K_p} J_p(k)$$
$$\text{where } J_p(k) = \frac{1}{2}\sum_{h=1}^{N_o}(e_h(k))^2 \quad \text{with } e_h(k) = r_h(k) - y_h(k) \tag{3}$$

where K_p is the length of the p-th training sequence, $e_h(k)$ and $r_h(k)$ are the error and desired output for the h-th output unit at instant k. The techniques used for training DTRNN can be broadly classified into two categories based on the nature of their *exact* gradient computations. They are backpropagation through time (BPTT) which uses spatially *local* computations and real time recurrent learning (RTRL) which uses spatially *nonlocal* computations to find the gradient. In RTRL, the formulation of the gradients which updates the DTRNN weights during training requires recursive computation of the rate of change of state variables with respect to weights. In BPTT, the computation of the error gradient requires that the state, input, and error vectors be stored at each time instant and hence, the storage requirement for BPTT increases with the length of the training pattern. In both cases, the weights are updated after the presentation of each training pattern by accumulating the respective error gradients over the length of each training pattern.

The N x N state feedback weight matrix $W = \{w_{ij}\}$ is updated after each presentation of the training sequence, according to:

$$w_{ij}(t+1) = w_{ij}(t) - \mu_1 \frac{\Delta w_{ij}(t)}{K_p}, \qquad i, j = 1 \,..\, N$$
$$\text{with } \Delta w_{ij}(t) = \sum_{k=1}^{K_p} \frac{\partial J_p(k,t)}{\partial w_{ij}(t)} = \frac{\partial J_p(t)}{\partial w_{ij}(t)} \tag{4}$$

where t is the iteration index for updating the weights, μ_1 is the learning rate parameter and K_p is the length of the pattern presented for learning. Note that a given pattern can be divided into subpatterns or several patterns can be combined into one superpattern for the purpose of updating the weights. In such cases K_p represents the length of the corresponding pattern, subpattern or super pattern over which the accumulated instantaneous error gradient, Δw_{ij}, is computed prior to weight update.

The learning algorithm also updates the N x M input weight matrix B after K_p steps of the training sequence have been presented, according to:

$$b_{ij}(t+1) = b_{ij}(t) - \mu_1 \frac{\Delta b_{ij}(t)}{K_p}, \qquad i = 1 \,..\, N, \;\; j = 1 \,..\, M$$
$$\text{with} \quad \Delta b_{ij}(t) = \sum_{k=1}^{K_p} \frac{\partial J_p(k,t)}{\partial b_{ij}(t)} = \frac{\partial J_p(t)}{\partial b_{ij}(t)} \tag{5}$$

where Δb_{ij} is the accumulation of the instantaneous error gradient over the K_p steps of the training pattern.

The basic difference between the RTRL and the BPTT algorithm lies in the way the partial derivative $\partial J_p(k,t)/\partial w_{ij}(t)$ is computed. In the RTRL algorithm, this partial derivative is computed directly as an iterative computation, while in the BPTT algorithm it is computed in two passes using a chain rule.

2.2.2 Real-time recurrent learning (RTRL)

For clarity t, the iteration index for weight update, will be omitted in the following derivations. In the RTRL algorithm [Williams et al 1989], the partial derivative $\partial J_p(k)/\partial w_{ij}$ in (4) is given by:

$$\frac{\partial J_p(k)}{\partial w_{ij}} = -\sum_{h=1}^{N_o} (r_h(k)-y_h(k)) p_{ij}^h(k) \tag{6}$$

Here $p_{ij}^s(k)$ is expressed as:

$$p_{ij}^s(k) = \frac{\partial x_s(k)}{\partial w_{ij}} = f_a' \left(\sum_{\alpha_0=1}^{N} w_{s,\alpha_0} x_{\alpha_0}(k-1) + \sum_{\alpha_1=1}^{M} b_{s,\alpha_1} u_{\alpha_1}(k-1) \right) \cdot \left[\delta_{s,i} x_j(k-1) + \left(\sum_{\alpha_2=1}^{N} w_{s,\alpha_2} p_{ij}^{\alpha_2}(k-1) \right) \right] \tag{7}$$

where δ_{si} is the Kronecker delta function:

$$\begin{aligned} \delta_{s,i} &= 1 \qquad \textit{if } i = s \\ &= 0 \qquad \textit{otherwise} \end{aligned} \tag{8}$$

From (7) it is seen that the computation of $p_{ij}^s(k)$ is recursive and requires that the partial derivatives of the state variables $x_s(k\text{-}1)$ with respect to the elements of w_{ij}, $p_{ij}^s(k\text{-}1)$ at the k-1-th instant be stored. Hence, the exact gradient computation represented by (7) is spatially *nonlocal* by nature and requires a storage of O(Nm) where m is the number of weights in the feedback weight matrix W. In the case of a fully recurrent DTRNN the number of weights $m = N^2$ and the storage required is O(N^3). The gradient computation p_{ij}^s for RTRL is illustrated in Figure 2.1.

The partial derivative $\partial J_p(k)/\partial b_{ij}$ in (5) is given by:

$$\frac{\partial J_p(k)}{\partial b_{ij}} = -\sum_{k=1}^{K_p} \sum_{h=1}^{N_o} (r_h(k)-y_h(k)) \zeta_{ij}^h(k) \tag{9}$$

where $\zeta_{ij}^s(k)$ can be expressed as

$$\zeta_{ij}^s(k) = \frac{\partial x_s(k)}{\partial b_{ij}} = f_a' \left[\sum_{\alpha_0=1}^{N} w_{s,\alpha_0} x_{\alpha_0}(k-1) + \sum_{\alpha_1=1}^{M} b_{s,\alpha_1} u_{\alpha_1}(k-1) \right] \cdot [\delta_{s,i} u_j(k-1)] \tag{10}$$

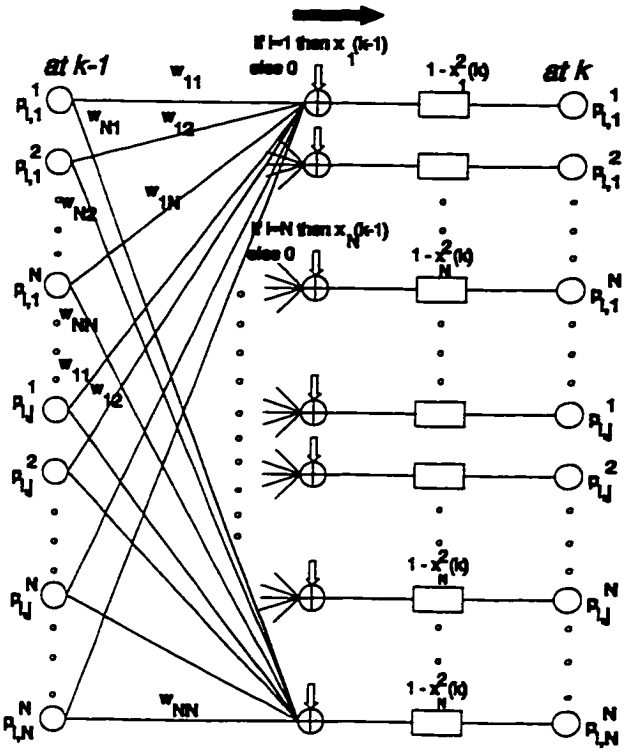


Figure 2.1 Gradient computation in the RTRL algorithm.

2.2.3 Backpropagation through time (BPTT)

Backpropagation through time [Werbos] is accomplished by first unfolding the recurrent network into a multi-layer feedforward network with one layer for each time instant k, and then applying standard backpropagation [Widrow, Hush and Horne] to calculate the exact error gradients used in updating the weights. The gradient computation in the BPTT algorithm is accomplished in two passes; the forward pass and the backward pass. In the forwardpass, the state vector $\{x_i(k)\}$ and the errors $\{e_h(k)\}$ are computed at all the time steps k in an epoch of length K_p, and stored. In the backwardpass, the error gradient is computed using the stored state and error vectors with spatially *local* computations.

Replacing the partial error derivative $\partial J_p(t)/\partial w_{ij}(t)$ in (4) with the ordered partial derivative of J_p with respect to w_{ij}:

$$\Delta w_{ij}(t) = \frac{\partial^+ J_p}{\partial w_{ij}} = \sum_{k=1}^{K_p} \frac{\partial^+ J_p}{\partial x_i(k)} \frac{\partial x_i(k)}{\partial w_{ij}} \tag{11}$$

where,

$$\begin{aligned} \frac{\partial x_i(k)}{\partial w_{ij}} &= f_d'(s_i(k)) x_j(k-1); \\ \frac{\partial^+ J_p}{\partial x_i(k)} \equiv \varepsilon_i(k) &= \sum_{j'=1}^{N} \left(\frac{\partial^+ J_p}{\partial x_{j'}(k+1)} \right) \left(\frac{\partial x_{j'}(k+1)}{\partial x_i(k)} \right) + \sum_{h=1}^{N_o} \frac{\partial J_p}{\partial e_h(k)} \frac{\partial e_h(k)}{\partial y_h(k)} \frac{\partial y_h(k)}{\partial x_i(k)} \end{aligned} \tag{12}$$

The state vector $\{x_j(k-1)\}$ used in (12), is computed using (1) and stored in the forward pass of the algorithm, while $\varepsilon_i(k)$ is computed in the backward pass from the value of $\varepsilon_i(k+1)$ using:

$$\begin{aligned} \varepsilon_i(k) &= \sum_{j'=1}^{N} \varepsilon_{j'}(k+1) w_{ij'}^T f_d'(s_{j'}(k+1)) - \sum_{h=1}^{N_o} e_h(k) \delta_{h,i} \\ \textit{with} \quad s_{j'}(k+1) &= \sum_{i'=1}^{N} w_{j'i'} x_{i'}(k) + \sum_{i'=1}^{M} b_{j'i'} u_{i'}(k), \qquad j' = 1, \ldots, N \end{aligned} \tag{13}$$

where w^T_{ij}'s are the elements of the transpose of the state-feedback weight matrix. The initial value $\varepsilon_i(K_p)$ given by:

$$\varepsilon_i(K_p) = -\sum_{h=1}^{N_o} e_h(K_p)\delta_{h,i} \tag{14}$$

The partial derivative $\partial J_p(t)/\partial b_{i,j}(t)$ in (5) is computed using the chain rule:

$$\frac{\partial^+ J_p}{\partial b_{i,j}(t)} = \sum_{k=1}^{K_p} \frac{\partial^+ J_p}{\partial x_i(k)} \frac{\partial x_i(k)}{\partial b_{i,j}} \tag{15}$$

where,

$$\frac{\partial x_i(k)}{\partial b_{i,j}(k)} = f_a'(s_i(k))u_j(k-1); \qquad \frac{\partial J_p(k)}{\partial x_i(k)} = \varepsilon_i(k)$$
$$\text{with} \quad s_i(k) = \sum_{j=1}^{N} w_{i,j}x_j(k-1) + \sum_{j=1}^{M} b_{i,j}u_j(k-1) \tag{16}$$

From (12) and (13), it is seen that the error vector $\{e_h(k)\}$, the state vector $\{x_i(k)\}$ and the input vector $\{u_j(k)\}$ have to be stored in the forward pass. This requires $O(N_oK_p)$ storage for the error vector, $O(NK_p)$ storage for the state vector and $O(MK_p)$ storage for the input vector that increase with the length of the training pattern. From (13), it is seen that computing the gradient at any time instant k requires $O(m)$ spatially local computations. The computations are *local* in nature, since computing the error gradient with respect to a weight at any layer k requires information only from other nodes to which it connects. The gradient computation for the BPTT algorithm is illustrated in Figure 2.2.

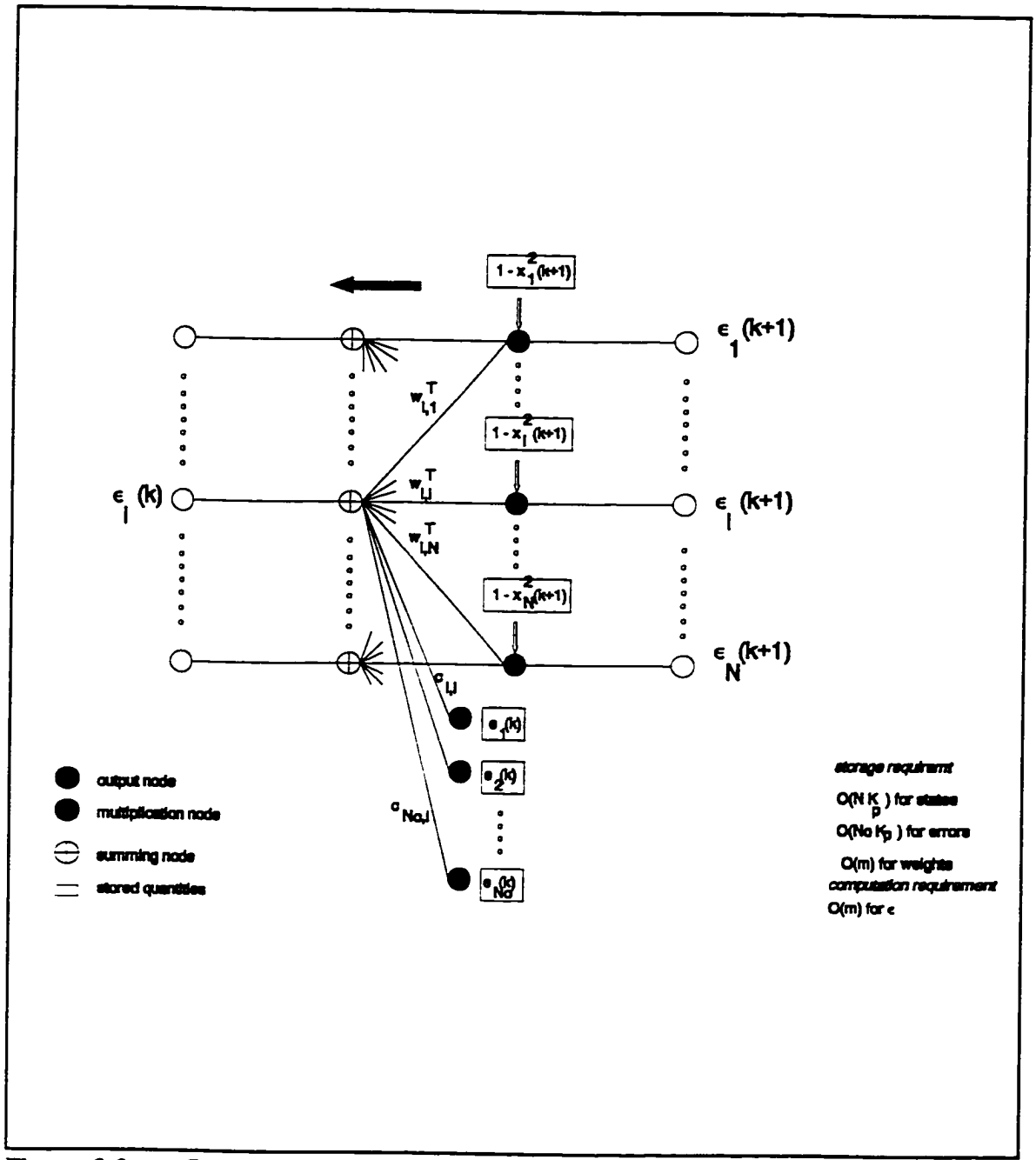


Figure 2.2 Gradient computation in the BPTT algorithm.

2.2.4 Variations of BPTT

Let us now review two popular techniques used to approximate the partial error derivative $\partial J_p(k,t)/\partial x_i(k) = \varepsilon_i(k)$ in (13).

2.2.4.1 Truncated backpropagation through time or BPTT-t_o step

The idea is to use only the most recent error $e_h(k)$, but unlike the regular BPTT where the error from earlier time steps is also used, to *approximate* the partial error derivative $\partial J_p(k,t)/\partial x_i(k)$ at each time step k. Hence, any older information is forgotten. This error is percolated through t_0 layers of the unfolded network to compute the appropriate weight changes at each time step k. Thus, the BPTT-t_o step algorithm requires that the network state variables and network inputs of only the last t_o-time steps be stored.

A backwardpass through the most recent t_o-time steps (instead of K_p-time steps which represents the entire history of the network in regular BPTT) is performed by approximating the partial error derivative $\partial J_p(k,t)/\partial x_i(k) = \varepsilon_i(k)$ in (13) by:

$$\varepsilon_i(\tau) = \sum_{j=1}^{N} w_{ij}^T f_b'\left(\sum_{j=1}^{N} w_{ij} x_j(\tau+1) + \sum_{j=1}^{M} b_{ij} u_j(\tau+1)\right)\varepsilon_j(\tau+1), \quad k-t_o \le \tau < k \tag{17}$$

with the initial value $\varepsilon_i(k)$ given by:

$$\varepsilon_i(k) = -\sum_{h=1}^{N_o} e_h(k)\delta_{h,i} \tag{18}$$

The weights are updated, as given in (4), at every time step k in the training pattern, with Δw_{ij} the accumulation of the instantaneous error gradient over the t_o steps of the training pattern given by:

$$\Delta w_{ij}(k) = \sum_{\tau=k-t_o}^{k} \frac{\partial J_p(\tau,k)}{\partial w_{ij}(k)} \quad \textit{for} \quad i, j = 1 \,..\, N \tag{19}$$

The formulation of the partial error derivative $\partial J_p(\tau,k)/\partial w_{i,j}(k)$ is similar to that in (11) and (12), with τ replacing k. This process is repeated at each time step k and makes BPTT-t_o step, t_o times as computationally intensive as regular BPTT, while reducing the storage requirements for the state variables from $O(K_pN)$ to $O(t_oN)$.

2.2.4.2 Williams-Peng BPTT

The Williams-Peng BPTT [Williams and Peng 1990] stores the network state variables and network inputs of the last t_o-time steps and in this respect is similar to truncated-BPTT. However, the error in the t_o' most recent time steps with ($t_0' \leq t_0$) are used, instead of the error in the most recent time step as in truncated BPTT, to *approximate* the error gradients at these t_o' time steps. Thus, it is seen that the Williams-Peng BPTT will behave like truncated BPTT, if $t_0' = 1$. Similarly, if $t_0' = t_0 = K_p$, then it behaves like regular BPTT.

A backward pass through the most recent t_o-time steps is performed by approximating the partial error derivative $\varepsilon_i(k)$ in (13) by:

$$\varepsilon_i(\tau) = -\sum_{h=1}^{N_o} e_h(\tau)\delta_{h,i} + \sum_{j=1}^{N} w_{i,j}^T f_b'\left(\sum_{j=1}^{N} w_{i,j}x_j(\tau+1) + \sum_{j=1}^{M} b_{i,j}u_j(\tau+1)\right)\varepsilon_j(\tau+1), \quad \text{if } k-t_o' \leq \tau < k$$

$$= \sum_{j=1}^{N} w_{i,j}^T f_b'\left(\sum_{j=1}^{N} w_{i,j}x_j(\tau+1) + \sum_{j=1}^{M} b_{i,j}u_j(\tau+1)\right)\varepsilon_j(\tau+1), \quad \text{if } k-t_o \leq \tau < k-t_o' \tag{20}$$

with the initial value $\varepsilon_i(k)$ given by:

$$\varepsilon_i(k) = -\sum_{h=1}^{N_o} e_h(k)\delta_{h,i} \tag{21}$$

The weights are updated every t_0' time steps, with $\Delta w_{i,j}$ the accumulation of the instantaneous error gradient over the t_o steps of the training pattern given by:

$$\Delta w_{ij}(t) = \sum_{\tau=k-t_o}^{k} \frac{\partial J_p(\tau,t)}{\partial w_{ij}(t)} \qquad i, j = 1 \;..\; N \tag{22}$$

The partial error derivative $\partial J_p(\tau)/\partial w_{ij}(t)$ is identical to that in (20) and (21). This process is repeated every t_o' time steps. Therefore in BPTT-(t_o:t_o'), the computations required to calculate the gradient is reduced by a factor of t_o' as compared to BPTT-t_o. However, BPTT-(t_o:t_o') has the same storage requirement as BPTT-t_o.

2.2.4.3 BPTT-RTRL hybrid

The BPTT-RTRL hybrid algorithm [Schmidhumber] computes the *exact* gradient by first decomposing the calculation of the gradient into blocks of length t_o, and then performing regular BPTT (as described in section 2.2.3) on each block. RTRL calculations are performed to integrate the results of the BPTT calculations of each block into the results obtained from the previous blocks. BPTT-RTRL has an average computational requirement of $O(N^3)$ if t_o is chosen equal to N, and has the same storage requirement as the RTRL algorithm.

2.2.5 Discussion on trajectory learning algorithms

Table 2.1 summarizes the computation time required per gradient computation, quantities stored and their storage requirement, and the nature and type of gradient computation for each of the algorithms discussed in this section. In this table, N is the number of state variables, M is the number of external inputs, N_0 is the number of outputs, m is the number of weights in the feedback weight matrix, K_p, $t_{0,}$, and t_0' represent the number of time steps for which the various quantities are stored in the different algorithms.

Table 2.1 Comparison of various DTRNN algorithms used for trajectory learning.

Algorithm	Time per gradient	Quantities stored	Storage requirement	Nature	Local / nonlocal
RTRL	$O(N^2m)$	gradient $p_{ij}(k)$ weights	$O(Nm)$	forward, exact	nonlocal in space.
BPTT (reg)	$O(m)$	state, input, o/p error vectors; weights	$O(K_p(N + M + N_o)$ $O(m)$	backward, exact	local in space.
Truncated BPTT	$O(t_0m)$	state, input vectors; The latest o/p error; weights	$O(t_0(N+M))$ $O(N_o)$ $O(m)$	backward, approx-imate	local in space.
William-Peng BPTT	$O((t_0/t_0')m)$	state, input vectors for t_0 steps; o/p error for t_0' steps; weights.	$O(t_0(N+M))$ $O(t_0'N_o)$ $O(m)$	backward, approx-imate	local in space.
BPTT-RTRL hybrid	$[O(m) + O(N^2m)]/N = O(Nm)$	state, input and error vector for $t_0=N$ steps; weights	$O(Nm)$ in RTRL phase	backward / forward in BPTT (reg) / RTRL. exact	nonlocal in space

The RTRL algorithm requires $O(N^2m)$ time units to compute the exact error gradient at each time step, and $O(Nm)$ storage units. The BPTT algorithm requires the least computational effort to compute the exact error gradient, and a storage requirement of $O(K_pN)$ that increases with the length of the training pattern. Hence, if the state variables can be recomputed in the backward pass of BPTT with a small increase in the computation time required to perform these state variable recomputations, then, the state vector storage requirement can be reduced. Ideally, this would still require $O(m)$ time to compute the error gradient at each time step, while, the storage is reduced to $O(N)$ and is the best that can be achieved. However, as can be seen from (1), the technique of recomputing the state vector in the backward pass would require inversion of the feedback weight matrix. Generally, matrix inversion is a nonlocal operation and may result in numerical stability problems if the feedback weight matrix is ill conditioned. Hence any algorithm that involves recomputation of the state vector would have to address the twin issues of

(i) matrix invertibility and

(ii) numerical stability of the state vector recomputations.

2.3 CURRENT TECHNIQUES FOR MONITORING THE STABILITY OF RECURRENT NEURAL NETWORKS

2.3.1 Stability of network dynamics and learning dynamics

The feedback structure of the RNN necessarily raises the question of stability, both of the network and during its training. While, for a stable network, the stability of training is well understood and can be ensured by suitably choosing the learning rate [Almeida], there are several unresolved issues in monitoring and maintaining the stability of the network itself at each weight update.

The autonomous dynamics of the DTRNN, described by (1), can be described by eliminating the forcing function from (1). It can be seen that the autonomous network dynamic is given by:

$$x_i(k+1) = f_a\left(\sum_{j=1}^{N} w_{ij}x_j(k)\right), \qquad i = 1,\ldots,N \tag{23}$$

From (23), it can be seen that the stability of network dynamics depends on the feedback weight matrix W. Similarly, after eliminating the forcing functions from (7), it is seen that the learning dynamics for the RTRL learning algorithm is given by:

$$\frac{\partial x_s(k)}{\partial w_{ij}} = p_{ij}^{s}(k) = f_a'(.)\left[\sum_{\alpha_2=1}^{N} w_{s,\alpha_2} p_{ij}^{\alpha_2}(k-1)\right] \tag{24}$$

where the maximum bound on $|f_a'(.)|$ is $a/2$. From (24) it can be seen that, for a DTRNN whose weights are updated using the RTRL algorithm, the stability of the learning dynamics depends on W. Hence, ensuring the stability of the network dynamics will automatically ensure the stability of the learning dynamics when the RTRL algorithm is used to update the elements of W.

Similarly for the BPTT algorithm and its variations, after eliminating the forcing functions from (13), (17), (20), it can be seen that the learning dynamics are given by:

$$e_i(k) = \sum_{j=1}^{N} w_{ij}^{T} f_b'(.) e_j(k+1) \tag{25}$$

From (25), it can be seen that for BPTT and its variants, the stability of learning dynamics depends on the transpose of the feedback weight matrix, i.e., on W^T. Since the conditions that ensure the stability of a matrix W are the same as those that ensure the stability of its transpose W^T, therefore, for BPTT and its variants too, ensuring the stability of the network dynamic will automatically ensure the stability of the learning dynamics. Summarizing, irrespective of the learning algorithm used, the formulation of the gradients which updates the DTRNN weights has the same dynamics as the DTRNN

state variables at a given instant in the training phase. Therefore, the stability of the recursive computations requires that the DTRNN itself be stable at its equilibrium point at each weight update [Williams and Zipser 1989, Almeida]. A major difficulty in monitoring the stability of a fully connected DTRNN relates to the computation of a suitably selected norm of its feedback matrix at each weight update during training.

Another issue worth investigating relates to whether one would like to ensure the local or global stability of the DTRNN under consideration. Local asymptotic stability conditions ensure that, for bounded inputs, the DTRNN will ultimately settle down to a steady state provided that the initial state variables of the DTRNN are already in the neighbourhood of the equilibrium state. However, global absolute stability conditions ensure that the DTRNN will settle down to a steady state irrespective of the choice of initial state variables and bounded inputs to the system. While, ensuring the global stability of a DTRNN system is sufficient to ensure stability of its learning, it may place too many restrictions on the movements and assignment of weights of its feedback matrix, thus restricting its learning ability. On the other hand, ensuring local stability may be sufficient to ensure stable learning in the neighbourhood of the equilibrium point of the system, and may not place as many restrictions in the movements or placement of the feedback weight matrix. Here, the following sections will review sufficient conditions to be satisfied by W to ensure the local and global stability respectively.

2.3.2 Global stability - Liapunov approach - contractive mapping approach - limitations

The global stability of a nonlinear dynamic system can be investigated by the direct method of Liapunov. The Liapunov function is a continuous scalar function of the state variables of the system. The properties of the derivative of the Liapunov function of the autonomous state-space system with respect to the equilibrium point of the system are investigated to find if the equilibrium point is asymptotically stable. The existence of the Liapunov function is a sufficient but not necessary condition for the global stability of the

system. While the Liapunov function provides a mathematical basis for investigating the global stability of a nonlinear dynamic system, there does not exist any formal mathematical results that can be used to find the Liapunov function. It is generally found by trial and error [Haykin 1994] or by guessing a function appropriate to an application [Hunt et al]. However, inability to find a suitable Liapunov function does not necessarily prove that the system under investigation is unstable.

Conditions for the global stability of the network dynamics of a DTRNN, given a specific architecture, have been studied by a number of authors [e.g., Gori et al, Li, Jin et al 1994 and 1996, Simard et al]. The global stability of a nonlinear dynamic system was also investigated using the contraction mapping theorem by which it was shown that [Kelly], if (1) is a contractive system, then (1) has an unique equilibrium point that is globally asymptotically stable. The nonlinear dynamic system (1) is contractive if it satisfies the following:

$$\beta |W| < 1 \tag{26}$$

where β is the bound on the $|f'(x)|$ i.e., the bound on the magnitude of the derivative of the sigmoidal function (2) and $|W|$ is a suitably defined matrix norm. It can be easily noted that $\beta = a/2$ for (2). Hence, for the matrix norms $\|W\|_1$, $\|W\|_2$, $\|W\|_\infty$, the following are three sufficient conditions for the global stability of (1), [e.g., Jin et al 1996, Barnes]:

$$\begin{aligned} \|W\|_1 &= \max_j \sum_{i=1}^{n} |w_{ij}| < \frac{2}{a} \\ \|W\|_2 &= \left[\lambda_{max}((W)^T W) \right]^{1/2} < \frac{2}{a} \\ \|W\|_\infty &= \max_i \sum_{j=1}^{n} |w_{ij}| < \frac{2}{a} \end{aligned} \tag{27}$$

where $\lambda_{max}(W^T W)$ is the maximum eigenvalue of the matrix $W^T W$.

[Jin et al 1994] used the intermediate value theorem to prove that the nonlinear dynamic system (1) has at least one equilibrium point $x^* \in [-1,1]^N$, for any bounded input $u_i(k)$ and complex feedback weight matrix W. A sufficient condition for the absolute stability of system (1) using Ostrowski's theorem is:

$$|w_{ij}| + (1)^{\gamma-1} R_i^{\gamma} C_i^{1-\gamma} < \frac{2}{a} \qquad i = 1,\ldots,N \tag{28}$$

where $\gamma \in [0,1]$ is arbitrarily chosen, and R_i and C_i are the deleted row and column sums of W defined as:

$$R_i = \sum_{j=1, j\neq i}^{N} |w_{ij}| \;, \quad C_i = \sum_{j=1, j\neq i}^{N} |w_{ji}| \tag{29}$$

For $\gamma = 1$, (29) degenerates to the $\|W\|_1$ condition given in (28):

$$\sum_{j=1}^{N} |w_{ij}| < \frac{2}{a} \tag{30}$$

A sufficient condition for the absolute stability of (1) can also be obtained using the similarity transformation approach to describe a region of absolute stability [Jin et al 1994]. For a nonsingular N x N matrix T, it is known that the transformation $T^{-1}WT$ has the same eigenvalues as W. Considering the particular case for which T is a diagonal matrix described by diag$[t_1, t_2, \ldots, t_N]$ with $t_i > 0$ a sufficient condition for absolute stability is:

$$|w_{ij}| + \frac{(1)^{\gamma-1}}{t_i^{2\gamma-1}} (R_i^t)^{\gamma} (C_i^t)^{1-\gamma} < \frac{2}{a} \qquad i = 1,\ldots,N \tag{31}$$

where $\gamma \in [0,1]$ is given, and R_i^t and C_i^t are the deleted row and column sums of W defined as:

$$R_i^t = \sum_{j=1,j\neq i}^{N} t_j|w_{ij}| \;, \quad C_i^t = \sum_{j=1,j\neq i}^{N} \frac{|w_{ji}|}{t_j} \tag{32}$$

The globally asymptotical stability (28) and absolute stability conditions (29) were shown to depend on the parameters and connection weights of the network. However, using these conditions to ensure the continued stability of the network during the training process has not been explored in the literature.

2.3.3 A sufficient condition for local asymptotical stability of a fully recurrent neural network

A sufficient condition for the local stability of the DTRNN for a bounded input is determined by linearizing the state equation (1) with zero external input around the equilibrium point:

$$\Delta x_i(k+1) = f_a'\left(\sum_{j=1}^{N} w_{ij}x_j(k)\right)\left[\sum_{j=1}^{N} w_{ij}\Delta x_j(k)\right], \qquad i=1,..,N \tag{33}$$

where $f'(.)$ is the derivative of $f(.)$. Defining $X(k)$ as the state vector $\{x_i(k)\}$, at the equilibrium point $X = 0$, equation (37) becomes:

$$\Delta X(k+1) = f_a'(0)[W\Delta X(k)] \tag{34}$$

The system is locally stable at the equilibrium point, if and only if

$$\left|\lambda_i\,[\,f_a'(0).W\,]\,\right| \leq 1.0, \qquad i=1,.....,N \tag{35}$$

where $\lambda_i(A)$ is the i-th eigenvalue of the square matrix A. The slope of $f_a(.)$ at the equilibrium point $x = 0$ is:

$$f_a'(0) = \frac{a}{2} \tag{36}$$

Therefore, a sufficient condition for the *local* stability of the DTRNN with a nonsingular feedback weight matrix W is given by

$$| \lambda_i(W) | \leq \frac{2}{a} \qquad i=1,..,N \tag{37}$$

From (37), it is clear that the local stability of this system can be ensured by monitoring the eigenvalues of the system at each weight update.

2.3.4 Structural relevance to stability issues

In the case of DTRNN with only diagonal feedback the global and local stability conditions (27) and (37) are equivalent and can be easily ensured by making sure that the magnitude of the self-weights is less than unity. However, in fully recurrent DTRNN satisfying the global stability constraints based on the matrix norms $\|W\|_1$, $\|W\|_\infty$, requires that $|w_{i,j}| \leq 1/N$ and therefore may place more restrictions on weight placements than satisfying $\|W\|_2$.

2.3.5 On-line stabilization techniques

[Simard et al] have proposed a general, but complex, approach to the global stabilization of fully recurrent DTRNN using the contraction mapping theorem. Equation (1) can also be expressed as the mapping K given by:

$$x_i(t+1) = K(x_1(t), x_2(t), \ldots, x_N(t)) \tag{38}$$

The mapping K is contracting in the direction of the unitary vector V if

$$\|K'(X).V\| \leq 1 \tag{39}$$

where K' is the Jacobian matrix of K. [Simard et al] used this contraction mapping result to devise an energy function $E_s(X,V)$ that makes the mapping K contracting at X in the

direction V where V is chosen to be a unitary vector corresponding to the largest eigenvalue of $K'(X)$. A learning algorithm, based on the RTRL algorithm, that minimizes the energy function $E_s(X,V)$ ensures globally stable convergence. However, determining $\partial E_s(X,V)/\partial w_{i,j}$ is complex, and difficult as it involves nonlocal computations to recursively compute the rate of change of state variables with respect to the weights. An alternate method proposed involves matrix inversion which is also nonlocal and complex. In addition nonlocal computations are required to find the largest eigenvalue of $K'(X)$.

DTRNNs with constrained or special network structures that inherently ensure stability have been studied by several researchers [Sato, Gori et al, Bruck, Vidyasagar, Ku and Lee]. For example, Gori et al considers a multi-layer feedforward architecture which contains some self-recurrent neurons. Each self recurrent neuron has a parameter χ_i associated with it. The learning algorithm is based on the minimization of the total mean square error with respect to χ_i and the gradient computation is given by $\partial J_p(k,t)/\partial \chi_i(t)$. The parameter χ_i associated with each self-recurrent weight is updated according to:

$$\chi_i(t+1) = \chi_i(t) - \mu_1 \frac{\Delta\chi_i(t)}{K_p}, \qquad i = 1 \,..\, N$$
$$\textit{with} \quad \Delta\chi_i(t) = \sum_{k=1}^{K_p} \frac{\partial J_p(k,t)}{\partial \chi_i(t)} \tag{40}$$

where t is the iteration index of the weights. $\Delta\chi_i$ is the accumulation of the instantaneous error gradient over the K_p steps of the training pattern and μ_1 is the learning rate parameter. Stability is ensured by the following relationship between the self recurrent weight $w_{i,i}$ and χ_i:

$$w_{i,i}(t) = B \frac{(1 - e^{-\chi_i(t)})}{(1 + e^{-\chi_i(t)})} \tag{41}$$

where $B < 1$, is a decay constant. Implementing (42) restricts the magnitude of the self-recurrent weights to less than unity and hence satisfies both local stability (33) and the sufficient conditions for global stability (28).

[Ku and Lee] have proposed a method to ensure stability and convergence of training of a diagonally recurrent neural network by an adaptive learning rate algorithm. In this approach the training rate is varied so as to maintain stability. The initial values of the state feedback weight matrix W were chosen to be small positive numbers that ensured that its eigenvalues are well inside the unit disk. Thereafter, the weights were updated after each presentation of the input pattern according to the weight update rule:

$$W(t+1) = W(t) - \mu \Delta W(t) \tag{42}$$

where $\Delta W(t)$ is the Jacobian matrix given by:

$$\Delta W(t) = \left\{ \frac{\partial J_t}{\partial w_{ij}} \right\} \tag{43}$$

and μ is the maximum value that satisfies

$$| \lambda_i [W(t+1)] | \leq 1.0, \qquad i=1, ..., N, \qquad 0 \leq \mu \leq \mu_{max} \tag{44}$$

where μ_{max} is the maximum limiting value that the learning rate can take to guarantee convergence of learning. μ_{max} for diagonally recurrent neural network is shown to depend on the norm of the partial derivative of the output with respect to the input weight matrix [Ku and Lee]. The validity of this approach is evident for small μ_{max} from the first-order small perturbation theory which states that the eigenvalues of a matrix vary continuously with reference to the matrix elements for small perturbations in the matrix elements [Stewart]. Extending this technique to the general DTRNN has several disadvantages associated with it. First, it requires finding the largest eigenvalue of W at every weight update which may be computationally expensive. Second, and more importantly, it does not make use of this knowledge of the largest eigenvalue to ensure stability, but, rather limits μ at a value for which the DTRNN is marginally stable. Hence, when the

eigenvalues of W are close to the unit circle, the μ that satisfies (42) is very close to zero for a majority of the training inputs used. This implies that in the above formulation, when the system reaches a marginal stability region, it is rendered incapable of further effective learning.

2.3.6 Discussion on stability issues

A major difficulty in monitoring the stability of a fully connected DTRNN relates to the computation of a suitably selected norm of its feedback matrix at each weight update during training. From (27) and condition (ii) of (37), it is clear that the local or global stability of the DTRNN and hence of its training can be addressed by monitoring the eigenvalue with the largest magnitude of W or $(W)^T W$, respectively, at each weight update. The fully connected state feedback system makes this monitoring a difficult and computationally expensive problem. This is one of the motivations for considering a DTRNN with a sparse feedback weight matrix, where it is easily possible to monitor the magnitude of the eigenvalues in (27) and (37) and hence address stability in a simpler framework using local computations.

2.4 CURRENT TECHNIQUES FOR DETERMINING NEURAL NETWORK SIZE

The network architecture directly impacts the generalization capabilities of the network and the time required to train it. So far, the selection of a suitable architecture has mainly been empirical and there are no formal methods to customize or select the appropriate network structure. Typically, the number of hidden units in a multi layer neural network is determined by trial and error. Most work in this area is confined to feedforward networks, although they can readily be extended to DTRNNs.

2.4.1 Current techniques for determining network size in nonrecurrent networks

2.4.1.1 Trial and error method

Reported in [Hirose et al] is a trial and error approach that involves incrementing the number of hidden layer nodes by one, if the error does not decrease by more than a certain threshold value. The resultant network is trained using backpropagation and the process of incrementing the nodes is repeated until the network converges. Thereafter the number of hidden nodes is decremented by one, the reduced network is retrained and the process of decrementing the nodes is repeated until the network no longer converges. The architecture with the least number of hidden nodes for which the network converges is taken to be the optimal architecture.

2.4.1.2 Destructive methods for determining optimal network size

[Mozer et al] addresses the problem of minimizing the number of hidden units in a backpropagation network by estimating the sensitivity of the error function to the elimination of each of the hidden nodes. A non-quadratic error function which sums the magnitude of the error between the desired and actual output was used, since this error function provides a better estimate of sensitivity when the actual output activation is close to the desired output activation. A disadvantage of this technique is that it is computationally intensive, since two separate backpropagation phases are required. The first phase computes the weight updates, and the second computes the sensitivity measures. At the end of training the nodes with the smallest sensitivity numbers are eliminated. Karnin [Karnin] prunes a backpropagation network to its optimal size by estimating the sensitivity of the error (cost) function to the removal of each connection. The error function used is the familiar quadratic error function. This technique is better than the Mozer technique, since the sensitivity computation uses terms that are already available during training. An array of sensitivity numbers, one for each connection in the

network is maintained. At the end of training the connections with the smallest sensitivity numbers are pruned.

2.4.1.3 Constructive methods for determining optimal network size

Dynamic construction (node creation) methods in feedforward networks can be broadly classified into two categories depending on the type of weight update. In the first method, the existing network is frozen and only the weights connecting the new node(s) to the rest of the network are updated [Fahlman and Lebiere 1990]. In the second method the entire network is retrained after each addition of new node(s) e.g. [Ash, Frean, Moody et al]. Node creation is started when a measure associated with the network is below (or above) a certain threshold. The measures commonly used are squared error [Frean, Azimi-Sadjadi et al, Moody et al], error slope [Ash], and error variance [Hanson, Fahlman and Lebiere 1990].

The dynamic node creation method [Ash] is designed for backpropagation networks in which a new node is added to the hidden layer(s) if the slope of the average error curve in a particular window of training epochs falls below a certain user defined threshold. Node creation is disabled when the data mapping is learned to a user specified precision.

The upstart algorithm [Frean] starts with an input and an output layer. Each output node (parent) that wrongly classifies the training patterns, builds a couple of daughter nodes by interpolating them between the input and output layers (i.e., in a hidden layer). These daughter nodes aim to correct some of the mistakes made by the output node. This process is continued until all patterns are correctly classified by the output nodes.

In the meiosis network [Hanson], each connection weight has a mean and variance measure associated with it. The algorithm measures a coefficient of input and output variance (ratio of standard deviation to the mean) and mean at each hidden node. When

the composite variance is > 1.0, the node is split into two hidden nodes. Each new node is assigned half the variance of the old node with a new mean centred at the old mean. This node splitting process is continued until convergence.

In [Waibel et al] recognition of several classes of consonants was demonstrated using a construction method of a feedforward network. Here the authors start out by training separate modular multi-layer feedforward networks (whose architectures were arrived at by studying the individual characteristics of the consonants classes) for recognizing certain features unique to each consonant class. The weights of these modular subnetworks are then frozen and then integrated into a single larger network by adding some "glue" units and training only the weights that connect these "glue" units to the various modular subnets.

The cascade correlation learning architecture [Fahlman and Lebiere] as the name suggests is used to construct a network with a cascading architecture. The architecture starts with a minimal number of neurons in a single-hidden layer whose input and output weight matrices are trained until there is no significant improvement in the error. In the cascading architecture each new hidden node that is created is fully connected to the input nodes and to all preexisting hidden nodes. At this stage the output of this new node is not connected to the output nodes. After each pass of the training set the input weights to the new node are updated, so as to maximize the magnitude of the covariance between the output of the new node and the residual error observed at each output node over the complete set of training patterns. The input weights to the new node are now frozen and the new node is connected to all the output nodes. The output weights from the new node are now updated to maximize the above error covariance. The new node is installed when there is no significant improvement in this covariance. The process of node creation and installation is continued until convergence.

[Moody et al] developed a dependence identification method for constructing feedforward networks designed to work with continuous training problems. The algorithm uses the concept of linear dependence to classify the total training patterns into groups of similar patterns. Initially a single layer neural network is constructed that correctly classifies a group of similar patterns that is a subset of the total training patterns. At this stage a layer of hidden nodes is created to classify the data into groups of similar patterns such that every pattern in the training set is correctly classified by at least one hidden node. The outputs of the first hidden layer are treated as inputs to the next hidden layer, and the process of linear dependence identification is repeated. New layers are added until the number of layers equals a user defined maximum, or the network error is below a certain threshold.

[Draelos and Hush] construct a composite network whose each stage consists of a single hidden-layer feedforward network. The output of the composite network is required to approximate a target function. This approach starts out by training a single hidden layer feedforward network on the target function. At any stage, when the error performance is not satisfactory, the weights in the previous and current stages are frozen and a single layer network is added. The hidden layer at any stage of the composite network is trained on the residual error between the most recent estimate (computed at the previous stage) and the target function. At any stage in the construction process, the number of nodes in the hidden layer are determined by functional approximation techniques which approximate the target/residual error function at each stage by a series of hyperplanes. Each of these hyperplanes is replaced by a set of piece-wise linear functions. A neuron in the hidden layer describes each of these piece-wise linear functions.

2.4.2 Current techniques for determining recurrent network size

There are several approaches to dynamic construction methods in DTRNNs such as structures supporting genetic algorithms [Angeline et al, McDonell], cascading

architectures [Fahlman 1991], fully connected DTRNNs [Giles et al 1995], Gaussian radial basis function RNNs [Obradovik].

Treated in [Fahlman 1991] is the recurrent version of the cascade correlation architecture [Fahlman and Lebiere 1990], in which an arbitrary minimal number of neurons in a single hidden layer feedforward network with self recurrent connections is trained until there is no significant improvement in the error. At this stage, if the performance of the network is not satisfactory, the weights of the existing network are frozen and a single hidden self recurrent neuron is added. The added hidden neuron is connected to all the external inputs and to all preexisting neurons. These weights are updated so as to maximize the covariance between the residual error of the frozen network and the output of the new neuron. At this stage the input weights to the hidden neuron are frozen and it is connected to the output units. The output weights and the self recurrent weight are trained in a similar fashion. The process of addition of hidden neurons is continued until the desired performance is achieved. Simulations performed to learn finite state grammar, show that the training time for this algorithm is fast compared to the traditional technique that starts out with a single layer fully recurrent architecture. This may be so because only the weights to and from the newly added neuron are trained. The time-delayed recurrent self loop gives the recurrent cascade correlation network state memory.

[Giles et al 1995] start with a fully recurrent DTRNN with hard threshold neurons and add one neuron at a time to the recurrent layer after a fixed number of epochs until the network has learned all the training samples. Each added neuron is fully connected to the existing network and is initially assigned very small values close to zero. This procedure for assigning weights is thought to preserve the knowledge acquired by the network upto that point. The old weights start with their trained values, however they are not frozen and are still trainable during and after the addition of the new recurrent nodes. This process of adding new nodes is continued until convergence. They have demonstrated that this

technique is successful in constructing minimal size networks for learning a number of finite state automata.

2.4.3 Discussion on techniques used to determine DTRNN size

The destructive technique is computationally expensive, since most of the training time is spent on a network of a larger size than necessary. Also large networks (larger than required) may overfit the data being learnt leading to poor generalization capability. The same data can be mapped by networks with different sizes, since, destructive techniques start with a larger network size, this approach may lead to a nonoptimal architecture. The Giles fully recurrent DTRNN construction method is computationally intensive since the entire network is retrained after the addition of each node. Moreover each node is added arbitrarily after a fixed number of training epochs. The cascade correlation architecture [Fahlman 1991] is a freeze and learn approach that adds hidden self-recurrent neurons one per cascading layer. However, it constructs "deep" networks that have limited state memory. The functional approximation feedforward network construction technique [Drealos and Hush] uses the freeze and learn approach together with the residual error to construct a cascading feedforward neural network. However, this technique requires considerable preprocessing of the target function in order to approximate it by a series of hyperplanes and piecewise linear functions. Hence, any DTRNN construction technique that involves the freeze and learn approach, together with automatic functional approximation (without or with minimal preprocessing) by a series of cascading DTRNN modules would enable the composite DTRNN network to approximate the target function satisfactorily in terms of computation time required, performance and size of the network.

2.5 CONCLUSION

This chapter provides a review of common DTRNN architectures, viz., the fully recurrent and the locally recurrent architectures. The current algorithms for training DTRNN viz., real time recurrent learning (RTRL), backpropagation through time (BPTT) and its variations are reviewed. The existing concepts of local and global stability of DTRNN and its training and approaches to their stabilization are reviewed. Finally, constructive learning techniques for DTRNN and for feedforward neural networks that can be extended to DTRNN are reviewed. It is seen that, successful modelling of nonlinear dynamic systems with DTRNN requires critical consideration of the network architecture, the training algorithm and the stability of the network and its training.

3 BLOCK DIAGONAL RECURRENT NEURAL NETWORKS

3.1 INTRODUCTION

In this chapter, a sparse but structured architecture in which the feedback connections are restricted to between pairs of state variables is introduced. This network has two layers, the first of which is a DTRNN feedback layer made up of a block-diagonal structure and the second an interconnecting output layer that combines the state variables of the feedback layer to generate the network output. This architecture is referred to as the block-diagonal recurrent neural network (BDRNN) [Sivakumar et. al, 1995]. The BDRNN structure is then extended to include a multi-layer feedforward network in order to model any complex nonlinear mapping which exists directly between the external inputs and the outputs of the system. The resultant structure, referred to as the feedforward block-diagonal recurrent neural network (FF-BDRNN), provides a framework to model both static and dynamic characteristics in a unified fashion. The importance of such an approach to practical applications has been widely recognized [Narendra and Parthasarathy 1990 and 1991, Gori et al, Day and Davenport]. For example, Gori et al consider a multi-layer feedforward architecture with some self-recurrent neurons to model speech signals such as the voiced speech consonants "b" and "d". Reported in [Waibel et al, Unnikrishnan et al], are multi-layer feedforward networks with time-delays to recognize voiced stop consonants and speaker independent connected digits respectively. Reported in [Tsoi and Back] are studies that show that the performance of locally recurrent globally feedforward (LRGF) networks with state feedback, in the one-step prediction of the utterance "one", is superior to that of a fully recurrent single layer network.

3.2 MOTIVATION

The motivation for the choice of the BDRNN architecture is threefold:
first, a block-diagonal feedback weight matrix is capable of modelling plants with complex eigenvalues in a linear sense; second, this structure is conducive to devising a training algorithm in which the computation of the gradient is "local" in both space and time; third, the use of the sparser but structured feedback matrix eases the problem of monitoring and maintaining network stability at each weight update [Kung]. These motivations are discussed in detail below.

3.2.1 Dynamic modelling

From linear time-invariant systems theory it is known that an n-th order dynamics can be represented by a combination of several first or second order dynamics. This stems from the fact that the characteristic polynomial with real coefficients of such a system can be expressed as the product of a number of first degree polynomials with real roots and second degree polynomials with complex conjugate roots of the form:

$$a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0 = \prod_{i=1}^{n_1} (x^2 + b_i x + c_i) \prod_{j=1}^{n_2} (x + d_j) \qquad (1)$$

where n_1 and n_2 are integers related to n by $(2n_1 + n_2) = n$. Here, the a_i's are the real coefficients of the characteristic polynomial, b_i's and c_i's are the coefficients of the second-order polynomial and d_i's are the coefficients of the first-order polynomial. The roots of the first-order polynomials are all real and are of the form $-d_j$, while the roots of the second order polynomials are complex-conjugate pairs of the form $(\alpha_i \pm j\beta_i)$.

Let W_i be a scaled orthogonal 2 x 2 weight matrix with elements as:

$$W_i = \begin{bmatrix} \alpha_i & \beta_i \\ -\beta_i & \alpha_i \end{bmatrix} \tag{2}$$

that models one of these first or second order dynamics. Then the coefficients of the second order polynomial in (1) are related to the matrix elements of W_i as follows:

$$b_i = -2\alpha_i , \qquad c_i = \alpha_i^2 + \beta_i^2 \tag{3}$$

The second order polynomial modelled by (2) has a damped oscillatory dynamic and is especially useful in modelling the oscillating modes of a dynamic process. On the other hand, W_i can be used to model the first order polynomials of (1) by setting $\beta_i = 0$. Although W_i models a "double root" in this case, one of these roots can be deselected by making it unobservable using the output matrix. Note that the coefficients of the first-order polynomial are related to the matrix element α_i by $-d_j = \alpha_i$.

Alternately, let $W_{fi} = T^{-1}W_iT$ be a transformation of the scaled orthogonal weight matrix W_i, such that T is a 2 x 2 nonsingular matrix, with elements as:

$$W_{fi} = \begin{bmatrix} \gamma_{11i} & \gamma_{12i} \\ \gamma_{21i} & \gamma_{22i} \end{bmatrix} \tag{4}$$

that models one of these first or second order dynamics. Then the coefficients of the second order polynomial in (3) are related to the matrix elements of W_{fi} as follows:

$$c_i = \gamma_{11i}\gamma_{22i} - \gamma_{12i}\gamma_{21i} , \qquad b_i = -(\gamma_{11i}+\gamma_{22i}) \tag{5}$$

The second order polynomial modelled by (4) also has a damped oscillatory dynamic. On the other hand, if W_{fi} is to be used to model the first order polynomials of (1), then, the off-diagonal elements γ_{12i} and γ_{21i} must be set equal to zero and the coefficients of the first-order polynomial are related to the diagonal matrix elements γ_{11i} and γ_{22i} by $-d_j = \gamma_{11i} = \gamma_{22i}$.

Extending this concept to a nonlinear process, if its dynamics can be "decoupled" into several lower order dynamics, then it may be feasible to model it with a DTRNN having a block diagonal feedback weight matrix with blocks of size 2 with each submatrix modelling a low order dynamic of the process. It should also be noted that a purely diagonal feedback weight matrix popular in literature, like the ones considered in [Tsoi and Back, Gori et al, Ku and Lee], has only real eigenvalues and hence cannot model the oscillatory dynamics of a second order system with complex conjugate roots. Thus, LRGF networks with self-recurrent connections may not effectively model the oscillating modes of a dynamic process.

3.2.2 Training

As seen in Chapter 2, trajectory training of DTRNN is generally accomplished by using either real time recurrent learning (RTRL) [Williams and Zipser 1989] or backpropagation through time (BPTT) [Werbos] algorithms or variations thereof [Williams and Peng 1990, Schmidhuber, Sun et al], (see [Pearlmutter 1995] for a survey). Both the RTRL and BPTT algorithms compute the exact error gradient [Beaufays and Wan]. RTRL is computationally expensive due to the spatially nonlocal nature of its error gradient computation. For RTRL, from equation (7) of Chapter 2, it is seen that the formulation of the gradients which updates the DTRNN weights during training requires recursive computation of the rate of change of state variables with respect to weights $\partial x_s(k)/\partial w_{ij}$, and requires that the partial derivatives of the state variables $x_s(k-1)$ with respect to the elements of $w_{ij}\,\partial x_s(k)/\partial w_{ij}$ at the $k-1$-th instant be stored. This makes the *exact* gradient computation spatially *nonlocal* and is computationally expensive. In BPTT, from equations (12)-(15) of Chapter 2, it is seen that the computation of the error gradient in the backward pass is spatially *local* in nature, since this computation at any layer k requires information only from other nodes to which it connects. The *exact* error gradient computation is done in the backward pass and requires that the state, error and input vectors be stored at each time instant in the forward pass and hence, the storage

requirement for BPTT increases with the length of the training pattern. This makes the error gradient computation in conventional BPTT *nonlocal in time*. The gradient computation can be made local in time, by eliminating the storage requirement for the state vectors, by recalculating the states vectors in the backward pass. This, however, requires that the feedback weight matrix be inverted after each weight update. Also, this technique of recalculating the state vectors is susceptible to numerical instability problems [Pearlmutter 1995]. The block diagonal structure of the BDRNN lends itself to ensuring invertibility through local computations as the feedback weight matrix is easily inverted by simple manipulation and scaling of its elements. Also the reduced interaction between the state variables of the BDRNN helps in reducing the numerical instability problem. *Thus, the BDRNN structure is conducive to devising a trajectory training algorithm in which the computation of the gradient is "local" in both space and time, provided that the numerical stability of the algorithm is ensured suitably*. In Chapter 4 of this thesis, a modified online BPTT algorithm that recalculates the state vectors and computes the exact gradient is proposed. The numerical stability of the algorithm is ensured by the following: first, state vectors at evenly-spaced intermediate time intervals are stored in the forward pass; second, these intermediate stored values are used as initial values to recompute the state vectors during the backward pass while simultaneously monitoring for signs of numerical instability; when numerical instability is detected, the state vector for that time instant/layer is computed by performing a forward pass using the nearest stored intermediate state vector and the process of recomputing the state vectors is continued. In the proposed algorithm the gradient computation is local in both space and time in the time interval from one stored intermediate value to the next. Implementing the proposed algorithm results in reduced storage compared to conventional BPTT by trading off some of the later's computational advantage.

3.2.3 Stability

The feedback structure of the DTRNN necessarily raises the question of stability, both of the network and during its training. While, for a stable network, the stability of training is well understood and can be ensured by suitably choosing the learning rate [Almeida], there are several unresolved issues in monitoring and maintaining the stability of the network itself at each weight update.

Defined in [Kelly] are the conditions that must be satisfied for nonlinear dynamics to converge, while [Jin et al 1994 and 1996] define conditions based on the norm of the feedback weight matrix that must be satisfied for the absolute and global stability of a DTRNN. These conditions were summarised in equations (27) and (37) of Chapter 2. However there is no simple mechanism for finding the weights required to implement contractive nonlinear dynamics or find absolute and globally stable DTRNN networks. A major difficulty in ensuring contractive dynamics or in monitoring the stability of a fully recurrent DTRNN while training relates to the computation of a suitably selected norm of the feedback matrix at each weight update during training. Satisfying the 1-norm requires that the column sum of the magnitude of the weights in each column be less than $2/a$, similarly satisfying the ∞-norm requires that the row sum of the magnitude of the weights in each row be less than $2/a$. Hence, satisfying these conditions places severe restrictions on the weight placement, in particular, for large order network. Satisfying the 2-norm at each weight update requires computing the largest eigenvalue of the product of the feedback weight matrix and its transpose. This condition is deceptively simple, and in the case of a fully recurrent DTRNN is a difficult and computationally expensive problem. However, if the feedback weight matrix is structured to be sparser, such that it is possible to monitor these eigenvalues by simple manipulations of the weights, then the stability of such DTRNN can be addressed in a simpler framework. For example, consider a DTRNN with only self recurrence [Gori et al]; in this case the feedback weight matrix is diagonal and the global stability of such a network can be ensured by restraining the

magnitude of each diagonal weight to less than *2/a*. Now let us consider the case of the BDRNN. In this structured architecture, the feedback connections are restricted to between pairs of state variables. Hence the weight matrix consists of 2 x 2 subsections as shown in (1). Hence, the stability of the BDRNN can be ensured by ensuring the stability each individual 2 x 2 subsection of the BDRNN. Global and local stability conditions pertaining to a particular 2 x 2 subsection can be found using local computations by simple manipulation of the weights in that subsection and do not depend on the weights in the other subsections. Since, the stability of the BDRNN architecture has been decomposed it can be addressed in a simpler framework.

In Chapter 5 of this thesis, the block diagonal structure of the BDRNN is exploited to find conditions that ensure network stability at each weight update during training. This problem is addressed in two steps: first, by devising a cost function that includes the desired stability margin as one of its components; next, by ingraining the stability margin component as a multi-layer feedforward neural structure which augments the BDRNN structure.

3.3 BLOCK-DIAGONAL RECURRENT NEURAL NETWORK STRUCTURES

3.3.1 Block-diagonal recurrent neural network

The block diagonal recurrent neural network (BDRNN) is now formally introduced. It consists of a DTRNN with a block-diagonal state feedback matrix with 2 x 2 sub matrices as given in (6), which is referred to as block diagonal recurrent neural network.

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & . & . & 0 & 0 \\ w_{2,1} & w_{2,2} & . & . & 0 & 0 \\ 0 & 0 & . & . & 0 & 0 \\ 0 & 0 & . & . & 0 & 0 \\ 0 & 0 & . & . & w_{n-1,n-1} & w_{n-1,n} \\ 0 & 0 & . & . & w_{n,n-1} & w_{n,n} \end{bmatrix} \tag{6}$$

An N state variable, M input, N_o output BDRNN is shown in Figure 3.1. The system equation for the BDRNN is given by:

$$x_i(k+1) = f_a\left(\sum_{j=v}^{v+1} w_{i,j}x_j(k) + \sum_{j=1}^{M} b_{i,j}u_j(k)\right) \qquad i=1,..,N$$
$$\text{where} \quad v = i, \quad \text{if } i \text{ is odd}, \quad v = i-1, \quad \text{if } i \text{ is even.} \tag{7}$$

where $x_i(k)$ and $u_j(k)$ are the i-th state variable and j-th input elements respectively, at instant k, $w_{i,j}$'s are the state-feedback weight parameters. $b_{i,j}$'s are the external input weight parameters and $f_a(.)$ is a symmetric sigmoidal squashing function defined as:

$$f_a(x) = \frac{1-e^{-ax}}{1+e^{-ax}}, \quad a \geq 0 \tag{8}$$

with $\alpha = a$. The output equation for the BDRNN of Figure 3.1 is given by:

$$y_h(k) = f_b\left(\sum_{j=1}^{N} c_{h,j}x_j(k)\right), \qquad h=1,...,N_o \tag{9}$$

where $y_h(k)$ is the h-th output element at instant k. $c_{i,j}$'s are the weights connecting the decoupled state variables to the N_o output units and $f_b(.)$ is a sigmoidal function as defined in equation (8) with $\alpha = b$. In (7), the process dynamics are modelled by interactions limited to between pairs of state variables. For example, state variables 1 and 2 interact with each other and with no other state variable in the system.

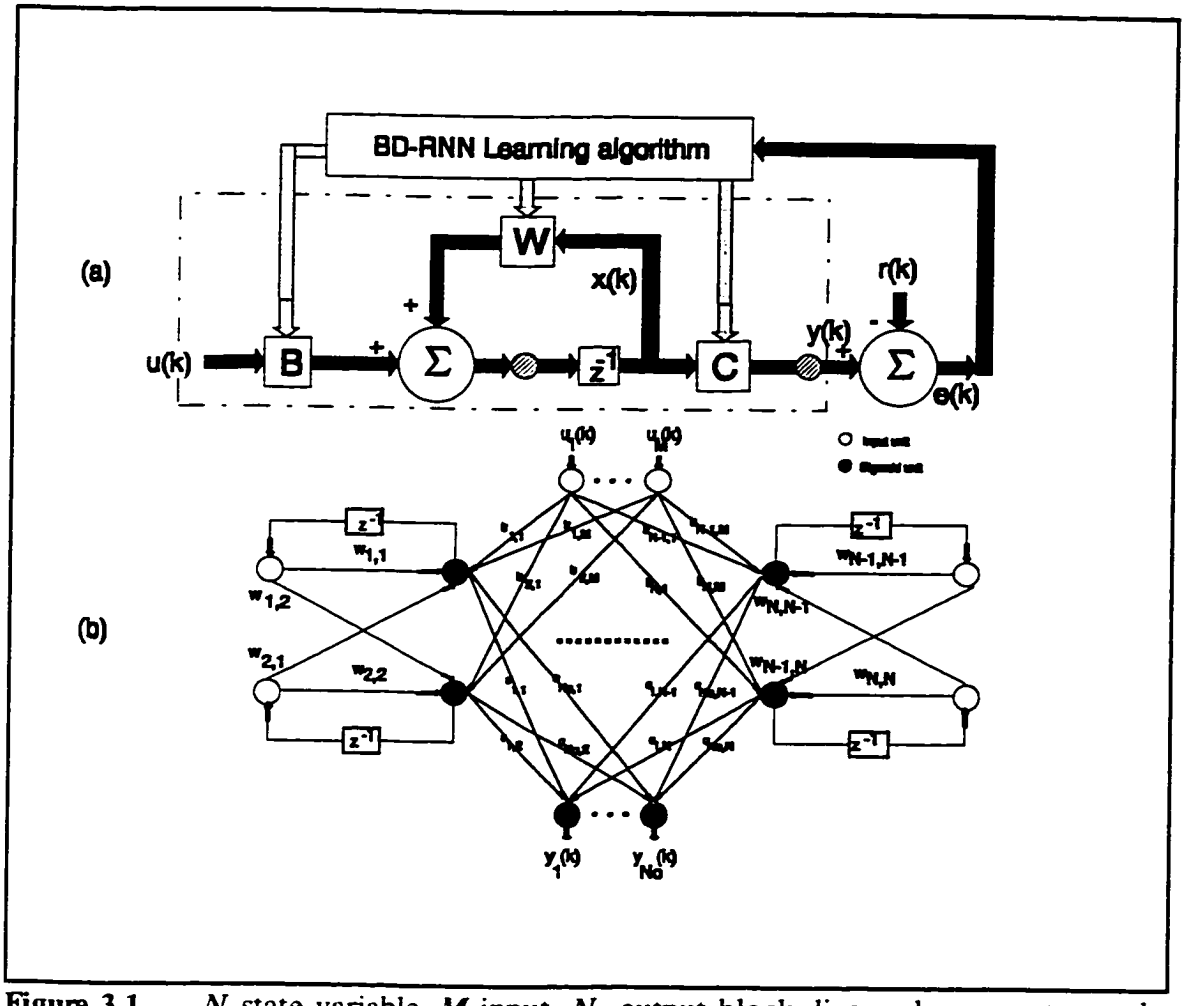


Figure 3.1 N state variable, M input, N_o output block-diagonal recurrent neural network: (a) Block diagram (b) Actual connections in a BDRNN.

The matrix form of (7) and (9) is given by

$$\begin{aligned} \underline{x}(k+1) &= \underline{f}_a(W\underline{x}(k) + B\underline{u}(k)) \\ \underline{y}(k) &= \underline{f}_b(C\underline{x}(k)) \end{aligned} \tag{10}$$

where, $\underline{x}(k+1) = \{x_i(k+1)\}^T$, $\underline{u}(k) = \{u_i(k)\}^T$, $\underline{f}_a(\underline{x}(k)) = \{f_a(x_i(k)\}^T$, $B = \{b_{i,j}\}$, $\underline{y}(k) = \{y_i(k)\}^T$, $C = \{c_{i,j}\}$. An alternate form of (10) can be given by substituting:

$$\underline{x}''(k+1) = W\underline{x}(k) + B\underline{u}(k) \tag{11}$$

the state equation of the BDRNN of (10) can be rewritten in the form:

$$\underline{x}''(k+1) = W \underline{f}_a(\underline{x}''(k)) + B\underline{u}(k) \qquad (12)$$

A plausibility argument for the potential capability of the BDRNN architecture (12) to model nonlinear dynamics can be given by considering how it relates to a fully recurrent DTRNN architecture given by (1) of Chapter 2 (in which the outputs are a subset of the state variables) which is given by

$$x_i^f(k+1) = \sum_{j=1}^{N} w_{ij}^f f_a(x_j^f(k)) + \sum_{j=1}^{M} b_{ij}^f u_j(k) \qquad i=1,..,N \qquad (13)$$

The matrix form of (13) is given by

$$\underline{x}^f(k+1) = W^f \underline{f}_a(\underline{x}^f(k)) + B^f \underline{u}(k) \qquad (14)$$

where, $\underline{x}^f(k) = \{x^f_{i,j}(k)\}^T$, $\underline{u}(k) = \{u_{i,j}(k)\}^T$ and $W^f = \{w^f_{i,j}\}$, $B^f = \{b^f_{i,j}\}$. Let us assume that W^f is invertible and has distinct eigenvalues. From linear systems theory it is well known that, for this case, there exists a transformation matrix T such that

$$W^f = T^{-1} \Lambda \, T \qquad (15)$$

where Λ is also a block diagonal matrix with the same form as W in (6). Substituting $z(k) = Tx^f(k)$ in (14) we obtain

$$\underline{z}(k+1) = \Lambda \, T \underline{f}_a(T^{-1}\underline{z}(k)) + TB^f \underline{u}(k) \qquad (16)$$

Comparing equations (16) and (12), the following observations are made:

(i) (16) and (12) have the same form with $T\underline{f}_a(T^{-1}\underline{z}(k))$ in (16) replaced by $\underline{f}_a(\underline{x}(k))$ in (12).

(ii) If a nonlinear process can be modelled by (13) with the characteristics that $T\underline{f}_a(T^{-1}\underline{z}(k))$ in (16) is "weakly" nonlinear, which implies that the nonlinear system

being modelled is "decomposable" with its decomposed dynamics related to the eigenvalues of Λ, then $T\underline{f}_a(T^{-1}\underline{z}(k))$ can be approximated as $\underline{f}_a(\underline{x}(k))$ with reasonable accuracy; i.e., the above nonlinear process with decomposable dynamics can be effectively modelled by the BDRNN given by (7) and (9).

(iii) If a nonlinear process that is modelled by (13) with the characteristics that $T\underline{f}_a(T^{-1}\underline{z}(k))$ has a "strong" nonlinearity, which implies that dynamics of the nonlinear system that is modelled has only a partial relevance to Λ, the approximation of $T\underline{f}_a(T^{-1}\underline{z}(k))$ as $\underline{f}_a(\underline{x}(k))$ is invalid and such a nonlinear system cannot be effectively modelled by the BDRNN.

Several nonlinear systems considered in literature fall within the category (ii). These include nonlinear limit cycles such as the "figure-0" and "figure-8" dynamics considered by [Pearlmutter 1989 and 1995], chaotic systems with recognizable periodicity and oscillatory modes such as the chaotic signal produced by integrating the Mackey-Glass delay-differential equation [Mackey and Glass]. This signal is quasi-periodic with a characteristic time of $\approx$ 100, and provides a useful benchmark for testing predictive techniques [Sanger, Day and Davenport]. Examples of practical nonlinear temporal signals, which fall within the category (ii), include any speech signal with rich formant content such as the voiced stop consonants ("b"-"d" considered by [Gori et al]) and digits zero-nine of speech (digit "one" considered by [Tsoi and Back]).

3.3.2 Feedforward block diagonal recurrent neural network (FF-BDRNN)

Dynamic nonlinear processes are generally modelled by recurrent networks [Nerrand et al], while static nonlinear processes are generally modelled by multi-layer feedforward networks [Hush and Horne]. However, there exist many practical systems, such as speech, that possess both static and dynamic nonlinear characteristics. In order to model such systems in a unified fashion, it is necessary to incorporate multi-layer feedforward

structures in a recurrent framework. Although, several forms of feedforward structures are suitable for this purpose, the structures considered here are restricted to be multi-layered since these have proved extremely successful in pattern recognition problems [Waibel et al, Unnikrishnan et al, Hush and Horne]. Therefore, the BDRNN structure is extended to include a multi-layer feedforward network in order to model any complex nonlinear mapping between the external inputs and outputs of the system. The resultant structure is referred to as the feedforward block-diagonal recurrent neural network (FF-BDRNN). Illustrated in Figure 3.2 is the block diagram of a N state variable, M input, N_o output, L feedforward layer FF-BDRNN. The output equation for this FF-BDRNN is given by:

$$y_h(k) = f_b\left(\sum_{j=1}^{N} c_{hj} x_j(k) + \sum_{j=1}^{N_{L-1}+1} d_{hj}^{L-1} u_j^{L-1}(k)\right), \quad h=1,..,N_o \tag{17}$$

where

$$u_i^l(k) = f_c\left(\sum_{j=1}^{N_{l-1}+1} d_{ij}^{l-1} u_j^{l-1}(k)\right), \quad l=2,..,L-1;\ i=1,..,N_l \tag{18}$$

where the second summation term in the right hand side of the equation (17) represents the feedforward connection. In (17) and (18), d_{ij}^{l-1} is the feedforward interconnection weight from the j-th unit of the l-1^{th} layer to the i-th unit in the l-th layer; $u_j^l(k)$ denotes output activation of the j-th unit of the l^{th} layer at instant k; N_l is the number of units in the l^{th} feedforward layer of the network. $f_c(.)$ is a sigmoidal squashing function defined in (8) with $\alpha = c$. The bias weights are accounted for by defining the N_l+1 unit of each layer l to be the bias unit. For notational convenience let the first feedforward layer hold the input vector, i.e. $u_j^1(k) = u_j(k)$. Also the L-th feedforward layer is the output layer of the composite FF-BDRNN network, i.e. $u_h^L(k) = y_h(k)$, the output activation of the h-th unit of the L-th feedforward layer equals the h-th component of the output vector. The rest of the symbols in (17) and (18) are as defined for (9).

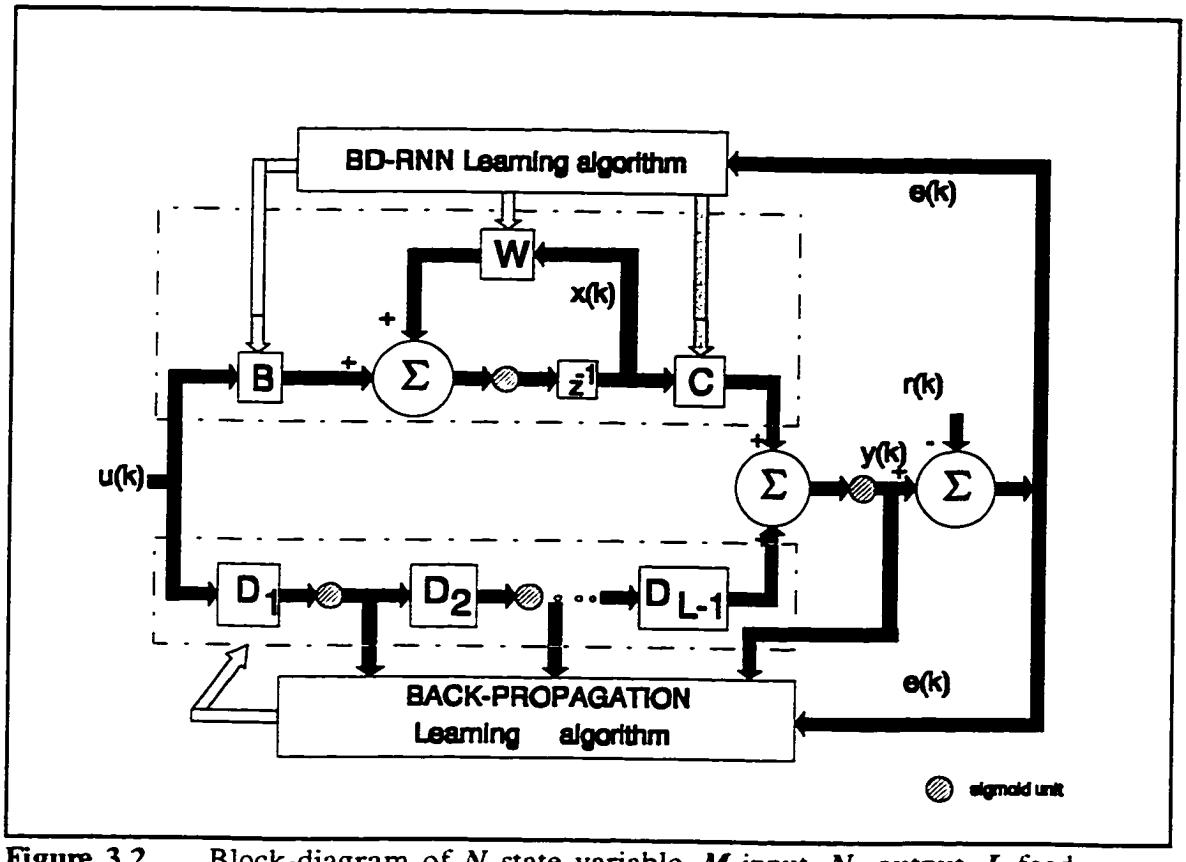


Figure 3.2 Block-diagram of N state variable, M input, N_o output, L feed-forward layer FF-BDRNN

3.3.3 Training of FF-BDRNN

The training of the FF-BDRNN described by (7), (17) and (18) is based on the minimization of a mean-squared output error function. In order to ensure that the feedback weight matrix is stable, this training should be performed with the constraint that a suitably defined norm of the updated W is within a bound determined by the slope of the symmetric sigmoidal limiter (8). This problem can be restated by defining an augmented error function E_t, given by:

$$E_t = J_t + P_t , \qquad t = 1, .. ,T \tag{19}$$

where J_t is the mean squared output error and P_t is a penalty term that is a function of the stability constraint for the t^{th} sequence of the T total number of training sequences. J_t is given by:

$$J_t = \frac{1}{K_t}\sum_{k=1}^{K_t}\frac{1}{2}\sum_{j=1}^{N_o}(e_j(k))^2 \qquad (20)$$
$$\text{where } e_j(k) = r_j(k) - y_j(k)$$

where K_t is the length of the t-th training sequence, $r_j(k)$ is the desired output for the j-th output unit at instant k.

The penalty term P_t is a function of the elements of the feedback weight matrix W and, is generally chosen to be smooth, non-linear and continuously differentiable with respect to the elements of W. The algorithm for training the FF-BDRNN is developed in Chapter 4. The specific choice of P_t is addressed in Chapter 5.

3.3.4 Special block-diagonal structures

Two cases of block diagonal structure for the FF-BDRNN feedback matrix which differ from each other in terms of complexity and the degree of freedom allowed in the values that their elements can assume are considered. In the first case, each 2 x 2 submatrix of the block-diagonal feedback matrix is a scaled orthogonal matrix of the form (2) (in which the diagonal elements are identical and the off-diagonal elements are equal in magnitude but opposite in sign). This scaled orthogonal submatrix models a second order dynamic system with a complex-conjugate eigenvalue pair using only two distinct elements. Also, as will be shown later in Section 5.2.1, the global stability condition for this structure is the same as the local stability condition and hence, satisfying the former does not impose additional restrictions on the weights assignment.

In the second case, each element of the 2 x 2 submatrices can assume any value of the form (4). Each of such freeform submatrices can model a second order dynamic system with real or complex-conjugate eigenvalue pairs using four distinct elements. Also, as will be discussed in Section 5.2.2, the global stability condition for the freeform structure is different from the local stability condition and imposes more restrictions on the weights assignment.

While the FF-BDRNN with scaled orthogonal submatrices has fewer elements and is hence less complex, the FF-BDRNN with freeform submatrices has more elements and hence a larger degree of freedom.

The local and global stability conditions for the two structures considered will be derived in sections 5.2.1 and 5.2.2 of Chapter 5.

3.4 CONCLUSION

In this chapter, a DTRNN with a block diagonal feedback weight matrix is introduced. The BDRNN is shown to be advantageous in terms of its ability to model the oscillating modes of a nonlinear dynamic process. The block-diagonal structure is conducive to devising a trajectory learning algorithm in which the gradient computation is local. The stability of the BDRNN is ensured by ensuring the stability of individual subsections of the BDRNN. As static and dynamic characteristics are best modelled by feedforward and recurrent connections respectively, a feedforward block-diagonal recurrent neural network (FF-BDRNN) framework is proposed. A learning algorithm in which the computation of the gradient is exact and local and which requires significantly reduced storage requirement, but with marginally increased computations compared to conventional BPTT will be presented in the next chapter.

4 A RECOMPUTED STATE VARIABLE MODIFIED BPTT ALGORITHM FOR BDRNN

4.1 INTRODUCTION

As seen in Chapter 2, trajectory training of DTRNN is generally accomplished by using either real time recurrent learning (RTRL) [Williams and Zipser 1989] or backpropagation through time (BPTT) [Werbos] algorithms or variations thereof [Williams and Peng 1990, Schmidhuber, Sun et al, Pearlmutter 1995]. To summarize, both RTRL and BPTT are online techniques that compute the exact error gradient [Beaufays and Wan]. While RTRL requires spatially nonlocal error gradient computation which is computationally expensive, BPTT calculates the error gradients using spatially local computations with a storage requirement that increases with the length of the training pattern. However, the exact gradient computation in BPTT is nonlocal in time and requires storing the state variables, the inputs and errors at all the time steps in an epoch. BPTT-t_0 step and BPTT-William-Peng are online variations of BPTT that approximate the error gradient, have reduced storage requirements and are computationally more complex when compared with conventional BPTT. The BPTT-RTRL hybrid algorithm [Schmidhuber] computes the exact gradient and has an average computational requirement of one order less than that required in RTRL and the same storage requirements as RTRL. Thus BPTT is the least expensive of the trajectory training algorithms because of the spatially local gradient computations. However, the gradient computations are nonlocal in time and hence, it has excessive storage requirements especially for long storage patterns.

The gradient computation in BPTT can be made local in time, by eliminating the storage requirement for the state vectors by recalculating the state vectors in the backward pass. This, however, requires that the feedback weight matrix be inverted after each weight update. Since ensuring invertibility and inverting of a fully recurrent feedback weight matrix are complex problems and is computationally expensive, this technique has not

been considered a viable alternative in RNN research circles. The block diagonal structure of the BDRNN lends itself to ensuring invertibility as the feedback weight matrix is easily inverted by simple manipulation and scaling of its elements. Also, this technique of recalculating the state vectors is susceptible to numerical instability problems [Pearlmutter 1995] especially, in fully recurrent DTRNN. However, the reduced interaction between the state variables of the BDRNN helps in reducing the numerical instability problem.

The training of the BDRNN described by (7), of Chapter 3, is based on the minimization of a mean-squared output error function. In order to ensure that the feedback weight matrix is stable, this training should be performed with the constraint that a suitably defined norm of the updated W is within a bound determined by the slope of the symmetric sigmoidal limiter (8). This problem can be restated by defining an augmented error function E_t, given by:

$$E_t = J_t + P_t , \qquad t = 1, .. ,T \tag{1}$$

where J_t is the mean squared output error and P_t is a penalty term that is a function of the stability constraint for the t-th sequence of the T total number of training sequences. J_t is given by:

$$J_t = \frac{1}{K_p}\sum_{k=1}^{K_t}\sum_{j=1}^{N_o} \epsilon_j(k) \quad \textit{where } \epsilon_j(k) = \frac{1}{2}\left(r_j(k)-y_j(k)\right)^2 \tag{2}$$

where K_p is the length of the p-th training sequence, $r_j(k)$ is the desired output for the j-th output unit at instant k. The penalty term P_t is a function of the elements of the feedback weight matrix W and, is generally chosen to be smooth and continuously differentiable with respect to the elements of W. The specific choice of P_t is addressed in Chapter 5. The algorithm for training the BDRNN is developed below, assuming P_t is given.

4.2 TRAINING BDRNN WITH A RECOMPUTED STATE VARIABLE MODIFIED BPTT ALGORITHM

The proposed modified BPTT learning algorithm [Sivakumar et. al. 1996a] uses the conventional BPTT technique [Werbos] to update the block-diagonal state feed-back weight matrix W after K_p-steps of the training sequence have been presented, according to:

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) - \mu_1 \frac{\Delta w_{ij}(t)}{K_p} - \mu_2 \frac{\partial P_t}{\partial w_{ij}(t)} \\ \textit{with} \quad \Delta w_{ij}(t) &= \sum_{k=1}^{K_p} \frac{\partial J_p(k,t)}{\partial w_{ij}(t)} \\ \textit{for} \quad j &= i \,..\, i+1, \qquad \textit{if } i \textit{ is odd,} \\ j &= i-1 \,..\, i, \qquad \textit{if } i \textit{ is even.} \end{aligned} \tag{3}$$

where t is the iteration index of the weights. Δw_{ij} is the accumulation of the instantaneous error gradient over the K_p steps of the training pattern and μ_1 and μ_2 are learning rate parameters. For clarity t will be omitted in the following derivations. $J_p(k)$ is the total squared error over all the N_o output units for the p-th training pattern and is given by:

$$J_p(k) = \frac{1}{2} \sum_{h=1}^{N_o} \left(r_h(k) - y_h(k)\right)^2 \tag{4}$$

The algorithm is implemented by using the chain rule:

$$\Delta w_{ij}(t) = \frac{\partial^+ J_p}{\partial w_{ij}} = \sum_{k=1}^{K_p} \frac{\partial^+ J_p}{\partial x_i(k)} \frac{\partial x_i(k)}{\partial w_{ij}} \tag{5}$$

where,

$$\begin{aligned} \frac{\partial x_i(k)}{\partial w_{ij}} &= f_a'(s_i(k)) x_j(k-1); \qquad \frac{\partial^+ J_p}{\partial x_i(k)} = e_i(k) \\ \textit{with} \quad s_i(k) &= \sum_{j=v}^{v+1} w_{ij} x_j(k-1) + \sum_{j=1}^{M} b_{ij} u_j(k-1) \end{aligned} \tag{6}$$

In conventional BPTT, the state vector $\{x_j(k-1)\}$ used in (6) and the error vector $e_h(k)$ used in (7), are computed and stored in the forward pass, using (7) and (20) respectively of Chapter 3. For the BDRNN, $e_i(k)$ is computed recursively in the backward pass:

$$e_i(k) = -\sum_{h=1}^{N_o} e_h(k) f_h'\left(\sum_{j=1}^{N} c_{h,j} x_j(k)\right) c_{h,i} + \sum_{j'=v}^{v+1} w_{i,j}^T f_a'(s_j(k+1)) e_j(k+1)$$

where

$$s_j(k+1) = \sum_{i'=v}^{v+1} w_{j',j} x_i(k) + \sum_{i'=1}^{M} b_{j',j} u_i(k) \qquad j' = 1,\ldots,N; \quad \begin{array}{ll} v = j' & \textit{if } j' \textit{ is odd} \\ v = j'-1 & \textit{if } j' \textit{ is even} \end{array} \tag{7}$$

with the initial value $e_i(K_p)$ given by:

$$e_i(K_p) = -\sum_{h=1}^{N_o} e_h(K_p) c_{h,i} \tag{8}$$

Thus, in conventional BPTT the state, input and error vector requires $O(K_p(N+M+N_o))$ storage. In the proposed modified BPTT algorithm, the state variables $\{x_j(k-1)\}$ are recomputed in the backward pass from the previously recomputed value of the state vector $\{x_i(k)\}$. This recomputation process eliminates the requirement to store the state vector. The recomputation of the state variables is done recursively according to:

$$x_i^r(k) = \sum_{j=v}^{v+1} w_{i,j}^{\epsilon}\left[f_a^{-1}(x_i^r(k+1)) - \sum_{j=1}^{M} b_{i,j} u_j(k)\right], \quad \begin{array}{ll} v = i & \textit{if } i \textit{ is odd} \\ v = i-1 & \textit{if } i \textit{ is even} \end{array} \tag{9}$$

where $x_i^r(k)$ is the recomputed value of the i-th state variable at instant k, and W^{ϵ} is the inverse of the weight matrix W.

From (9) it is seen that the recomputation of the state variables requires the inversion of the sigmoidal function f_a and the inverse of W which presupposes that W is invertible. The task of ensuring the invertibility of W will be addressed in Chapter 5. The block-diagonal structure of W facilitates the computation of the inverse matrix W^{ϵ} by simple manipulation and scaling of the elements of W and is given by:

$$w^{\epsilon}_{n-1,n-1} = \frac{w_{n,n}}{w^{\delta}}, \qquad w^{\epsilon}_{n-1,n} = -\frac{w_{n-1,n}}{w^{\delta}},$$
$$w^{\epsilon}_{n,n-1} = -\frac{w_{n,n-1}}{w^{\delta}}, \qquad w^{\epsilon}_{n,n} = \frac{w_{n-1,n-1}}{w^{\delta}}, \tag{10}$$
$$\textit{where} \quad w^{\delta} = w_{n,n}w_{n-1,n-1} - w_{n-1,n}w_{n,n-1} \neq 0 \qquad n = 2,4,\ldots,N$$

It can be seen from (10) that the inversion of the block-diagonal W can be accomplished by inverting each 2 x 2 submatrix of W. Thus, inverting the block-diagonal W is very simple compared to inverting a fully recurrent DTRNN (which can be accomplished by the use of a fairly complicated matrix inversion algorithm).

The recomputation technique helps to significantly reduce the storage requirement but occasionally results in numerical instability problems, especially for long training patterns, due to the recursive nature of the recomputations. The main reason for the occurrence of numerical instability is a result of the amplifying effect of W^{ϵ} on a small numerical error introduced in any step of the iteration. This is especially true, since, maintaining W stable would necessarily make W^{ϵ} unstable. Another way to look at this is that the weights are updated such that the computation of the state variables in the forward pass is contractive or convergent, then the recomputation of the state variables in the backward pass is expansive or divergent, since the recomputation in the backward pass involves the product $W^{\epsilon}f_a^{-1}(.)$ which is the inverse mapping of the product $f_aW(.)$ that is used in computing the state variables in the forward pass. The dynamics of the numerical stability error is the same as the dynamics of recomputation of the state variable, which is divergent and hence a small error introduced at any time step increases due to the recursive nature of the state recomputations. In the case of the BDRNN, due to the block-diagonal structure of the feedback weight matrix, the error in recomputing a state variable at a particular time step affects the recomputation of only two state variables at the next time step. In the case of a fully recurrent DTRNN with a full weight matrix, the numerical stability problem is

more pronounced because the error in recomputing a state variable at a particular time step affects the recomputation of all the state variables at the next time step.

Another reason for the occurrence of numerical instability is the amplifying effect of f_a^{-1}. Figure 4.1 shows the inverse of the sigmoidal function given by (8) of Chapter 2. The numerical instability is pronounced especially when the computed value of the state variable in the forward pass is close to saturation i.e., close to ±1.

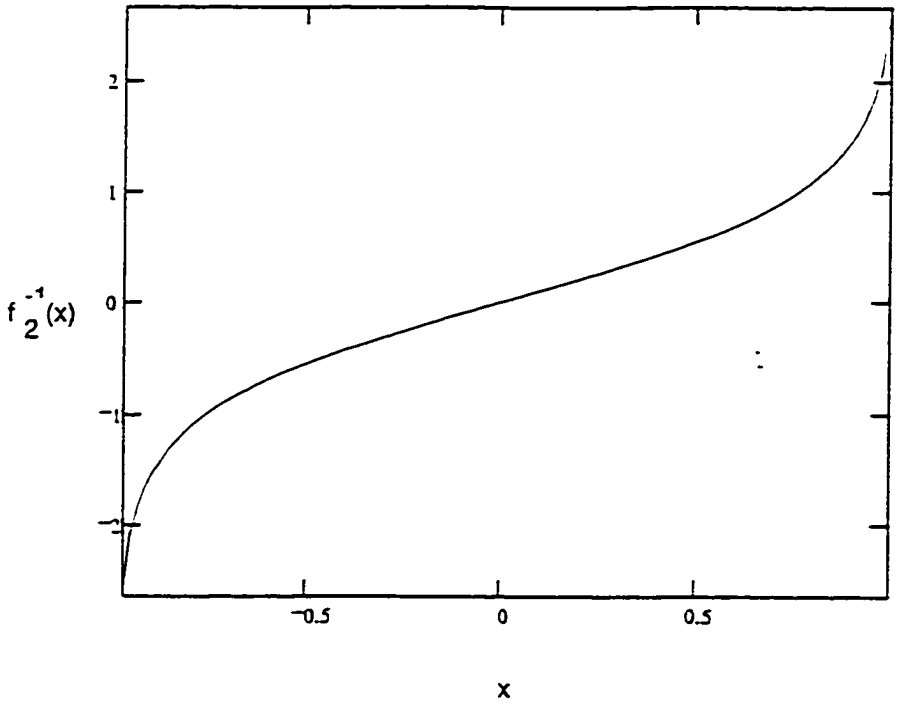


Figure 4.1 Inverse of the sigmoidal function given by (8) of Chapter 2

For practical situations, despite the amplifying effects of $W^{e}f_a^{-1}(.)$ it is still possible to perform several recursive computations without significant loss of accuracy. If numerical stability is degraded during this recursion, a recovery is possible by performing a forward pass computation.

In this thesis, the numerical stability of the recomputed state variable modified BPTT learning algorithm is ensured by the following steps:

(i) recursive state variable recomputation: in the forward pass, intermediate values of the state vector are stored at evenly spaced intervals over the length of the pattern. In the backward pass, these intermediate stored values are used as initial values for the recomputations performed over sublengths of the training pattern.

(ii) monitoring numerical stability: any signs of numerical instability in the backward pass are monitored by a scalar shadow error $e^{s}(k)$. The shadow error is obtained by comparing a scalar shadow output $y^{fs}(k)$ computed in the forward pass with the scalar shadow output $y^{rs}(k)$ computed in the backward pass using the recalculated values of the state vectors. The shadow error is thus a measure of the numerical stability of the recomputations.

(iii) recovery computation: when a numerical instability is encountered at any time k, the state vector $\{x_i(k)\}$ is computed from the nearest intermediate stored state vector by iteratively using the forward pass computation given by:

$$x_i(k+1) = f_a\left(\sum_{j=v}^{v+1} w_{ij}x_j(k) + \sum_{j=1}^{M} b_{ij}u_j(k)\right) \qquad i=1,..,N \tag{11}$$

where the symbols are as defined for (7) of Chapter 3.

The success of the above approach for a given problem depends on how few the number of recovery computations performed in (iii) are in relation to the number of recursive recomputations of the state variables performed in (i).

Now let us consider the weight updates for the input weight matrix B and the output weight matrix C. The learning algorithm updates the N x M input weight matrix B after K_p steps of the training sequence have been presented, according to:

$$b_{ij}(t+1) = b_{ij}(t) - \mu_1 \frac{\Delta b_{ij}(t)}{K_p}, \quad i = 1,...,N; \quad j = 1,...,M$$
$$\text{with} \quad \Delta b_{ij}(t) = \sum_{k=1}^{K_p} \frac{\partial J_p(k,t)}{\partial b_{ij}(t)} \tag{12}$$

where t is the iteration index of the weights. Δb_{ij} is the accumulation of the instantaneous error gradient over the K_p steps of the training pattern. For clarity t will be omitted in the following derivations. The algorithm is implemented by using the chain rule:

$$\frac{\partial^+ J_p}{\partial b_{ij}(t)} = \sum_{k=1}^{K_p} \frac{\partial^+ J_p}{\partial x_i(k)} \frac{\partial x_i(k)}{\partial b_{ij}} \tag{13}$$

where,

$$\frac{\partial x_i(k)}{\partial w_{ij}(k)} = f_a'(s_i(k)) u_j(k-1); \qquad \frac{\partial J_p(k)}{\partial x_i(k)} = \varepsilon_i(k)$$
$$\text{with} \quad s_i(k) = \sum_{j=v}^{v+1} w_{ij} x_j(k-1) + \sum_{j=1}^{M} b_{ij} u_j(k-1) \tag{14}$$

The learning algorithm updates the N_o x N output weight matrix C after K_p-steps of the training sequence have been presented, according to:

$$c_{ij}(t+1) = c_{ij}(t) - \mu_1 \frac{\Delta c_{ij}(t)}{K_p}, \qquad i = 1 \,..\, N_o, \quad j = 1 \,..\, N$$
$$\text{with} \quad \Delta c_{ij}(t) = \sum_{k=1}^{K_p} \frac{\partial J_p(k,t)}{\partial c_{ij}(t)} \tag{15}$$

where t is the iteration index of the weights. Δc_{ij} is the accumulation of the instantaneous error gradient over the K_p steps of the training pattern. For clarity t will be omitted in the following derivations. The algorithm is implemented by using the chain rule:

$$\frac{\partial^+ J_p}{\partial c_{hj}(t)} = \sum_{k=1}^{K_p} \frac{\partial^+ J_p}{\partial y_h(k)} \frac{\partial y_h(k)}{\partial c_{hj}} \tag{16}$$

where,

$$\frac{\partial y_h(k)}{\partial c_{hj}} = f_b'(s_h(k)) x_j(k), \qquad \frac{\partial^+ J_p}{\partial y_h(k)} = -e_h(k)$$
$$\text{with} \quad s_h(k) = \sum_{j=1}^{N} c_{hj} x_j(k), \qquad h = 1 \,,\ldots,N_o \tag{17}$$

The proposed algorithm is now described.

4.3 THE ALGORITHM

The forward pass, backward pass and the procedure for updating the weight matrices are described in the following subsections.

4.3.1 Forward pass

The instants at which the intermediate state vectors are stored are defined as $s_1, s_2, \ldots s_{N_s}$ which are evenly spaced over the interval 0, ... K_p with $s_1 = 0$ and $s_{N_s} = K_p$. At any instant k the following steps are performed:

Step 1: The input vector $\{u_j(k)\}$ is stored as it is required for the recomputation of the state vector in the backward pass (see (9)).

Step 2: The state vector $\{x^f_i(k+1)\}$ is computed from the state vector $\{x^f_i(k)\}$ using system equation (11) by substituting $x^f_i(k) = x_i(k)$.

Step 3: The state vectors at evenly spaced intermediate time intervals are stored according to:

$$\textit{if } (k = s_j) \textit{ then } x^s_i(j)=x^f_i(k), \qquad j = 1,..,N_s; \quad i = 1,..,N \tag{18}$$

where N_s is the number of state vectors to be stored. Note that $\{x^s_i(j)\}$ is the matrix of intermediate state vectors stored in the forward pass.

Step 4: The output vector $\{y_h(k)\}$ is computed using equation (9) of Chapter 3.

Step 5: The output error is computed and stored according to:

$$e_h(k) = (r_h(k)-y_h(k)), \qquad h = 1,..N_o \tag{19}$$

Step 6: The state vectors are used to compute a forward pass scalar shadow output $y^{fs}(k)$ (which is to be compared later with the corresponding shadow output computed in the backward pass) given by:

$$y^{fs}(k) = \left(\sum_{j=1}^{N} c^s_j x_j(k)\right) \tag{20}$$

where c^s_j are the constant weights connecting the state variables to the shadow output unit.

4.3.2 Backward pass

The elements of $W^{\epsilon} = \{w_{ij}^{\epsilon}\}$ are computed by simple manipulation and scaling of the matrix elements of W and are given in (10). Ensuring the condition $w^{\delta} \neq 0$ will be discussed in Chapter 5.

At each instant k the following steps are performed:

Step 1: An estimate $\{x_i^r(k-1)\}$ of the state vector $\{x_i^f(k-1)\}$ is recomputed according to:

$$\begin{aligned} &\textit{if } (k = s_j) \textit{ then } && x_i^r(k) = x_i^s(j) && j = 1,..,N_s, \quad i = 1,..,N \\ &\textit{else } x_i^r(k) = \sum_{j=v}^{v+1} w_{ij}^{\epsilon}\left[f_a^{-1}(x_i^r(k+1)) - \sum_{j=1}^{M} b_{ij}\mu_j(k)\right], && \begin{matrix} v=i & \textit{if } i \textit{ is odd} \\ v=i-1 & \textit{if } i \textit{ is even} \end{matrix} \end{aligned} \tag{21}$$

The recomputation of the state vector is illustrated in Figure 4.2.

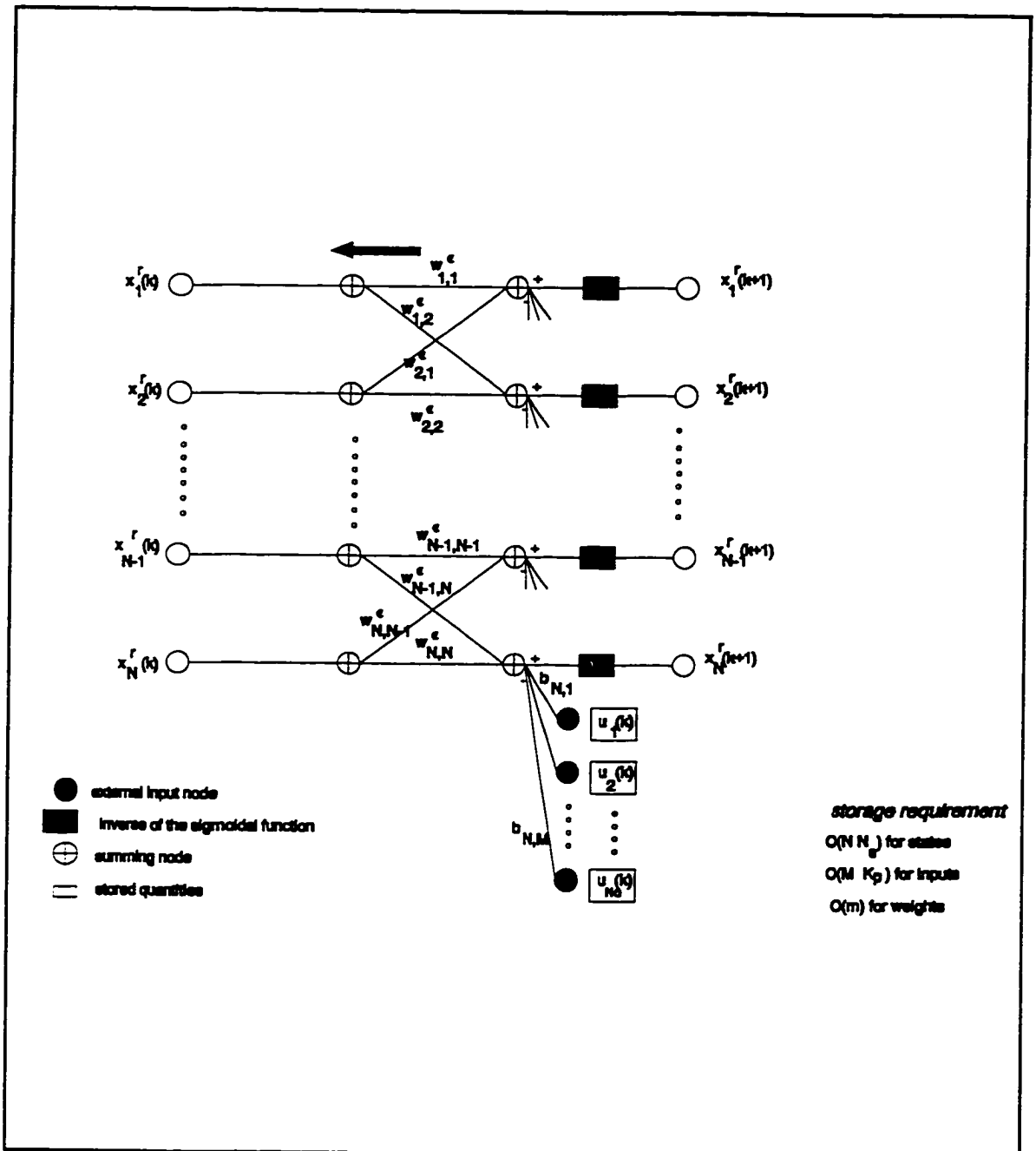


Figure 4.2 State vector recomputation in the recomputed state vector modified BPTT algorithm.

Step 2: The shadow output scalar $y^{fs}(k)$ is recomputed as an estimate $y^{rs}(k)$ given by:

$$y^{rs}(k) = \left(\sum_{j=1}^{N} c_j^s x_j^r(k)\right) \tag{22}$$

Step 3: The scalar shadow error $e^s(k)$, computed as

$$e^s(k) = \left(y^{rs}(k) - y^{fs}(k)\right) \tag{23}$$

which gives an indication of the amount of numerical error in the recomputation of the state vectors $\{x_i^f(k)\}$ at any time instant k.

Step 4: When $|e^s(k)|$ exceeds a certain threshold E, which indicates the occurrence of numerical instability, the state vector at the k-th step is computed from the nearest lower intermediate stored state vector $\{x_i^s(j_L)\}$ by using the forward pass (11) iteratively, with $j_L = max\ \{\ j : j < k\ , j = 1,..,N_s\ \}$

Step 5: $\varepsilon_i(k)$ are computed from the value of $\varepsilon_i(k+1)$, the stored values of $e_h(k)$ and $\{x_i^r(k)\}$ using:

$$\begin{aligned} \varepsilon_i(k) &= -\sum_{h=1}^{N_o} e_h(k) f_b'\left(\sum_{j=1}^{N} c_{hj} x_j^r(k)\right) c_{hi} + \sum_{j'=v}^{v+1} w_{ij}^T f_a'(s_j(k+1))\varepsilon_j(k+1) \\ \text{where}\quad s_j(k+1) &= \sum_{j'=v}^{v+1} w_{j'j'} x_i^r(k+1) + \sum_{i'=1}^{M} b_{j'i'} u_i(k+1) \qquad j' = 1,\ldots,N \end{aligned} \tag{24}$$

with the initial value $\varepsilon_i(K_p)$ given by:

$$\varepsilon_i(K_p) = -\sum_{h=1}^{N_o} e_h(K_p) f_b'\left(\sum_{j=1}^{N} c_{hj} x_j^f(K_p)\right) c_{hi} \tag{25}$$

The computation of $\varepsilon_i(k)$ for the numerically stable recomputed state variable modified BPTT algorithm is illustrated in Figure 4.3.

Step 6: The gradient computations for the W matrix are performed according to (5) and (6) with $x^r_i(k)$ substituted for $x_i(k)$. Similarly, the gradient computation

for the input matrix B is performed with (13) and (14) and the output matrix C is performed with (16) and (17).

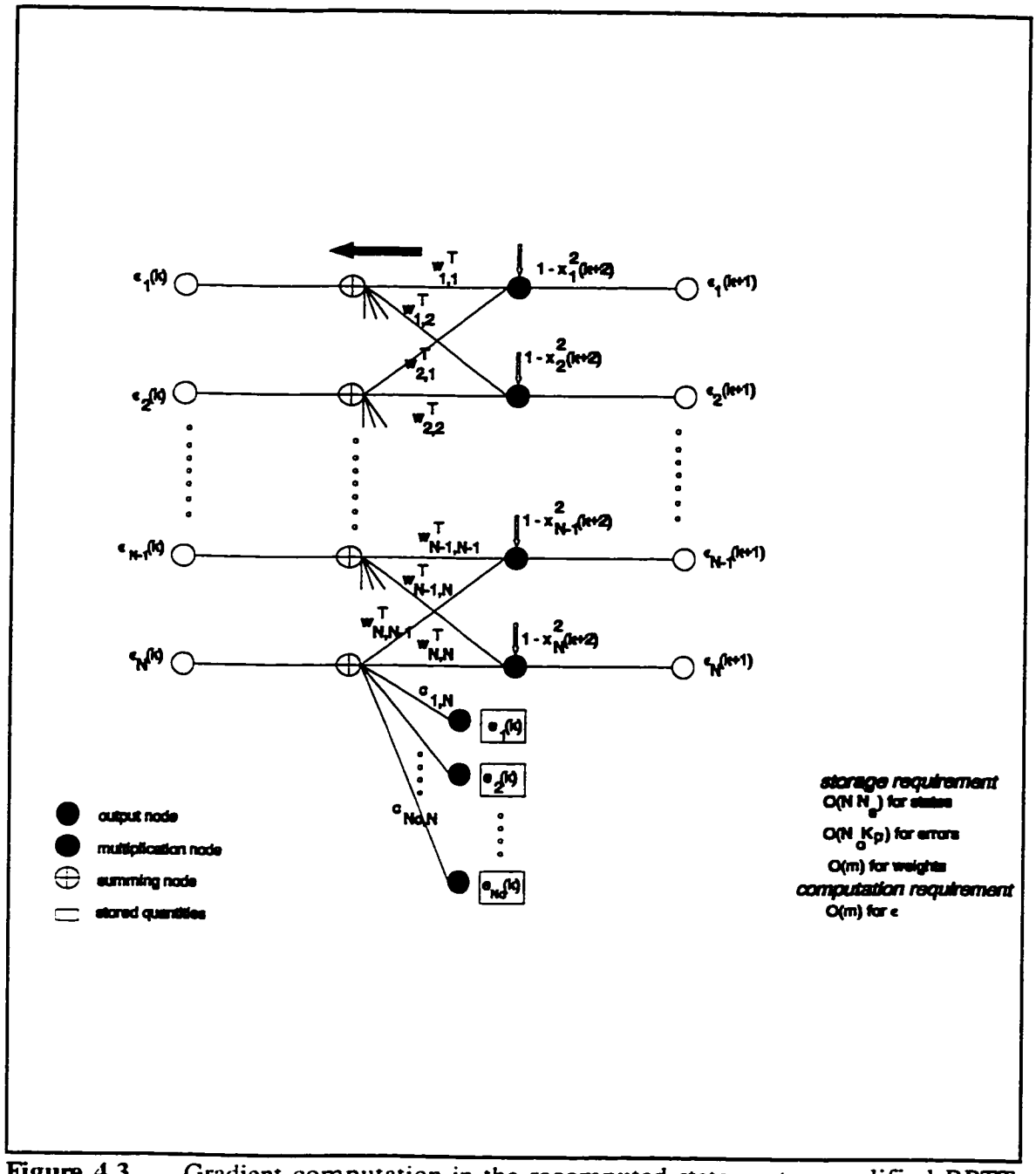


Figure 4.3 Gradient computation in the recomputed state vector modified BPTT algorithm.

4.3.3 Weight updates

At the end of K_p steps, the weights for the W, B and C matrices are performed according to (3), (12) and (15) respectively.

4.4 STORAGE AND COMPUTATION REQUIREMENT

The recomputed state variable modified BPTT algorithm requires storage of $O(NN_s)$ for storing the state vector at evenly spaced intermediate N_s time intervals. In addition it requires storage of $O(N_oK_p + MK_p + K_p)$, for storing the error vector $\{e_h(k)\}$, the input vector $\{u_i(k)\}$ and the scalar shadow output $y^{fs}(k)$. In conventional BPTT, the state vectors $\{x_i(k)\}$ and the error vectors $\{e_h(k)\}$ are stored at each instant and require storage of $O(NK_p + N_oK_p)$. Thus, with N_s a fraction of K_p, the storage requirement of the proposed algorithm is a fraction of that required for conventional BPTT. This storage reduction is achieved by a tradeoff of computational requirement when compared to conventional BPTT.

In the best case, when no numerical instabilities are encountered at any time step, the proposed algorithm computes the state variables twice, once in the forward pass and once in the backward pass and requires $O(2mK_p=4NK_p)$ computations for obtaining the state vectors. In the worst case, which corresponds to encountering numerical instability at every time step, the number of computations required to obtain the state vectors is $O(2K_p^2N/N_s)$. In reality one would expect that the actual computations required to obtain the state vectors are larger than that for the best case and far smaller than that for the worst case. This issue will be examined in detail, for a practical problem in Section 6.9 of Chapter 6.

All other computations required to obtain the gradient are the same as for conventional BPTT and is $O(N^2)$ at every time step.

4.5 CONCLUSION

The proposed algorithm is a modification of the online BPTT algorithm, and computes the exact gradient by recalculating the state vectors. The question of ensuring numerical stability of the algorithm is addressed by the following: first, state vectors at evenly-spaced intermediate time intervals are stored in the forward pass; second, these intermediate stored values are used as initial values to recompute the state vectors during the backward pass while simultaneously monitoring for signs of numerical instability; when numerical instability is detected, a recovery is made by recomputing the state vector for that time instant by performing a forward pass using the nearest stored intermediate state vector and the process of recomputing the state vectors is continued. The proposed algorithm results in reduced storage compared to conventional BPTT by trading off some of the later's computational advantage.

5 STABILIZATION OF BDRNN STRUCTURES

5.1 INTRODUCTION

This chapter derives the stability conditions for the two BDRNN structures introduced in Chapter 3. These stability conditions are fashioned into penalty functions that are implemented as part of the learning algorithm and guide the BDRNN towards stabilization. The stability conditions can also be implemented as constrained feedforward networks and ingrained as part of the FF-BDRNN architecture.

It was seen in Section 2.3.1 of Chapter 2, that the stability of the recursive computations performed to compute the gradients that update the DTRNN weights requires that the DTRNN itself be stable at its equilibrium point at each weight update [Williams and Zipser 1989, Almeida]. In Section 2.3.3 of Chapter 2, it was shown that a sufficient condition for the *local* stability of the fully recurrent DTRNN with a nonsingular feedback weight matrix W^f was [Williams and Zipser 1989, Kung]

$$| \lambda_i(W^f) | \leq \frac{2}{a} \qquad i = 1,\ldots,N \tag{1}$$

where $\lambda_i(W^f)$ is the i-th eigenvalue of the square matrix $W^f = \{w^f_{ij}\}$. In Section 2.3.2 of Chapter 2, it was shown that a sufficient condition for the *global* stability of the DTRNN with feedback weight matrix W^f was [e.g., Jin and Gupta 1996, Barnes]

$$\left[\lambda_{max}((W^f)^T W^f) \right]^{1/2} < \frac{2}{a} \tag{2}$$

where $\lambda_{max}(.)$ is the maximum eigenvalue of the matrix. From (1) and (2), it is clear that the local or global stability of the DTRNN can be addressed by monitoring the eigenvalue with the largest magnitude of W^f or $(W^f)^T W^f$, respectively, at each weight update.

The local and global stability conditions (1) and (2) directly apply to the BDRNN architecture. Without loss of generality, all the results pertaining to stability conditions for BDRNN will be derived in this chapter using $a = 2$ in (8) of Chapter 3. As the derivative of the symmetric sigmoidal function is unity for $a = 2$, the local or global stability of the BDRNN depend on the eigenvalue with the largest magnitude of W or $W^T W$, respectively.

5.2 LOCAL AND GLOBAL STABILITY APPROACHES

For an autonomous nonlinear dynamic system whose equilibrium point is given by $\underline{x}$, the following are the definitions for the uniform stability and convergence of the equilibrium state, asymptotic stability and global stability of the system as given in [Haykin 1994].

Definition 1: The equilibrium state $\underline{x}$ is said to be uniformly stable if for any given positive ϵ, there exists a positive δ such that the condition:

$$\|x(0)-\underline{x}\| < \delta \quad \rightarrow \quad \|x(t)-\underline{x}\| < \epsilon \qquad t > 0 \tag{3}$$

This definition states that a trajectory of the system can be made to stay within a small neighbourhood of the equilibrium state $\underline{x}$ if the initial state $x(0)$ is close to $\underline{x}$.

Definition 2: The equilibrium state $\underline{x}$ is said to be convergent if there exist a positive δ, such that the condition:

$$\|x(0)-\underline{x}\| < \delta \qquad \textit{implies that} \quad x(t) \rightarrow \underline{x} \quad \textit{as} \quad t \rightarrow \infty \tag{4}$$

This definition means that if the initial state $x(0)$ of a trajectory is close enough to the equilibrium state $\underline{x}$, then the trajectory described by the state vector $x(t)$ will approach $\underline{x}$ as time t approaches infinity.

Definition 3: The equilibrium state $\underline{x}$ is said to be asymptotically stable if it is both stable and convergent.

Definition 4: The equilibrium state $\underline{x}$ is said to be globally asymptotically stable if it is stable and all trajectories of the system converge to $\underline{x}$ as $t \rightarrow \infty$.

This definition implies that the system cannot have other equilibrium states, and it requires that every trajectory of the system remains bounded for all time $t > 0$. Global asymptoticity implies that the system will ultimately settle down to a steady state for any choice of initial conditions.

In this section, the problem of ensuring FF-BDRNN stability during the training phase is pursued. As seen in Section 2.3.3, linearizing the state-space equation (1) of Chapter 2 provides useful information about the local asymptotic stability properties of an equilibrium state. It was seen that, the eigenvalues of the nonsingular feedback weight matrix W determine the local behaviour of the trajectories of the system in the neighbourhood of the equilibrium state. If all the eigenvalues of W are within the unit circle, then the equilibrium state $x = 0$, is a local asymptotical point of system (7) of Chapter 3 and this equilibrium point is a *sink*. If all the eigenvalues of W are outside the unit circle, then the equilibrium state $x = 0$, is an unstable equilibrium point of the system and this equilibrium point is a *source*. If some of the eigenvalues of W are within and some are outside the unit circle, then the equilibrium state $x = 0$, is an unstable equilibrium point of the system and this equilibrium point is a *saddle*. It must be noted that, while the forward propagation equation (7) of Chapter 3, which defines the network dynamics is nonlinear, the error propagation equation (19) of Chapter 4 is linear. This means that in order to guarantee the local stability of the learning dynamic, it is essential to closely match the dynamic properties of the error propagation network (learning dynamic) with those of the forward propagation network (network dynamic) [Almeida]. As seen in Section 2.3.3, the local stability of the network dynamic depends on the eigenvalues of W, while the local stability of the learning dynamic depends on the

eigenvalues of W^T. However since the eigenvalues of a matrix and its transpose are identical, hence, the local stability of the network dynamic is a sufficient condition for the local stability of the learning dynamic.

As seen in Section 2.3.2, the global asymptotic stability conditions for a nonlinear dynamic system can be found by applying the contraction mapping theorem. It was seen that, the square root of the eigenvalues of W^TW determine the global behaviour of the trajectories of the system for any initial state variable $\boldsymbol{x}(0)$ of the system. The global stability of the network dynamic depends on the eigenvalues of W^TW, while that of the learning dynamic depends on that of WW^T. Hence, the global stability of the network dynamic is a sufficient condition for the global stability of the learning dynamic as well.

It is desirable to ensure global stability in order to ensure the absolute stability of the entire system, because the absolute stability does not depend on the initial conditions of the state variables $\boldsymbol{x}(0)$ of the system. However, for most practical problems with bounded external inputs and random initial state variables $\boldsymbol{x}(0)$, it is sufficient if the trajectory $\boldsymbol{x}(t)$ approach $\underline{x}$ as $t \rightarrow \infty$. Here, satisfying the absolute global stability requirement may place too many restrictions on the placement of the weights. On the other hand the local stability, ensures that the trajectory of the system can be made to stay within a small neighbourhood of the equilibrium state $\underline{x}$ provided that the initial state variables $\boldsymbol{x}(0)$ are close to $\underline{x}$ and that this trajectory $\boldsymbol{x}(t)$ will approach $\underline{x}$ as time $t \rightarrow \infty$. For a specific bounded input bounded output problem, especially in problems where there is some knowledge of the initial state variables $\boldsymbol{x}(0)$ of the system, the local stability while not guaranteeing global stability, may afford better placement of the weights and be sufficient to ensure stable learning with a small learning rate. In view of this, both global and local stability requirements are presented in the following development.

The local and global stability conditions for the two BDRNN structures will now be derived, in section 5.2.1 for scaled orthogonal BDRNN and in section 5.2.2 for freeform BDRNN.

5.2.1 BDRNN with scaled orthogonal submatrices

Theorem 1: The local and global stability conditions are identical for a BDRNN with scaled orthogonal submatrices. The BDRNN with scaled orthogonal submatrices is globally and locally stable if the elements of each of its submatrices satisfy the condition that its determinant is less than or equal to 1.

Proof: Let $W'_{n/2}$ denote the $n/2$-th 2 x 2 submatrix of the block-diagonal W obtained by extracting rows n-1,n and columns n-1,n and constrained to be scaled orthogonal.

$$W'_{n/2} = \begin{bmatrix} w_{n-1,n-1} & w_{n-1,n} \\ -w_{n-1,n} & w_{n-1,n-1} \end{bmatrix} \quad n=2,4,..,N \tag{5}$$

Note that for a scaled orthogonal 2 x 2 submatrix the global and local stability conditions (1) and (2), respectively, are equivalent

$$\begin{aligned} \lambda_{\max}\left((W'_{n/2})^T(W'_{n/2})\right) &= \lambda_{\max}\begin{bmatrix} w_{n-1,n-1}^2+w_{n-1,n}^2 & 0 \\ 0 & w_{n-1,n-1}^2+w_{n-1,n}^2 \end{bmatrix} \\ &= w_{n-1,n-1}^2+w_{n-1,n}^2 \\ |\lambda(W'_{n/2})|_{\max} &= \left(w_{n-1,n-1}^2+w_{n-1,n}^2\right)^{1/2} \end{aligned} \tag{6}$$

$$\textit{Hence} \quad \left[\lambda_{\max}\left((W'_{n/2})^T(W'_{n/2})\right)\right]^{1/2} < 1 \equiv |\lambda(W'_{n/2})|_{\max} < 1 \quad n=2,4,...,N$$

For this case, the local *and* global stability conditions are given by,

$$w_{n-1,n-1}^2 + w_{n-1,n}^2 \leq 1.0 \qquad n=2,4,..,N \tag{7}$$

5.2.2 BDRNN with freeform submatrices

Let $W''_{n/2}$ denote the $n/2$-th 2 x 2 matrix of the block-diagonal W obtained by extracting rows n-1,n and columns n-1,n and in which each of the elements is allowed to take any value:

$$W''_{n/2} = \begin{bmatrix} w_{n-1,n-1} & w_{n-1,n} \\ w_{n,n-1} & w_{n,n} \end{bmatrix} \qquad n=2,4,..,N \tag{8}$$

5.2.2.1 Global stability condition

Theorem 2: The BDRNN with freeform submatrices is globally stable if the elements of each of its submatrices satisfy the condition that the sum of the square of its elements less the square of its determinant is less than or equal to 1.

Proof: Let:

$$W''^T_{n/2} W''_{n/2} = \begin{bmatrix} a_{n/2,1} & a_{n/2,3} \\ a_{n/2,3} & a_{n/2,2} \end{bmatrix}$$

Then

$$\begin{aligned} a_{n/2,1} &= w_{n-1,n-1}^2 w_{n,n-1}^2 \\ a_{n/2,3} &= w_{n-1,n-1} w_{n-1,n} + w_{n,n-1} w_{n,n} \\ a_{n/2,2} &= w_{n-1,n}^2 w_{n,n}^2 \qquad n=2,4,..,N \end{aligned} \tag{9}$$

For this case, the global stability condition (2) is

$$\lambda_{max}\left(W''^T_{n/2} W''_{n/2}\right) = \frac{1}{2}\left[(a_{n/2,1} + a_{n/2,2}) + \sqrt{(a_{n/2,1} - a_{n/2,2})^2 + 4a_{n/2,3}^2}\right] \tag{10}$$

From (2), the global stability condition is given by:

$$a_{n/2,3}^2 \leq 1 + a_{n/2,1}a_{n/2,2} - (a_{n/2,1} + a_{n/2,2}) \qquad (11)$$

Using (9) in (11), the global stability condition (2) for the freeform submatrix is given by:

$$(w_{n-1,n-1}^2 + w_{n,n-1}^2 + w_{n-1,n}^2 + w_{n,n}^2) - (w_{n-1,n-1}w_{n,n} - w_{n,n-1}w_{n-1,n})^2 \leq 1 \qquad (12)$$

5.2.2.2 Local stability condition

Theorem 3: The BDRNN with freeform submatrices is locally stable, if each of its submatrices satisfy one of the following conditions:

Case (i): If $(T_{n/2})^2 > 4\nabla_{n/2}$

where $T_{n/2}$ is the trace of $W''_{n/2} = w_{n-1,n-1} + w_{n,n}$ and $\nabla_{n/2}$ is the determinant of $W''_{n/2}$, then the local stability condition is given by:

$$\left|\frac{T_{n/2}}{2}\right|\left(1 + \sqrt{1 - \frac{4\nabla_{n/2}}{(T_{n/2})^2}}\right) \leq 1 \qquad (13)$$

Case (ii): If $(T_{n/2})^2 = 4\nabla_{n/2}$ then the local stability condition is given by:

$$\left|\frac{T_{n/2}}{2}\right| \leq 1 \qquad (14)$$

Case (iii): If $(T_{n/2})^2 < 4\nabla_{n/2}$ then the local stability condition is given by:

$$|\nabla_{n/2}| \leq 1 \qquad (15)$$

Proof: The eigenvalues of $W''_{n/2}$ are given by:

$$\frac{T_{n/2}}{2} \pm \sqrt{\Omega_{n/2}}$$

$$\text{where} \quad \Omega_{n/2} = \left(\frac{T_{n/2}}{2}\right)^2 - \nabla_{n/2} \tag{16}$$

Case (i) If $\Omega_{n/2}$ in (16) is positive, then the eigenvalues (16) are real and unequal. The local stability condition determined by the eigenvalue with the larger magnitude is given by:

$$\left|\frac{T_{n/2}}{2} + \frac{sign(T_{n/2})}{2}\sqrt{\Omega_{n/2}}\right| \leq 1 \tag{17}$$

(17) can be rewritten as:

$$\left|\frac{T_{n/2}}{2}\right|\left(1 + \sqrt{1 - \frac{4\nabla_{n/2}}{(T_{n/2})^2}}\right) \leq 1 \tag{18}$$

Case(ii): If $\Omega_{n/2}$ in (16) is zero, i.e., $(T_{n/2})^2 = 4\nabla_{n/2}$, then the eigenvalues of $W''_{n/2}$ are real and equal. In this case, the condition for local stability is given by:

$$\left|\frac{T_{n/2}}{2}\right| \leq 1.0 \tag{19}$$

Case (iii): If $\Omega_{n/2}$ in (16) is negative, then the eigenvalues of $W''_{n/2}$ are complex conjugates. In this case, the condition for local stability is given by

$$\left[\left(\frac{T_{n/2}}{2}\right)^2 + \Omega_{n/2}^2\right] \leq 1.0$$

$$\Rightarrow \left| w_{n-1,n-1}w_{n,n} - w_{n-1,n}w_{n,n-1} \right| \leq 1.0 \qquad (20)$$

$$i.e., \quad \left|\nabla_{n/2}\right| \leq 1.0$$

Q.E.D

5.3 PENALTY FUNCTIONS AND STABILITY FUNCTIONS

The approach taken here is to minimize the output error function subject to the constraint that the eigenvalues in (1) and/or (2) are limited to stay within a stability region. As indicated in Section 3.4.1 of Chapter 3, this problem is overcome by including in the output error function a suitable penalty term P_t (see (19) of Chapter 3), that is a function of the relative stability of the FF-BDRNN. The technique considered here is to determine where the eigenvalues are located in relation to the unit circle and use this information to guide the eigenvalue placement towards a more stable configuration while simultaneously reducing the system output error at each update of W. To this end, a stability function that quantifies the stability of the FF-BDRNN at each weight update is considered. The penalty function P_t is then defined in terms of the error between the stability function and its desired value.

Using the following notations

$$\begin{aligned} W^* &= W \quad && \textit{for local stability} \\ &= W^T W \quad && \textit{for global stability} \end{aligned} \qquad (21)$$

and representing the i-th 2 x 2 block-diagonal submatrix of W^* as W^*_i, the stability function y_i^s which quantifies the stability of W_i^* can be expressed as

$$y_i^s = g\left(\left|\lambda(W_i^+)\right|_{max} \right) \qquad i = 1,...,N/2 \tag{22}$$

where $g(x)$ is a suitably defined monotonously non-decreasing function of the maximum magnitude of the eigenvalue of W_i^+.

The penalty function P_t can be defined as the mean squared error between this stability function y_i^s and its desired value r^s i.e.,

$$P_t = \frac{1}{2}\sum_{i=1}^{N/2} \left(r^s - y_i^s\right)^2 \tag{23}$$

The stability function $g(.)$ should be chosen such that whenever the stability condition $y_i^s = r^s$ is violated, P_t increases the value of the error function E_t with larger violations resulting in larger increases in E_t. E_t is given by:

$$E_t = J_t + P_t , \qquad t = 1, .. , T \tag{24}$$

where J_t is the mean squared output error for the t^{th} sequence of the T total number of training sequences as defined by (20) of Chapter 3. The choice of $g(.)$ is addressed in the following sections.

5.3.1 Ideal stability function

An ideal stability function $g(.)$ is given by:

$$\begin{aligned} g\left(\left|\lambda(W_i^+)\right|_{max}\right) &= -1 && \text{for } \left|\lambda(W_i^+)\right|_{max} \le 1.0 \\ &= 1 + \left(\left|\lambda(W_i^+)\right|_{max} - 1\right)k_s && \text{for } \left|\lambda(W_i^+)\right|_{max} > 1.0 \end{aligned} \tag{25}$$

where the constant $k_s > 0$ as shown in Figure 5.1(a). Here, the desired value $r^s = -1.0$. With the stability function as defined in (25), at any given instant during training, if all the eigenvalues of W^+ lie inside the unit circle then, (i) the stability condition $y_i^s = -1.0$

is satisfied, hence, (ii) the penalty term $P_t = 0$ and hence, (iii) only the mean squared system output error contributes to E_t.

At any given instant during training, if some or all the eigenvalues of W^+ were to lie outside the unit circle then, (i) the stability condition is violated and $y_i' = -1.0$, hence, (ii) E_t contains both stability error P_t and system output error J_t components and also, (iii) the more unstable the system, the larger the stability error component P_t and the larger the resultant E_t.

The ideal stability function of Figure 5.1(a) has a major drawback in that it is nondifferentiable at the border of the feasible region:

$$\left|\lambda(W_t^+)\right|_{max} = 1.0 \qquad (26)$$

This means that if the ideal penalty function as given in (25) were implemented as part of the error function E_t given in (24) then as the parameters of the BDRNN are varied, the discontinuous first-order derivatives of E_t along the boundary defined by (26) would potentially result in parasitic oscillations, abrupt discontinuities, or nonconvergence of the corresponding learning curve [Baldi]. Learning may be disrupted in the following ways: when unwanted abrupt changes occur in the trajectory space of a dynamical system, or when desirable changes to the structure of its trajectories are prevented from occurring. Such phenomena might be observed when the stability condition is satisfied for a training pattern and is violated for the next and so on. To avoid these problems and to ensure good learning, it is essential that the selected stability function is smooth and continuously differentiable with respect to the elements of W.

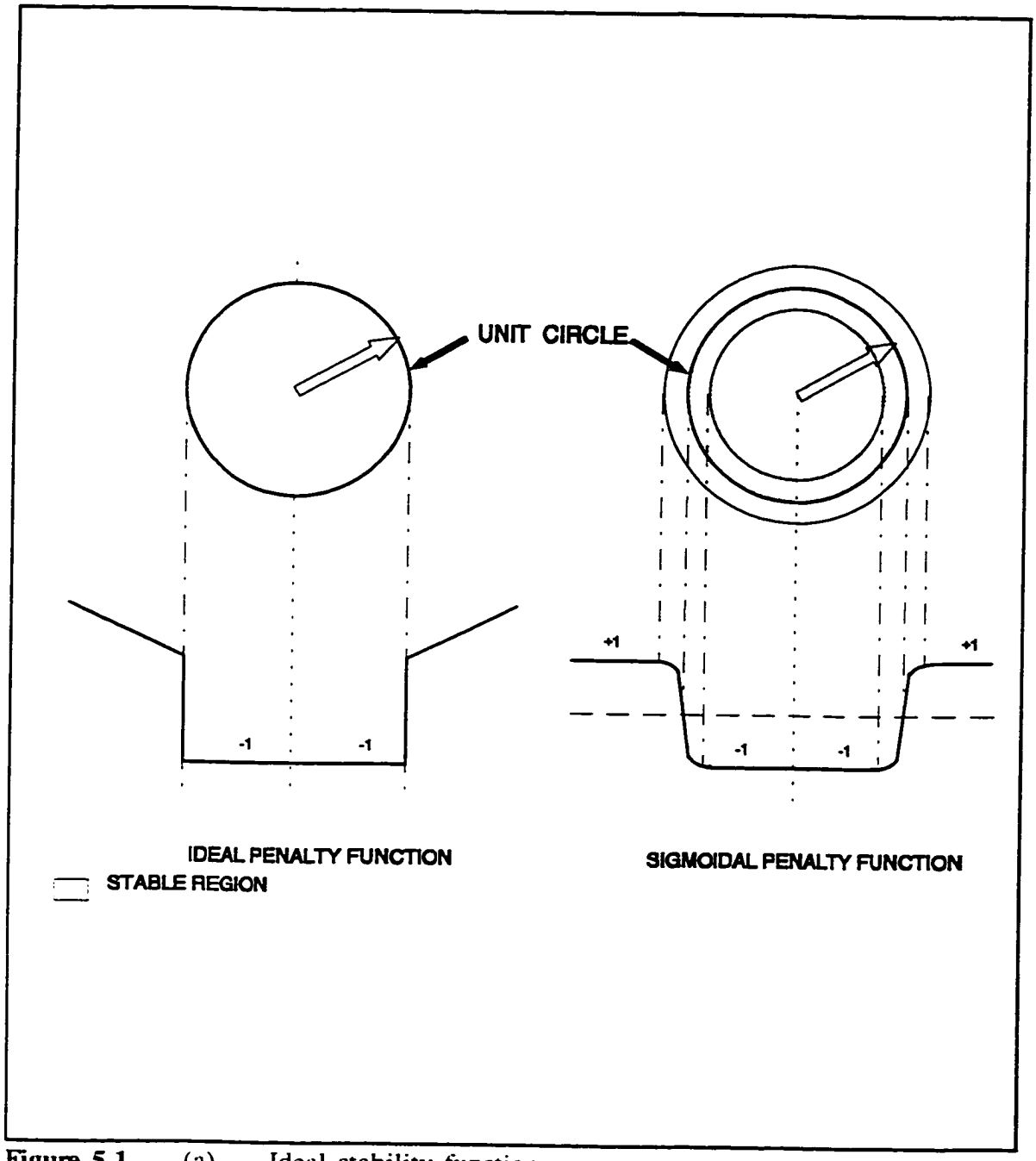


Figure 5.1 (a) Ideal stability function.
(b) Sigmoidal stability function.

5.3.2 Sigmoidal stability function

A smooth and continuously differentiable function that provides a good approximation to the ideal stability function of Figure 5.1(a) is the sigmoid function $f_d(.)$ given by:

$$g(x) = f_d(1 - x) = \frac{1-e^{-d(1-x)}}{1+e^{-d(1-x)}}, \quad d = n\alpha, \; n > 1 \tag{27}$$

where n is chosen such that (27) has a sharp midpoint slope as shown in Figure 5.1(b). For this function, the stability condition is still $y^S = -1.0$.

From Figure 5.1(b), it is clear that if the eigenvalues of W^* lie within a circle of radius $\rho_1 < 1.0$, (i) then the BDRNN is stable and in this region, $y_i^S \approx -1.0$, hence, (ii) $P_t \approx 0$ and (iii) only the mean squared system output error contributes to E_t.

Also, Figure 5.1(b) describes an annular disc of radius 1.0 whose width $(\rho_2-\rho_1)$ is a function of the slope of the sigmoid function $f_d(.)$ chosen. If the eigenvalues of W^* lie in this annular region, (i) the BDRNN is marginally stable/unstable, hence, (ii) the stability condition $y_i^S = -1.0$ is marginally violated, and (iii) E_t contains a small stability error component together with the system output error component.

The area outside this annular disc corresponds to the region of instability. If any of the eigenvalues of W^* were to lie well within this unstable region, then E_t has a large near-constant stability error component in addition to the system output error component.

Note that although any smooth continuously differentiable non-saturating stability function can be used, here the sigmoidal stability function is used as it provides for better numerical conditioning for large stability violations.

It is desirable to formulate the above stability function in a neural network framework such that the stability of the BDRNN at each training instant is computed *locally* in an auxiliary neural network. The motivation for doing so are outlined below:

(i) In biological systems, learning of global tasks (i.e., a task that the organism is trying to learn) is widely thought to result from local synaptic changes (i.e., synaptic changes brought about by the local electrochemical environment). This is a fundamental motivation for artificial neural network researchers to use spatially local computations, in their effort to "mimic" biological systems.

(ii) It is also advantageous to use local computations, since, this means that each node requires information only from other nodes to which it connects, resulting in simpler hardware implementations as opined by several researchers [Almeida, Pineda, Day and Davenport].

(iii) Also, such an auxiliary network when added on to the existing BDRNN would allow the use of a unified gradient search technique to ensure guidance towards network stabilization *simultaneously* with output error reduction.

However, it should be noted that the stability constraint may also be implemented by other means that do not require the use of a neural network.

5.4 STABILITY AND PENALTY FUNCTIONS FOR BDRNN STRUCTURES

The local and global stability functions together with the corresponding penalty functions are now derived for the scaled orthogonal and freeform block diagonal structures of the BDRNN feedback weight matrix, introduced in Section 3.4.3 of Chapter 3. These two structures differ from each other in terms of complexity and the degree of freedom

allowed in the values that their elements can assume. In the first case, each 2 x 2 submatrix of the block-diagonal feedback matrix is scaled orthogonal and models a second order dynamic system with real or complex-conjugate eigenvalue pairs using only two distinct elements. As was seen in Section 5.2.1, the global and local stability conditions for this structure are the same and hence, satisfying the former does not impose additional restrictions on the weights assignment. In the second case, each element of the 2 x 2 submatrices can assume any value and each of such freeform submatrices models a second order dynamic system with real or complex-conjugate eigenvalue pairs using four distinct elements. As seen in section 5.2.2, the global stability condition for this structure is different from the local stability condition and imposes more restrictions on the weights assignment. While the BDRNN with scaled orthogonal submatrices has fewer elements and is hence less complex, the BDRNN with freeform submatrices has more elements and hence a larger degree of freedom.

5.4.1 Local /Global stability and penalty functions for scaled orthogonal BDRNN

For BDRNN with scaled orthogonal submatrices, $(W'_{n/2})^T W'_{n/2}$ is a diagonal matrix with each diagonal element equal to the square of the magnitude of the complex eigenvalues of $W'_{n/2}$ as shown in (6). Hence, for this case as seen in Section 5.2.1, the local *and* global stability conditions are equivalent and given by (7). However, from equation (10) of Chapter 4, it is seen that W must be invertible for the recomputation of the state variables in the backward pass. To ensure the invertibility of W, it is essential that **det** $(W'_{n/2}) \geq w^{\delta}_{min}$, where $w^{\delta}_{min} \gg 0$. Hence (7) is modified as:

$$w^{\delta}_{\min} \leq w^2_{n-1,n-1} + w^2_{n-1,n} \leq 1.0 \qquad n=2,4,..,N \tag{28}$$

A suitable stability function for (28) that implements the upper limit is given by:

$$y^s_{n/2} = f_d\left(w^2_{n-1,n-1} + w^2_{n-1,n} - 1.0\right) \qquad n=2,4,..,N \tag{29}$$

A suitable invertibility function for (28) that implements the lower limit is given by:

$$y_{n/2}^{\epsilon} = f_d\left(w_{\min}^{\delta} - (w_{n-1,n-1}^2 + w_{n-1,n}^2)\right) \qquad n=2,4,..,N \tag{30}$$

The corresponding penalty function is given by

$$P_t = \frac{1}{2}\left[\sum_{i=1}^{N/2}\left(r_i^s - y_i^s\right)^2 + \sum_{i=1}^{N/2}\left(r_i^{\epsilon} - y_i^{\epsilon}\right)^2\right] \qquad \textit{where } r_i^s = r_i^{\epsilon} = -1.0 \tag{31}$$

5.4.2 Global stability and penalty function for freeform BDRNN

A global stability condition for the freeform submatrix was derived in (9) through (12) of section 5.2.2. For BDRNN with freeform submatrices, the global stability condition given by (12) is satisfied, if the elements of each of its submatrices satisfy the condition that, the sum of the square of its elements minus the square of its determinant is less than or equal to 1. However, from (10) of Section 4.2, it is seen that W must be invertible for the recomputation of the state variables in the backward pass. To ensure the invertibility of W, it is essential that $|\det(W''_{n/2})| \geq w^{\delta}_{min}$ with $w^{\delta}_{min} \gg 0$. This is represented as:

$$w_{\min}^{\delta} \leq \left| w_{n-1,n-1}w_{n,n} - w_{n-1,n}w_{n,n-1} \right| \tag{32}$$

The stabilization function corresponding to the global stability condition (12) is given by:

$$y_{n/2}^{sg} = f_d\left[w_{n-1,n-1}^2 + w_{n,n-1}^2 + w_{n-1,n}^2 + w_{n,n}^2 - \left(w_{n-1,n-1}w_{n,n} - w_{n,n-1}w_{n-1,n}\right)^2 - 1\right] \tag{33}$$
$$n = 2,4,\ldots,N$$

An invertibility function that implements (32) is given by:

$$y_{n/2}^{\epsilon 2} = f_d\left((w_{\min}^{\delta})^2 - (w_{n-1,n-1}w_{n,n} - w_{n-1,n}w_{n,n-1})^2\right) \qquad n=2,4,..,N \tag{34}$$

The corresponding penalty function is given by

$$P_t = \frac{1}{2}\left[\sum_{i=1}^{N/2}\left(r_i^{sg} - y_i^{sg}\right)^2 + \sum_{i=1}^{N/2}\left(r_i^{\epsilon 2} - y_i^{\epsilon 2}\right)^2\right] \quad \textit{where } r_i^{sg} = r_i^{\epsilon 2} = -1.0 \tag{35}$$

5.4.3 Local stability and penalty function for freeform BDRNN

The local stability condition (1) for BDRNNs with freeform submatrices was derived in Section 5.2.2.2 using the eigenvalues of $W''_{n/2}$. The eigenvalues of $W''_{n/2}$ can be (a) real and unequal, (b) real and equal, or (c) complex conjugates. The local stability conditions for these three cases are given by (18), (19) and (20) respectively. These local stability conditions should be considered along with (32) to ensure invertibility of W for the recomputation of the state vector in the backward pass.

5.4.3.1 Real and unequal eigenvalues

Let us now consider the case when Ω in (16) is positive i.e., the eigenvalues of $W''_{n/2}$ are real and unequal. In this case, the stability condition is determined by the larger of the two eigenvalues of $W''_{n/2}$, ie..

$$\left| \frac{T_{n/2}}{2} + \frac{sign(T_{n/2})}{2}\sqrt{\Omega_{n/2}} \right| \leq 1.0 \tag{36}$$

i.e. if $T_{n/2} \geq 0.0$ then

$$\left(\frac{T_{n/2}}{2}\right)^2 + \left(\frac{T_{n/2}}{2}\right)\sqrt{\Omega_{n/2}} + \Omega_{n/2} - 1.0 \leq 0.0 \tag{37}$$

and if $T_{n/2} < 0.0$, then

$$\left(\frac{T_{n/2}}{2}\right)^2 - \left(\frac{T_{n/2}}{2}\right)\sqrt{\Omega_{n/2}} + \Omega_{n/2} - 1.0 \leq 0.0 \tag{38}$$

The stability function corresponding to the constraining equation (37) and (38) is given by:

$$y_{n/2}^{s3} = f_d\left[\left(\frac{T_{n/2}}{2}\right)^2 + \left(\frac{T_{n/2}}{2}\right)\sqrt{\Omega_{n/2}} + \Omega_{n/2} - 1.0\right](1 + f_d(T_{n/2}))0.5 \qquad (39)$$
$$+ f_d\left[\left(\frac{T_{n/2}}{2}\right)^2 - \left(\frac{T_{n/2}}{2}\right)\sqrt{\Omega_{n/2}} + \Omega_{n/2} - 1.0\right](1 - f_d(T_{n/2}))0.5 \leq 0.0$$

From (39), it can be seen that the SFNN that implements (38) will require a perceptron that can multiply the outputs of two different constrained feedforward neural networks. Hence a network that can implement (39) requires an algorithm that can be used for higher order networks. Thus, the sign dependency introduces a discontinuity which requires a neural network framework that uses higher order perceptrons [e.g., Kosmatopoulos et al, Giles et al 1992, Sun et al 1991] and will not be considered in this thesis. However, the case of real and unequal eigenvalues will be considered in an indirect manner as discussed later.

5.4.3.2 Real and equal or complex conjugate eigenvalues

If the eigenvalues of $W''_{n/2}$ are real and equal or complex conjugates, the local stability condition is a smooth and continuous function of the elements of $W''_{n/2}$ as seen from (19) and (20) and hence can be directly implemented in a neural network framework that uses first order perceptrons. Note that (19) is a degenerate of (20) with $w_{n,n} = w_{n-1,n-1}$ and $w_{n,n-1} = w_{n-1,n} = 0$.

The local stability condition for real and equal or complex conjugate eigenvalues mandates that $\Omega_{n/2} \leq 0$, i.e.,

$$w_{n-1,n-1}^2 - 2w_{n-1,n-1}w_{n,n} + w_{n,n}^2 + 4w_{n-1,n}w_{n,n-1} \leq 0.0 \qquad (40)$$

(20) and (40) constitute the basic constraining equations in the implementation of the proposed local stabilization technique. The local stability functions corresponding to these constraining equations are given, respectively, by:

$$y_{n/2}^{s1} = f_d\left[(w_{n-1,n-1}w_{n,n} - w_{n-1,n}w_{n,n-1})^2 - 1.0\right], \quad n=2,4,..,N \tag{41}$$

and

$$y_{n/2}^{s2} = f_d\left(w_{n-1,n-1}^2 - 2w_{n-1,n-1}w_{n,n} + w_{n,n}^2 + 4w_{n-1,n}w_{n,n-1}\right) \quad n=2,4,..,N \tag{42}$$

The penalty function for the local stabilization case which includes the invertibility function (34) is:

$$P_t = \frac{1}{2}\left[\sum_{i=1}^{N/2}\left(r_i^{s1}-y_i^{s1}\right)^2+\sum_{i=1}^{N/2}\left(r_i^{s2}-y_i^{s2}\right)^2+\sum_{i=1}^{N/2}\left(r_i^{e2}-y_i^{e2}\right)^2\right] \tag{43}$$

$$\text{where } r_i^{s1} = r_i^{s2} = r_i^{e2} = -1.0$$

The case of real and unequal eigenvalues can be considered in this framework with constraints (20) and (40), by suitably increasing the order of the block-diagonal system in which a pair of block-diagonal sub-matrices each with real and equal eigenvalues models a real and unequal eigenvalue pair. For example, a system with L real and unequal eigenvalues can be modelled in a BDRNN by a $2L$ x $2L$ freeform block-diagonal feedback weight matrix which satisfies stability functions (41) and (42).

5.5 STABILITY FUNCTION AS CONSTRAINED FEEDFORWARD NEURAL NETWORKS

In this section we will show that the stability functions (29), (33), (41) and (42) and invertibility functions (30) and (34), which are polynomials of the weight elements w_{ij}, can be implemented as constrained multi-layer feedforward neural networks. This transformation allows for ingraining these equations as auxiliary neural networks that can be added on to the existing FF-BDRNN architecture as depicted in Figure 5.2 which facilitates a stable learning process. In Figure 5.2, the stabilizing feedforward neural network (SFNN) and the feedback weight matrix W of the FF-BDRNN are linked to each other through mutual constraints. In this figure, u^s, z^s, r^s and e^s are the input, the output, the desired output and the output error, respectively, of the SFNN. The output z^s of the SFNN is a vector of the stabilization and invertibility functions. The output error e^s of the SFNN is the stability error of the overall system. The SFNN is trained using the standard backpropagation algorithm simultaneously with the training of the FF-BDRNN. During training, while the output error $e^n(k)$ of the FF-BDRNN impacts the updating of weights W, B, C and D_i's, the stability error e^s impacts the updating of weights of the SFNN and hence W.

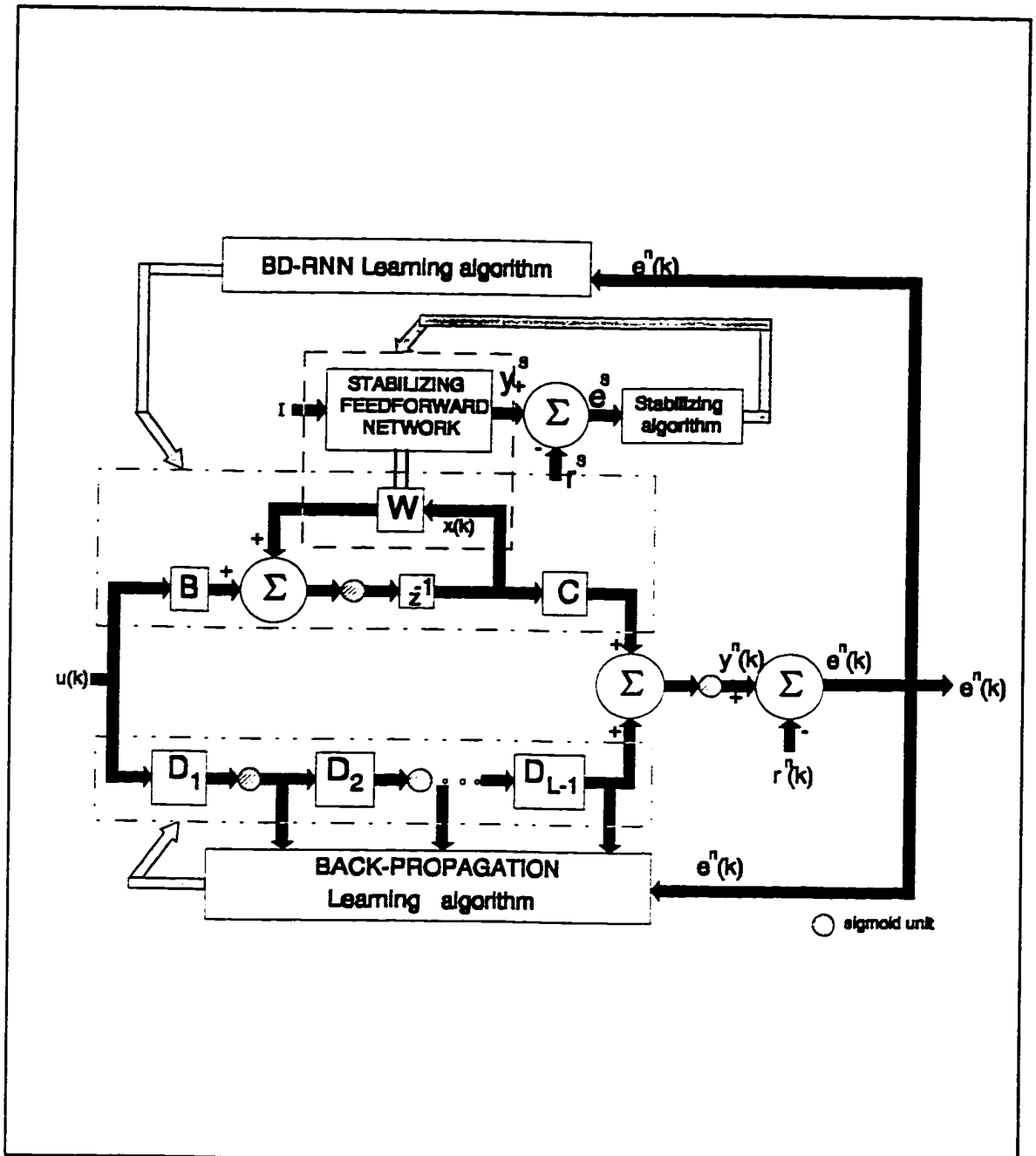


Figure 5.2 Block-diagram of stabilizing feed-forward neural network (SFNN) ingrained in the FF-BDRNN

For BDRNN with scaled orthogonal block-diagonal matrices, the stability function (29) can be transformed into an SFNN framework by the transformation described by:

$$y_{n/2}^{s} = f_d\left[w_{n-1,n-1}\, f_c(w_{n-1,n-1}) + w_{n-1,n}\, f_c(w_{n-1,n}) - \alpha_1\right], \quad 0 < \alpha_1 < 1.0; \qquad (44)$$

Note that α_1 in (44) replaces the limit 1.0 in equation (29) due to squashing of the weights in the first layer. The SFNN described by (44) is a three-layer feedforward network as depicted in Figure 5.3 for the $n/2$-th block-diagonal submatrix of W. It can be seen from this figure that the SFNN has a constrained architecture with its weights set equal to the elements of the scaled orthogonal W. The input u^s to the SFNN is the constant bias 1. The bias input at the output layer is $-\alpha_1$. The output z^s of the SFNN is a vector of the elements $y_{n/2}^{s}$. In a similar manner the invertibility function (30) can also be transformed into a feedforward network.

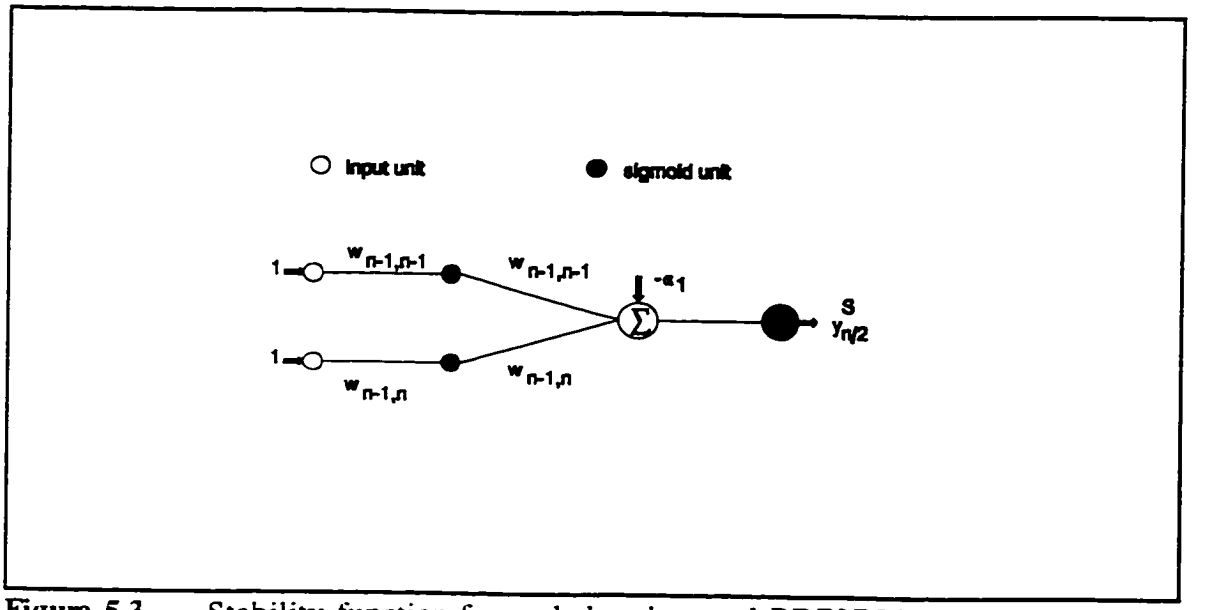


Figure 5.3 Stability function for scaled orthogonal BDRNN implemented as a constrained three layer feedforward neural network.

For a BDRNN with freeform block-diagonal matrix, the global and local stability functions can similarly be transformed into suitable SFNN framework. Shown in Figure 5.4 is the implementation of the local stability functions (41) and (42) consisting of two parallel networks, one for each transformation. As seen from this figure, these networks are constrained feedforward networks with five and three layers, respectively, with connection weights set equal to the elements of the freeform W. The input u^s to the SFNN is the constant bias 1. The bias input at the output layer is $-\alpha_2$. The output z^s of the SFNN is a vector of the elements $y_{n/2}^{s1}$ and $y_{n/2}^{s2}$.

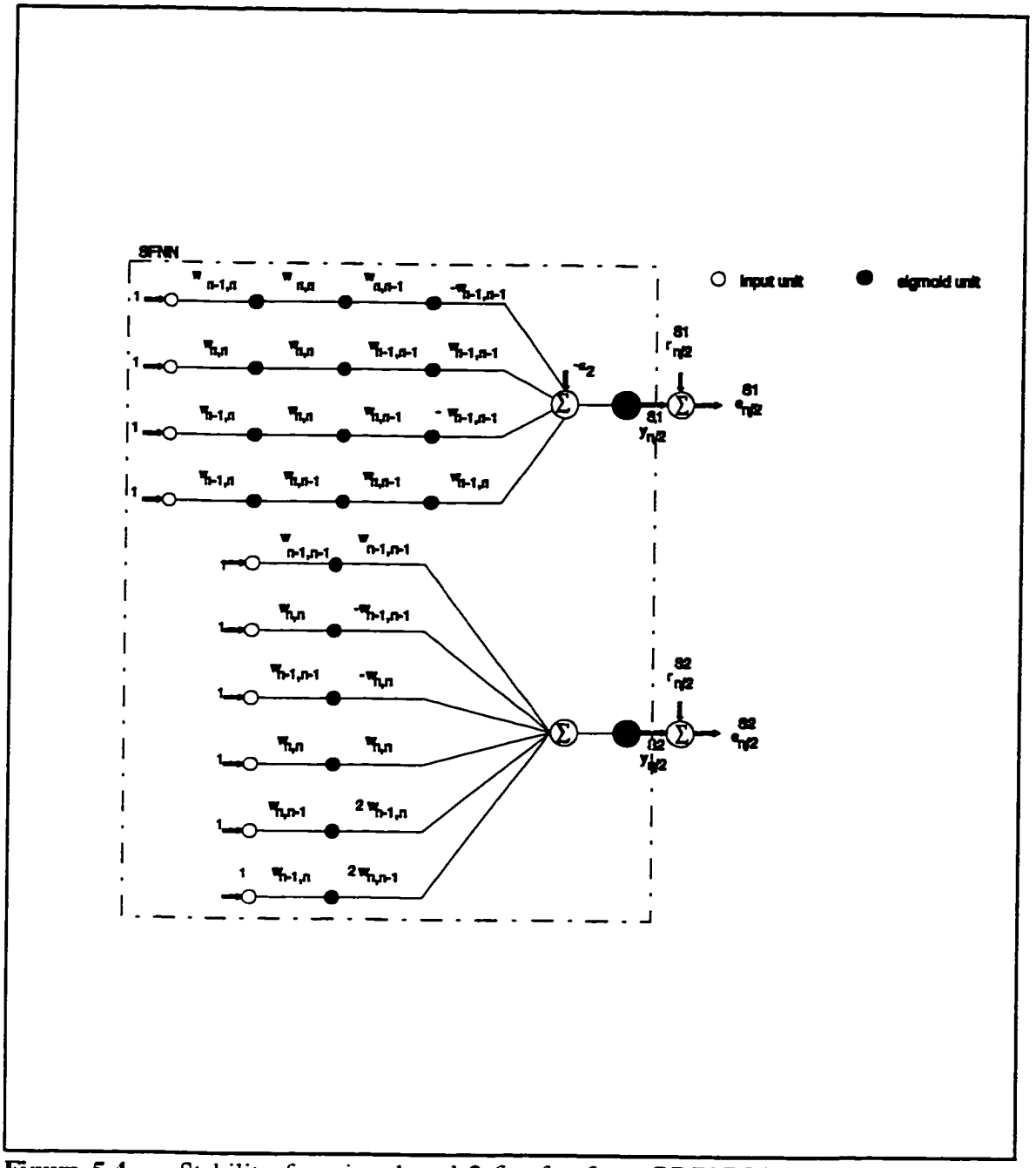


Figure 5.4 Stability function 1 and 2 for freeform BDRNN implemented as constrained five and three layer feedforward neural networks.

Table 5.1 **Summary of the CFNNs that implement the invertibility condition for the scaled orthogonal and freeform BDRNNs.**

BDRNN - invertibility conditions	constraining feedforward neural network that implements the invertibility conditions.
SCALED ORTHOGONAL BDRNN $w_{min}^{\delta} \le w_{n-1,n-1}^2 + w_{n-1,n}^2$ $n = 2,4,...,N$	$y_{n/2}^{\epsilon} = f_d[w_{min}^{\delta} - (w_{n-1,n-1} f_d(w_{n-1,n-1}) + w_{n-1,n} f_d(w_{n-1,n}))]$ $n=2,4,..,N$
FREEFORM BDRNN $w_{min}^{\delta} \le$ $\lvert w_{n-1,n-1} w_{n,n} - w_{n-1,n} w_{n,n-1} \rvert$	$y_{n/2}^{\epsilon 2} = f_d\{(w_{min}^{\delta})^2 - [w_{n-1,n-1} f(w_{n,n} f(w_{n-1,n-1} f(w_{n,n}))) + 2w_{n-1,n-1} f(w_{n,n} f(w_{n-1,n} f(w_{n,n-1}))) - w_{n-1,n} f(w_{n,n-1} f(w_{n-1,n} f(w_{n,n-1})))]\}$ $n=2,4,..,N$

Table 5.2 **Summary of the SFNNs that implement the global/local stability condition, for the scaled orthogonal BDRNN.**

SCALED ORTHOGONAL BDRNN Global and local stability condition	constraining feedforward neural network that implements the global/local stability conditions.
$w_{n-1,n-1}^2 + w_{n-1,n}^2 \le 1.0$ $n=2,4,..,N$	$y_{n/2}^{S} = f_d[w_{n-1,n-1} f_c(w_{n-1,n-1}) + w_{n-1,n} f_c(w_{n-1,n}) - \alpha_1]$, $0 < \alpha_1 < 1.0$;

Table 5.3 **Summary of the SFNNs that implement the global and local stability conditions for the freeform BDRNN.**

FREEFORM BDRNN global and local stability conditions	constraining feedforward neural network that implements these conditions.
Global stability condition $[(w_{n-1,n-1}^2+w_{n,n-1}^2+ w_{n-1,n}^2+w_{n,n}^2) -(w_{n-1,n-1}w_{n,n}-w_{n,n-1}w_{n-1,n})^2] \le 1$	$y^{ss}=f_d\{[w_{n-1,n-1}f(w_{n-1,n-1})+ w_{n,n-1}f(w_{n,n-1})+w_{n-1,n}f(w_{n-1,n})+ w_{n,n}f(w_{n,n})] - [w_{n-1,n-1}f(w_{n,n}f(w_{n-1,n-1}f(w_{n,n}))) - 2w_{n-1,n-1}f(w_{n,n}f(w_{n,n-1}f(w_{n-1,n}))) + w_{n,n-1}f(w_{n-1,n}f(w_{n,n-1}f(w_{n-1,n})))] -1\}$
Local stability condition 1 $\lvert w_{n-1,n-1}w_{n,n}-w_{n-1,n}w_{n,n-1}\rvert \le 1.0$ Local stability condition 2 $(w_{n-1,n-1}^2 - 2w_{n-1,n-1}w_{n,n} + w_{n,n}^2 + 4w_{n-1,n}w_{n,n-1}) \le 0.0$	$y_{n/2}^{s1}=$ $f_d\{[w_{n-1,n-1}f(w_{n,n}f(w_{n-1,n-1}f(w_{n,n}))) - 2w_{n-1,n-1}f(w_{n,n}f(w_{n,n-1}f(w_{n-1,n}))) + w_{n,n-1}f(w_{n-1,n}f(w_{n,n-1}f(w_{n-1,n})))] -1.0\}$ $n=2,4,..,N$ $y_{n/2}^{s2} = f_d[w_{n-1,n-1}f(w_{n-1,n-1}) -2w_{n-1,n-1}f(w_{n,n})+w_{n,n}f(w_{n,n}) +4w_{n-1,n}f(w_{n,n-1})]$ $n=2,4,..,N$

5.5.1 Network stability and stability of learning in FF-BDRNN with stabilizing feedforward neural networks (SFNN)

It can be seen that the architecture of Figure 5.2 has a framework with the FF-BDRNN forming the inner core and the SFNN the outer layer. The FF-BDRNN training is performed by accumulating the instantaneous output error gradient over a given training epoch during which W is held constant. The SFNN, on the other hand, computes the stability error gradient between epochs. W is updated at the end of each epoch using both the accumulated output error gradient and the stability error gradient. It is clear from (7) of Chapter 4, that the SFNN does not affect the dynamics of learning during an epoch, as W is held constant during each epoch. The role SFNN plays is to "direct" W towards a stable configuration between epochs and hence improve stability. Even with SFNN, it is still possible that the learning becomes unstable, particularly when the initial W chosen is highly unstable and the SFNN learning rate is so small in relation to the output error learning rate that the stabilization effect is inadequate. However, in the worst case, the FF-BDRNN with the SFNN can perform only as badly as the FF-BDRNN without the SFNN.

5.6 CONCLUSION

Conditions for local and global stability for two specific architectures of the block-diagonal feedback matrix that differ from each other in terms of complexity and degree of freedom are derived. The scaled orthogonal BDRNN in which the global and local stability constraints are equivalent, unlike the BDRNN with freeform submatrices, does not impose additional restrictions on weight assignments for satisfying the global stability constraints. Also, in this chapter, global and local stability functions for these two architectures are derived. The block diagonal structure of the BDRNN is exploited to find conditions that guide the network towards stability at each weight update during training.

This problem is addressed in two steps: first, by devising a cost function that includes the desired stability margin as one of its components; next, by ingraining the stability margin component as part of the neural network architecture. To quantify the stability of the FF-BDRNN at each weight update during the training process, a stability function that is a measure of the norm of the feedback weight matrix W is formulated. To ensure the stability of the FF-BDRNN and hence of its training process, the cost function to be minimized during the training process includes a penalty term which is a function of the stability function. For analytical tractability of the stability margin component of the cost function, some constraints are imposed on the values that the block diagonal weights can assume. It is shown that, under these non-restricting conditions, the stability function itself can be formulated as a multi-layer feedforward neural network using first order perceptrons which augments the BDRNN structure and hence ingrained in the FF-BDRNN architecture.

6 SIMULATION STUDIES ON FF-BDRNN WITH STABILIZATION

6.1 INTRODUCTION

In this chapter several examples are presented to demonstrate the feasibility of the FF-BDRNN architecture proposed in Chapter 3 to model a wide range of nonlinear processes, to illustrate the performance of the recalculated state variable modified BPTT learning algorithm proposed in Chapter 4 and to show the effectiveness of the stabilization technique presented in Chapter 5 for the scaled orthogonal and freeform BDRNN structures.

The first example, presented in section 6.2, illustrates the importance of maintaining the stability of the feedback weight matrix for successful learning, by training it without the stabilization technique presented in Chapter 5 and studying the effect of an unstable or a stable initial weight matrix on learning. This issue is further explored in the example in section 6.3. The examples in sections 6.3 and 6.4 compare the performance of scaled orthogonal and freeform BDRNN in modelling the dynamics of a plant with block-diagonal feedback structure. In the example of section 6.3, the plant output is a weakly nonlinear combination of the plant state variables, while, in the example of section 6.4 this nonlinearity of the plant is more pronounced. Also explored in the example of section 6.3, is the movement of the magnitude and phase of the eigenvalues of W^n as the training progresses, with and without stabilization, in comparison with the plant eigenvalue placement. In the example of section 6.5, a plant with a fully recurrent DTRNN architecture is modelled with scaled orthogonal and freeform BDRNN and the issue of global versus local stabilization is explored and tradeoffs discussed. In section 6.6-6.9, examples studied previously in literature with well documented results are considered. In the example of section 6.6, a nonlinear single-input driven single-output plant is considered for modelling with a BDRNN plant. The example of section 6.7, considers a nonlinear multiple-input driven multiple-output plant in which the dependence of the

current plant outputs on previous plant inputs and outputs is separable, is modelled with an FF-BDRNN. In the example of section 6.8, a BDRNN is used to reproduce the "figure 8" limit cycle without any inputs from the plant generating the limit cycle; this example poses special challenges due to the conflicting requirements of stable learning of marginally unstable autonomous plant dynamics. The example of section 6.9, consists of training a FF-BDRNN to predict the evolution of the chaotic process generated by a classic delay-differential equation; the numerical stability performance of the recomputed state vector modified BPTT algorithm is also studied. Finally, in section 6.10, an isolated word speech recognition problem is considered in which several FF-BDRNN modules are used for correctly classifying the utterance of numerals "zero"-"nine" using speech prediction techniques. The network and training parameters for the examples in section 6.2-6.10 are summarized in Table 6.4.

For these examples, the normalized root mean square error (NRMSE), ξ_h

$$\xi_h = \sqrt{\frac{E[(y_h^p(k) - y_h^n(k))^2]}{E[(y_h^p(k) - E[y_h^p(k)])^2]}}, \quad h = 1,..,N_o, \; k = 1,..,K_p \tag{1}$$

is used as a measure to assess the performance of the BDRNN/FF-BDRNN networks modelling a given plant. In each example, unless otherwise mentioned, the initial weights of the network feedback weight matrix W^n are assigned random numbers chosen such that the eigenvalues of its 2 x 2 submatrices are placed on the unit circle. The weight assignment is intended to fully "excite" the oscillatory modes of the network so as to enable the network to effectively model the plant dynamics. The initial weights of the input matrix B^n and the output matrix C^n are also assigned random values in the interval [-1.0,1.0]. In examples 6.7, 6.9 and 6.10 where the FF-BDRNN was used to model given plants, the weights of the feedforward network are assigned random values in the interval [-0.5.0.5]. The shadow output matrix C^s connecting the state vector $x^f(k)$ and $x^r(k)$ to the shadow output scalar $y^f(k)$ and $y^r(k)$, respectively, (which monitors the numerical stability

of state vector recomputations in the backward pass) is chosen as [1,1...,1] to facilitate full observability of all the state variables. The numerical instability threshold ϱ is chosen as (2.5E-3 N).

6.2 IMPORTANCE OF NETWORK STABILITY

This example provides simulation results to illustrate the feasibility of modelling a given dynamic process with a stable feedforward block-diagonal recurrent neural network. Let us consider a plant with a block-diagonal feedback structure. Let us also assume that the order of the plant is known *apriori* and set the order and structure of the FF-BDRNN the same as that of the plant.

The structures of the plant and the FF-BDRNN modelling the plant are shown in Figure 6.1. The plant uses a scalar input and generates a scalar output using a fourth order block diagonal state feedback matrix and a two-layer feedforward network. The output scalar generated by the plant at instant k is applied as external input to the FF-BDRNN. The one-step output estimate provided by the FF-BDRNN is compared with the plant output at instant $k+1$ to calculate the error. Several plant output vectors are generated as training data by setting the initial state variables of the plant to random values. In the training phase, the error is accumulated over the entirety of each training pattern and the FF-BDRNN weights are updated using the training algorithm outlined in Chapter 4. As the structure of the FF-BDRNN is chosen the same as that of the plant, if the training is successful, one would expect that the output error asymptotically reduces to a small value.

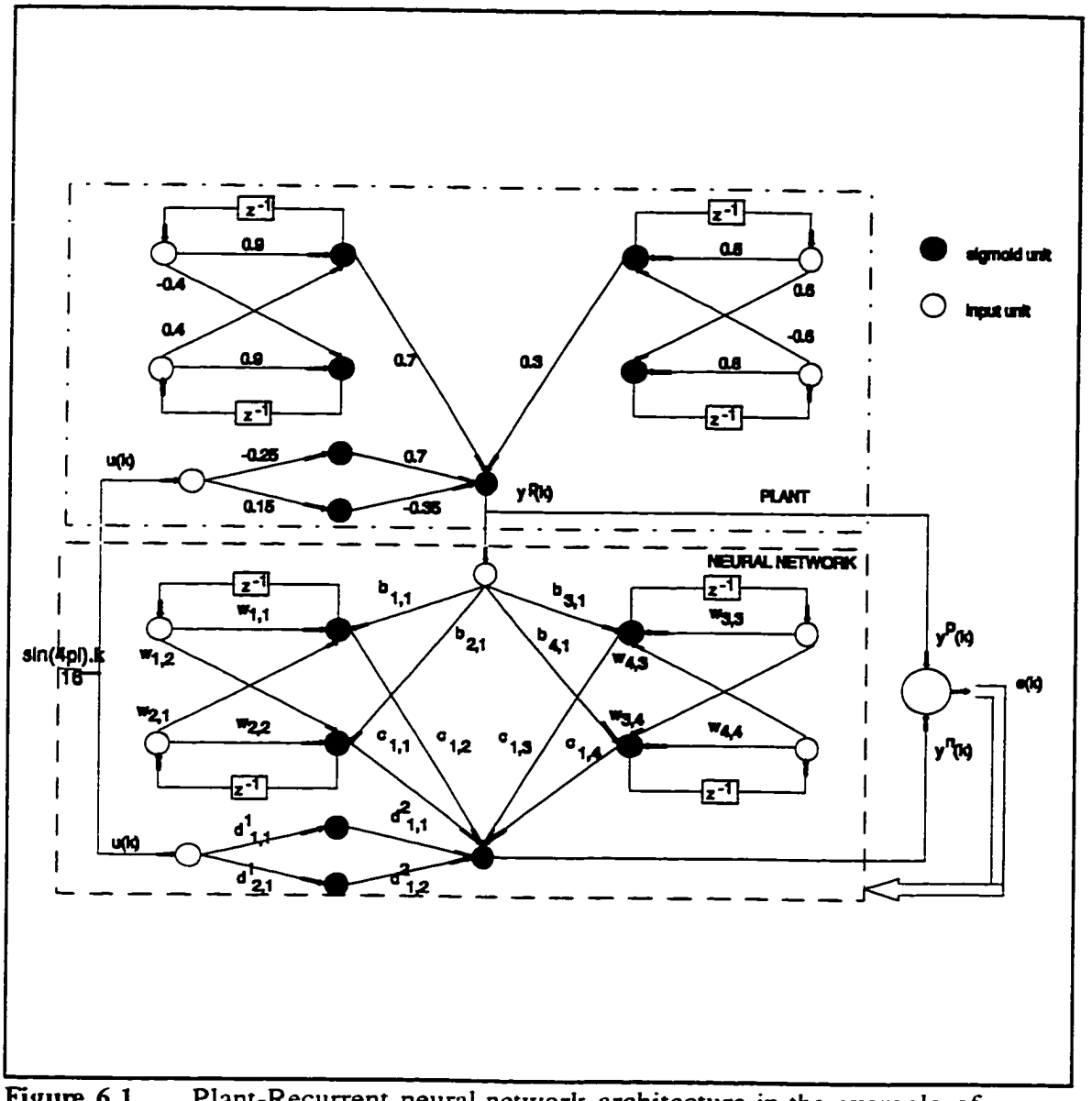


Figure 6.1 Plant-Recurrent neural network architecture in the example of Section 6.2

In training the FF-BDRNN, two cases are considered. In the first case, the initial values of the elements of the feedback weight matrix W^n are chosen to be small positive values so as to ensure that the eigenvalues of the initial $(W^n)^T W^n$ are well inside the unit disk and thus ensure the global stability of the initial W^n. In the second case, these initial values are chosen such that the eigenvalues of the initial $(W^n)^T W^n$ lie outside the unit disk and is initially unstable. Shown in Figure 6.2 are the results of training the FF-BDRNN for these two cases. Plot (a) depicts the output error performance of the FF-BDRNN as the training progresses. Plots (b) and (c) of Figure 6.2 depict the tracking performance of the trained FF-BDRNN with initial globally stable and unstable W^n respectively, for arbitrary outputs generated by the plant. As can be seen from plot (a), the output error for the first case with the initial globally stable feedback weight matrix (with small positive initial values of the elements of $(W^n)^T W^n$ converges to a small value over the training iterations. Apparently, for this case, the global stability of the FF-BDRNN has been maintained through the training process which enabled successful learning. This is also evident from plot (b) which shows that an arbitrary output of the plant has been tracked by the FF-BDRNN with near perfection. On the other hand, for the second case with the initial globally unstable feedback weight matrix (with the initial values of the elements of $(W^n)^T W^n$ chosen such that its eigenvalues lie outside the unit disk), the output error has remained high over the training iterations. Apparently, for this case, the FF-BDRNN has remained unstable through the training process resulting in poor learning. This is evident from plot (c) which shows a poor tracking performance for an arbitrary plant output.

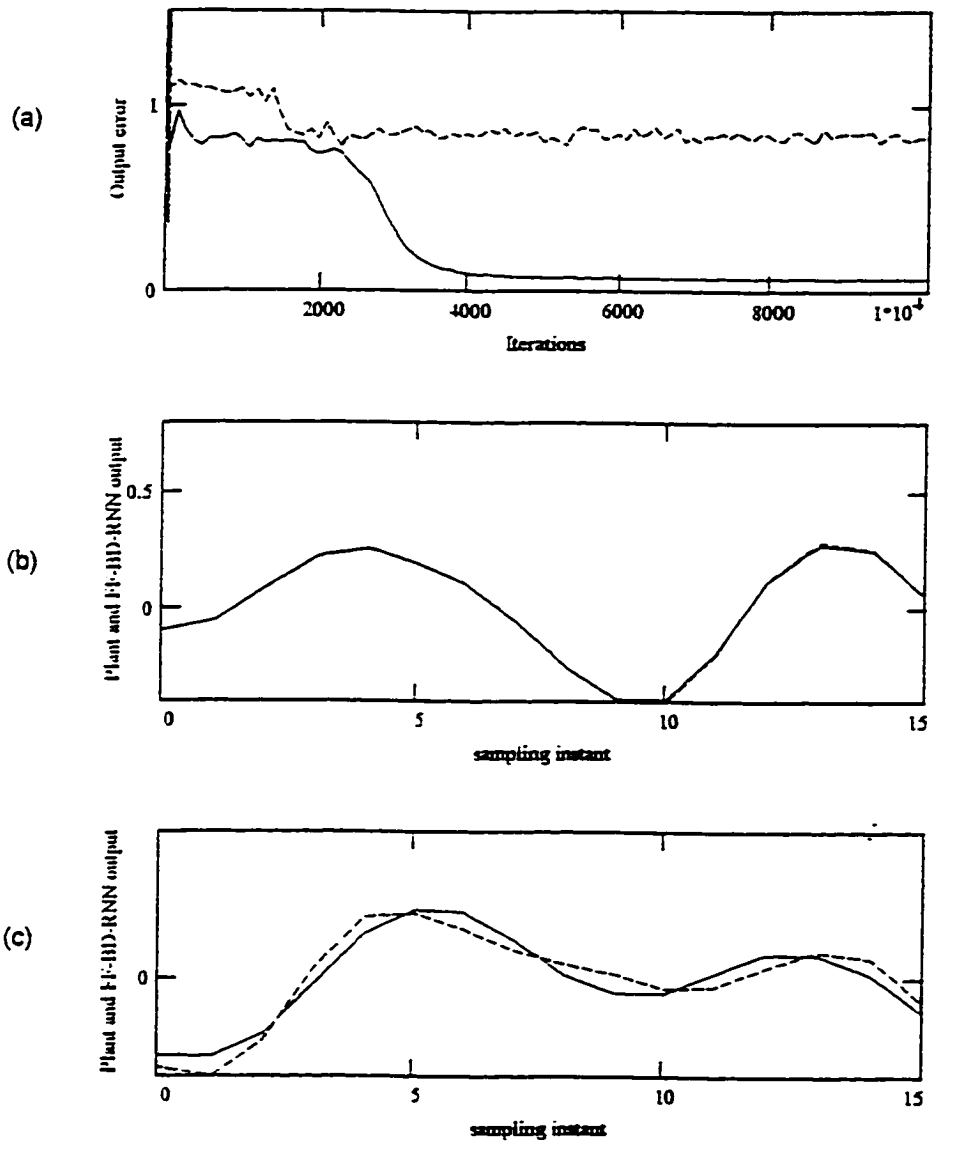


Figure 6.2. Training results of FF-BDRNN in Section 6.2 (a) with stable initial feedback weights - (——) BDRNN output compared with an (——) arbitrary plant output. (b) with feedback weights initialized to unstable values - (——) BDRNN output compared with an (-----) arbitrary plant output. (c) (——) with stable initial weights output error vs. training iterations. (——) with unstable initial weights output error vs. training iterations.

Possible explanations for the poor learning in this example, include factors such as (i) improper choice of learning rate μ_1, (ii) very small error gradient, and (iii) saturation effects of the sigmoidal function caused by improper scaling of the input data. It was not possible to improve learning performance of the second case by changing μ_1. However, these factors may not be solely responsible for the poor learning exhibited by the second case, since the conditions under which the simulation is performed are identical to those of the first case. This suggest that the initial choice of an unstable W^n, and the continued instability of FF-BDRNN during training are significant factors that result in poor learning performance. This example underlines that, for successful learning, the stability of the recurrent neural network is an important factor for consideration.

6.3 MODELLING A WEAKLY NON-LINEAR AUTONOMOUS PLANT WITH A BLOCK-DIAGONAL FEEDBACK STRUCTURE USING SCALED ORTHOGONAL AND FREEFORM BDRNN

Let us first consider an autonomous plant with a block-diagonal feedback structure described by the following:

$$\begin{aligned} x_i^p(k+1) &= f_2\left(\sum_{j=v}^{v+1} w_{ij}^p\, x_j^p(k)\right), \qquad i=1,\ldots,N \\ &\textit{where} \quad v=i, \qquad \textit{if } i \textit{ is odd} \\ &\qquad\quad v=i-1, \qquad \textit{if } i \textit{ is even} \\ y_h^p(k) &= f_2\left(\sum_{j=1}^{N} c_{hj}^p\, x_j^p(k)\right), \qquad h=1,\ldots,N_o \end{aligned} \tag{2}$$

where $x_i^p(k)$ is the i-th state variable of the plant at instant k, $\{w_{ij}^p\} = W^p$ is the block-diagonal state-feedback weight matrix of the plant. $f_2(.)$ is a sigmoidal squashing function defined in equation (8) of Chapter 3. $y_i^p(k)$ is the i-th output element at instant k, $\{c_{ij}^p\} = C^p$ are the weights connecting the N plant state variables to the N_o output units.

For this example, the number of plant state variables is $N = 4$, with the number of plant outputs $N_o = 1$. The number of sampling instants, K_p, for each pattern, was chosen to be 32. The plant is an oscillator with weights given by:

$$W^p = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & 0 & \sin(\theta_2) & \cos(\theta_2) \end{bmatrix} \tag{3}$$

$$\textit{where } \theta_1 = \left(\frac{2\pi}{K_p}\right); \quad \theta_2 = 3\theta_1; \quad K_p = 32$$

$$C^p = [\ 0.70 \quad 0.0 \quad 0.30 \quad 0.0\ ]$$

W^p models a fundamental and a third harmonic waveform in the sense of a linear system. The objective is to model the above plant, using one-step prediction, with a BDRNN described by the following:

$$\begin{aligned} x_i^n(k+1) &= f_2\left(\sum_{j=v}^{v+1} w_{ij}^n\, x_j^n(k) + \sum_{j=1}^{N} b_{ij}^n\, y_j^p(k)\right), \quad i=1,\ldots,N \\ &\textit{where} \quad v=i, \quad \textit{if } i \textit{ is odd} \\ &\qquad\quad v=i-1, \quad \textit{if } i \textit{ is even} \\ y_h^n(k) &= f_2\left(\sum_{j=1}^{N} c_{hj}^n\, x_j^n(k)\right), \qquad h=1,\ldots,N_o \end{aligned} \tag{4}$$

where $x_i^n(k)$ is the i-th state variable of the network at instant k, $\{w_{ij}^n\} = W^n$ is the state-feedback weight matrix of the network, $\{b_{ij}^n\} = B^n$ is the external input weight matrix of the network, $y_i^n(k)$ is the i-th output element at instant k, $\{c_{ij}^n\} = C^n$ is the weight matrix connecting the N state variables to the N_o output units of the network. The order of the plant is assumed to be known *apriori* and the order of the BDRNN is set the same as that of the plant. The output vector generated by the plant $y_j^p(k)$ at instant k is applied as external input to the BDRNN. The one-step output estimate provided by the BDRNN $y_i^n(k+1)$ is compared with the output of the plant $y_i^p(k+1)$ to calculate the error $e_i(k+1)$. In the training phase of the BDRNN, the error magnitude $|e_i(k)|$ accumulated over the entire pattern is used to update the weight parameters W^n, B^n and C^n after each

presentation of the output pattern. Several plant output vectors are generated by setting the initial state variables of the plant to random values so as to enable the network to model the dynamic modes of the plant irrespective of the initial values of the plant state variables. The initial state variables of the BDRNN are set to random values for each presentation of the plant output pattern. It is expected that the output error at initial sampling instants will be impacted by the choice of initial conditions and hence larger than the error at later sampling instants. The output error minimization at later sampling instants is weighted more than at earlier instants for each pattern by modifying the output error function (20) of Chapter 3 with an exponential decay as:

$$J_t = \frac{1}{2}\Sigma_{k=1}^{K_p}\left[\Sigma_{j=1}^{N_o}\left(r_j(k)-y_j(k)\right)^2\right]\left(1 - e^{-kc_s}\right) \qquad (5)$$

where c_s is a positive scaling constant.

The initial weights of the BDRNN were chosen such that the eigenvalues of the feedback matrix W^n are outside the unit circle. This represents a possible situation during the training phase where the BDRNN is unstable. A total of 10,000 training patterns were presented, each generated by setting the initial state variables of the plant at random. The learning rate μ_1 is chosen arbitrarily as 0.025. The training is performed with the weighted output error function (5) with $c_s = 0.0625$.

The plant described by (2) and (3) is now modelled by the two BDRNN structures, viz., the scaled orthogonal and freeform BDRNN introduced in Chapter 3.

6.3.1 Modelling with scaled orthogonal BDRNN with and without global stabilization

Shown in Figure 6.3.1 (a) to (f) are the results of training a scaled orthogonal BDRNN with no stability compensation compared to the corresponding results with global stability compensation. In Figure 6.3.1. (a) and (b) show the migration of the magnitude and the

phase, respectively, of only one eigenvalue of each complex conjugate pair of W^n with stabilization as compared to the corresponding results without stabilization, (c) and (d) are the plots of the output and the stability error, respectively, over the training iterations with stabilization as compared to the corresponding result without stabilization; (e) and (f) are the plots of the one-step output prediction of the scaled orthogonal BDRNN, with and without stabilization, respectively, compared to an arbitrary plant output. For the given choice of initial weights, the behaviour of the scaled orthogonal BDRNN without stability compensation, is in general observed to be very poor. From Figure 6.3.1(a) it is seen that both the eigenvalues of W^n hover in the unstable region (their magnitude is greater than 1.0). This is also reflected by a large stability error in Figure 6.3.1(d), which shows little improvement with training iterations or with reduction in the learning rate μ_1. It appears as though that the network has found a local minima from which it has little incentive to climb out, consequently resulting in a large near constant output error as seen in Figure 6.3.1(c). From Figure 6.3.1(e), which depicts the tracking performance of BDRNNs with unstable W^n, for an arbitrary output generated by the plant, it can be seen that the BDRNNs without stabilization have not been successful in modelling the plant. On the other hand, with stability compensation, the eigenvalues of W^n migrate into the stable region (their magnitude is less than 1.0) as seen in Figure 6.3.1(a), resulting in a very small stability error as seen in Figure 6.3.1(d). It is also seen that the network eigenvalues closely match the eigenvalues of the plant. From Figure 6.3.1(f), which depicts the tracking performance of BDRNNs with stable W^n, for an arbitrary output generated by the plant, it can be seen that the BDRNN with stabilization has been very successful in modelling the plant. This as also reflected by the low output error as seen in Figure 6.3.1(d).

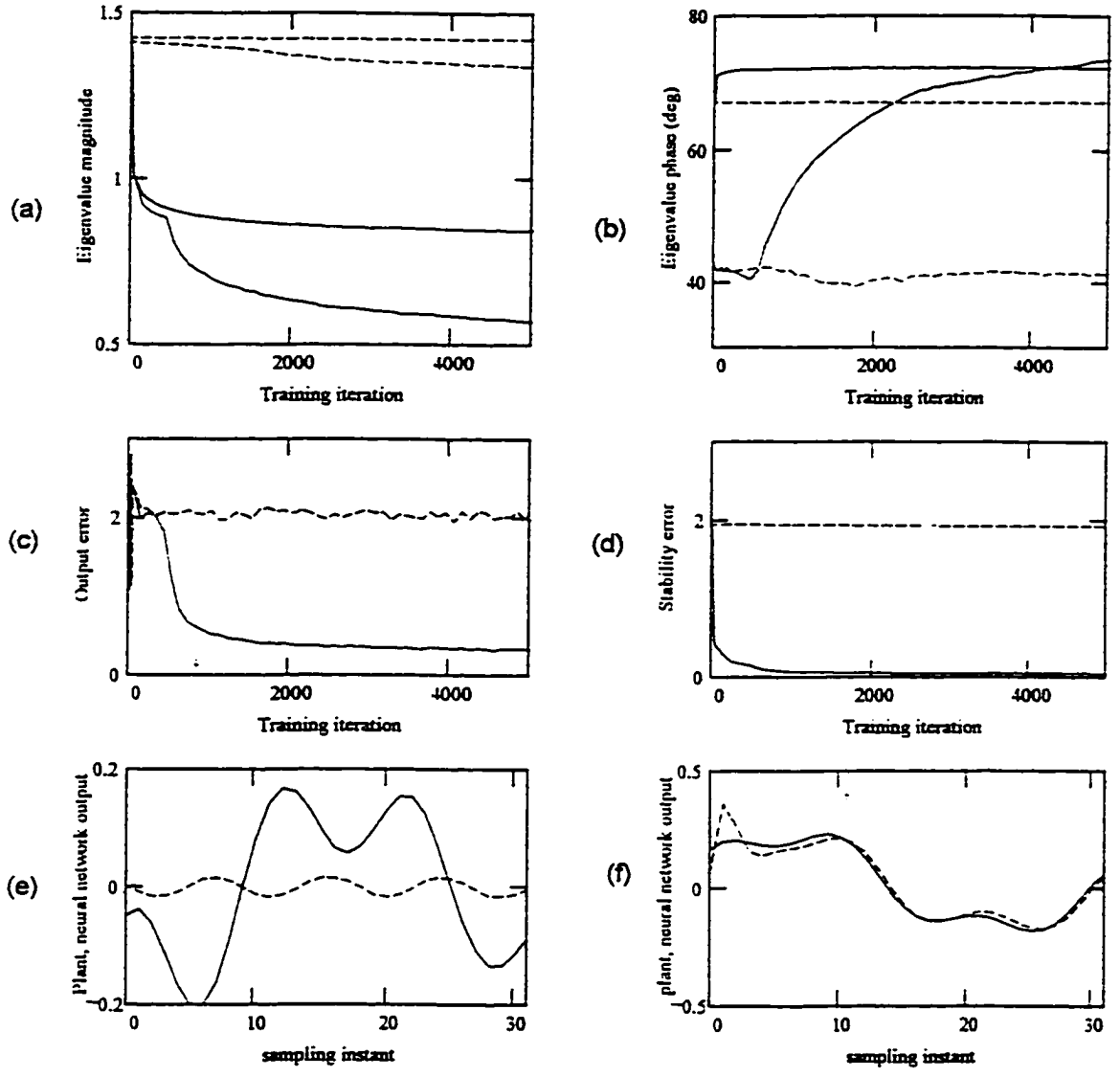


Figure 6.3.1 . Simulation results for **orthogonal** BDRNN in Section 6.3 with **no** stabilization (-----) and with **stabilization** (——): (a) Migration of the magnitude of the eigenvalues of W^n over training iterations, (b) Migration of the phase of eigenvalues of W^n over training iterations. (c) output error vs. training iterations. (d) stability error vs. training iterations. (e) output of BDRNN with **no stabilization** (-----) compared to an arbitrary plant output (——). (f) output of BDRNN with **stabilization** (-----) compared to an arbitrary plant output (——).

6.3.2 Modelling with Freeform BDRNN with and without global stabilization

Shown in Figure 6.3.2 (a) to (f) are the results of training a freeform BDRNN with no stability compensation compared to the corresponding results with stability compensation. The behaviour of the BDRNN with the freeform submatrix structure, with and without global stability compensation, is observed to be similar in terms of final eigenvalue placement, and output and stability errors to that of the corresponding cases of the BDRNN with the scaled orthogonal submatrix structure (Figure 6.3.1). In this example, for convergence, the freeform BDRNN with global stability compensation requires more than half the number of training iterations as the scaled orthogonal BDRNN with stability compensation. It appears that the freeform BDRNN in this case, spends more time, exploring several additional pathways (because of its larger degree of freedom), before settling on the solution.

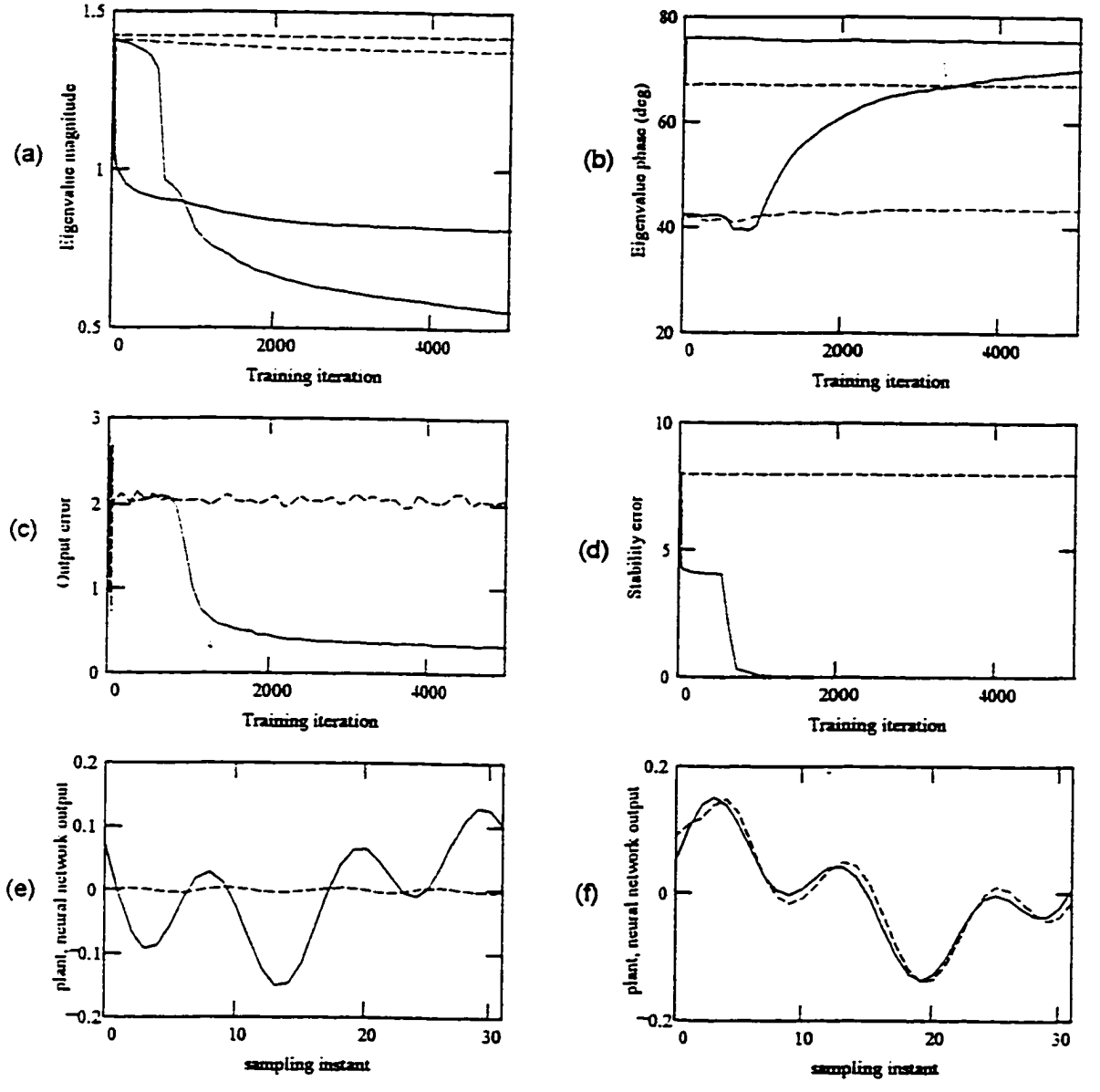


Figure 6.3.2 Simulation results for **freeform** BDRNN in Section 6.3 with **no** stabilization (-----) and with **stabilization** (——). (a) Migration of the magnitude of the eigenvalues of W^n over training iterations, (b) Migration of the phase of eigenvalues of W^n over training iterations. (c) output error vs. training iterations. (d) stability error vs. training iterations. (e) output of BDRNN with **no stabilization** (-----) compared to an arbitrary plant output (——). (f) output of BDRNN with **stabilization** (-----) compared to an arbitrary plant output (——).

6.4 MODELLING A NON-LINEAR PLANT WITH A BLOCK-DIAGONAL FEED BACK STRUCTURE USING SCALED ORTHOGONAL AND FREEFORM BDRNN

Let us now consider a plant with a higher degree of non-linearity when compared with the plants of the previous examples. The state equation for the fourth order non-linear plant is the same as the state equation in (2) with feedback weight matrix W_p given by equation (3) with K_p = 48 and θ_2 = $(5.\theta_1)$. However, the plant output is a non-linear combination of the state variables and is given by:

$$y^p(k) = \frac{x_1^p(k)}{1 + x_2^p(k)} - \frac{x_3^p(k)}{1 + x_4^p(k)} \qquad (6)$$

The objective is to model this plant, first by a fourth-order, and subsequently by an eighth-order BDRNN with scaled orthogonal and free-form submatrices. The training of the two classes of BDRNNs was performed with global stability compensation and with the following parameters: μ_1 = 0.025; weighted output error function (5) with c_s = 0.03125. Shown in Figure 6.4.1 are the results comparing the training of a fourth-order scaled orthogonal BDRNN with the corresponding results for the freeform structure. Shown in Figure 6.4.2 are the results comparing the training of a eighth-order scaled orthogonal BDRNN with the corresponding results for the freeform structure. In these figures, plots (a) and (b) depict the output error performance of the scaled orthogonal and freeform BDRNNs as training progresses; plots (c) and (d) depict the tracking performance of the scaled orthogonal and freeform BDRNNs, respectively, for arbitrary outputs generated by the plant.

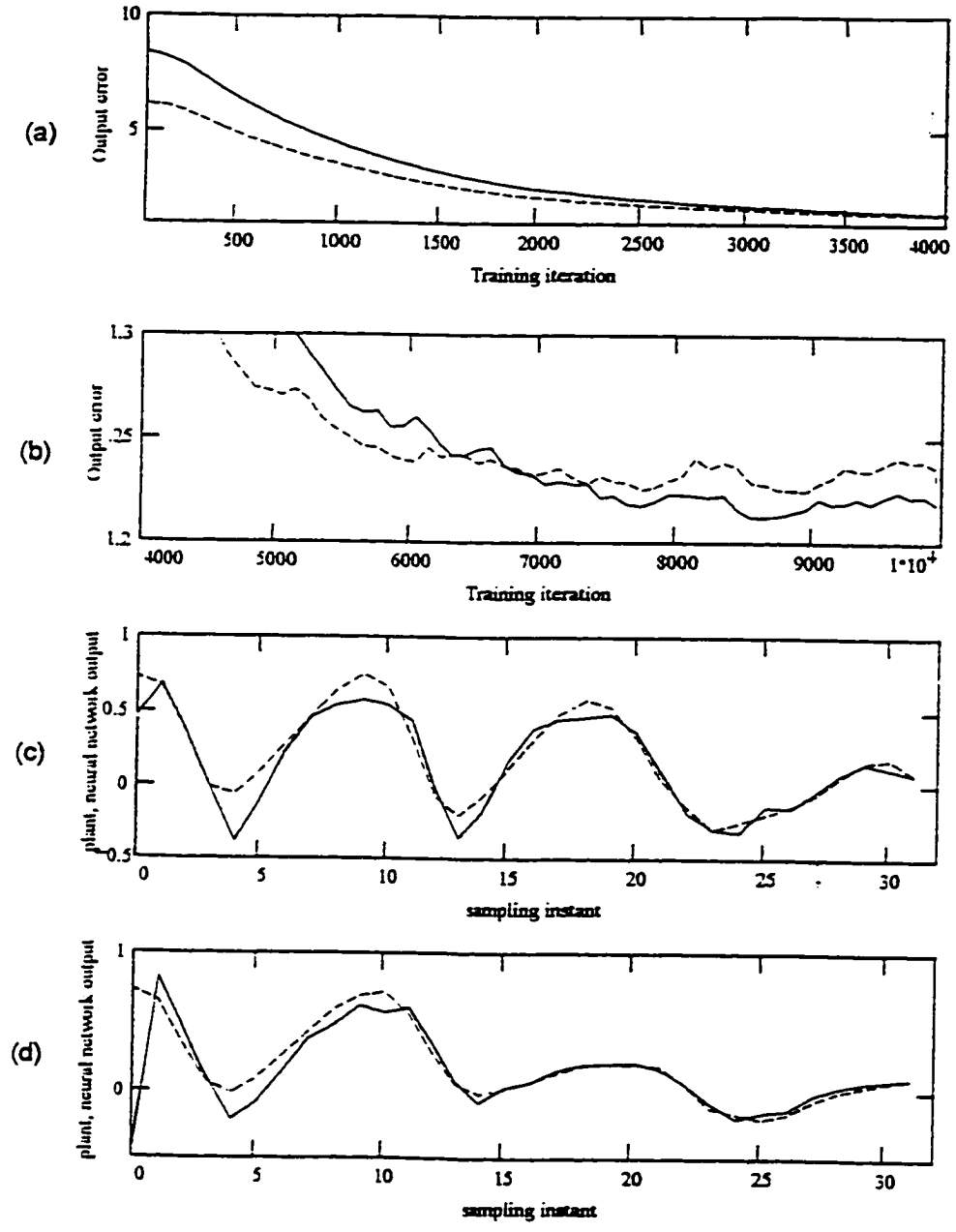


Figure 6.4.1 Simulation results for modelling a **non-linear** plant with a stabilized **fourth** order BDRNN in Section 6.4: (a) output error vs. training iterations for orthogonal (-----) and freeform (——) BDRNN upto 4000 iterations (b) output error vs. training iterations for orthogonal (-----) and freeform (——) BDRNN from 4000-10000 iterations (c) (-----) arbitrary plant output compared to (——) output of **orthogonal** BDRNN (d) (-----) arbitrary plant output compared to (——) output of **freeform** BDRNN

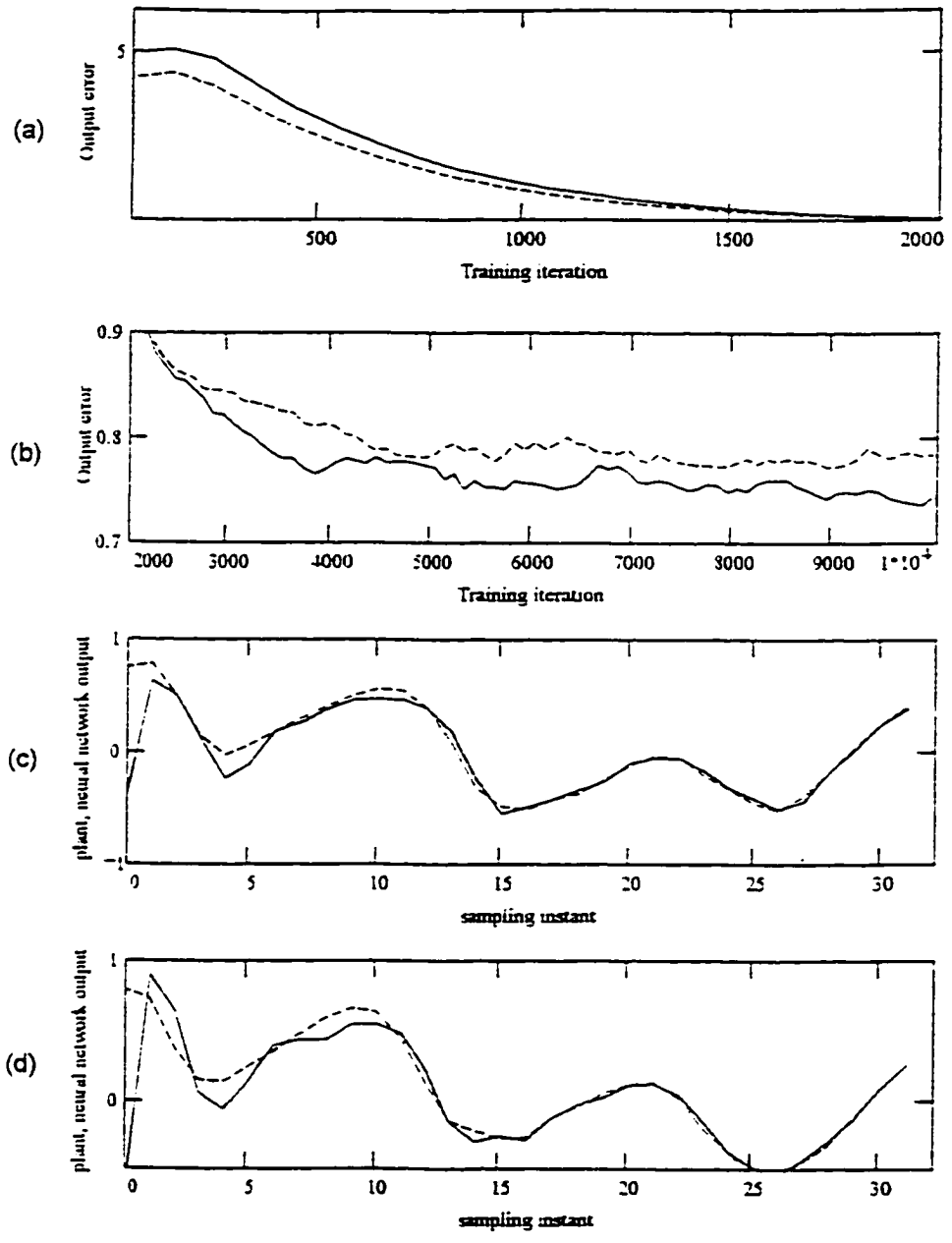


Figure 6.4.2 Simulation results for modelling a **non-linear** plant with a stabilized **eighth** order BDRNN, in Section 6.4: (a) output error vs. training iterations for orthogonal (-----) and freeform (——) BDRNN upto 4000 iterations (b) output error vs. training iterations for orthogonal (-----) and freeform (——) BDRNN from 4000-10000 iterations (c) (-----) arbitrary plant output compared to (——) output of **orthogonal** BDRNN (d) (- - -) arbitrary plant output compared to (——) output of **freeform** BDRNN

It is apparent from these figures that both the fourth and eighth order scaled orthogonal and freeform BDRNNs have been successful in tracking the plant output with small error. However, the eighth order BDRNN outperforms the fourth order BDRNN as is evident from the lower output error (plot (b) of Figures 6.4.2 and 6.4.1) and better tracking, in particular, at the initial sampling instants (plots (c) and (d) of Figures 6.4.2 and 6.4.1). The performance difference between the fourth and eighth order BDRNNs is obviously due to the fact that the eighth order BDRNN has a much higher degree of freedom to model the nonlinearity described in (6) more accurately than the fourth order BDRNN. A noteworthy observation is that, in BDRNNs of the same order, the freeform BDRNN converges to a lower output error than the scaled orthogonal BDRNN (plot (b) of Figures 6.4.1) and 6.4.2). This is also evident from the better tracking performance, in particular, at the initial sampling instants (plots (c) vs. (d) of Figures 6.4.1 and 6.4.2). This performance difference is due to the fact that the freeform BDRNN has a higher degree of freedom than the scaled orthogonal BDRNN to model the nonlinearity described in (6) more accurately.

6.5 MODELLING A NON-LINEAR FULLY RECURRENT DTRNN PLANT WITH SCALED ORTHOGONAL AND FREEFORM BDRNN

In this example, the performance of scaled orthogonal and freeform BDRNN for a one-step prediction problem for a plant with a DTRNN structure with a fully connected feedback weight matrix is considered. The plant is described by:

$$x_i^p(k+1) = f_2\left(\sum_{j=1}^{4} w_{ij}^p \, x_j^p(k)\right) \qquad i=1,..,4 \tag{7}$$

where $W^p=\{w^p_{i,j}\}$ is described by:

$$W^p = \begin{bmatrix} 1.024 & 0.221 & 0.193 & 0.008 \\ -0.076 & 0.951 & -0.196 & 0.01 \\ 0.121 & -0.242 & 0.714 & -0.463 \\ -0.224 & 0.356 & 1.075 & 1.059 \end{bmatrix} \tag{8}$$

The plant output is a nonlinear function of the state variables x^p_i described by:

$$y_1^p(k) = \frac{x_1^p(k)x_3^p(k)}{1 + (x_1^p(k))^2 + (x_2^p(k))^2 + (x_3^p(k))^2 + (x_4^p(k))^2} \tag{9}$$

The state equation (7) of the plant generates a periodic signal which can be decomposed into a fundamental and a fifth harmonic component in the linear sense (without the sigmoidal squashing) with a sampling rate of K_p = 48. The plant is modelled by an eighth-order BDRNN with scaled orthogonal and freeform submatrices. For freeform BDRNN both local and global stability constraints are considered. The training of the two classes of BDRNN has been performed with stability compensation and with the following parameters: μ_1 = 32E-3 with momentum = 16E-3, μ_2 = 25E-3, number of intermediate storage state-vectors = 5 and w^{δ}_{min} = 0.5 in (28) or (32) of Chapter 5, for scaled orthogonal or freeform BDRNN. The initial conditions for the state variables of the network were chosen to be random numbers in the interval [-1.0,1.0]. Shown in Figure 6.5(a) are the output errors of the BDRNNs as training progresses. Shown in Figure 6.5(b) and 6.5(c) are the tracking performance of the scaled orthogonal BDRNN and freeform BDRNN with local stabilization, respectively, for arbitrary output generated by (9).

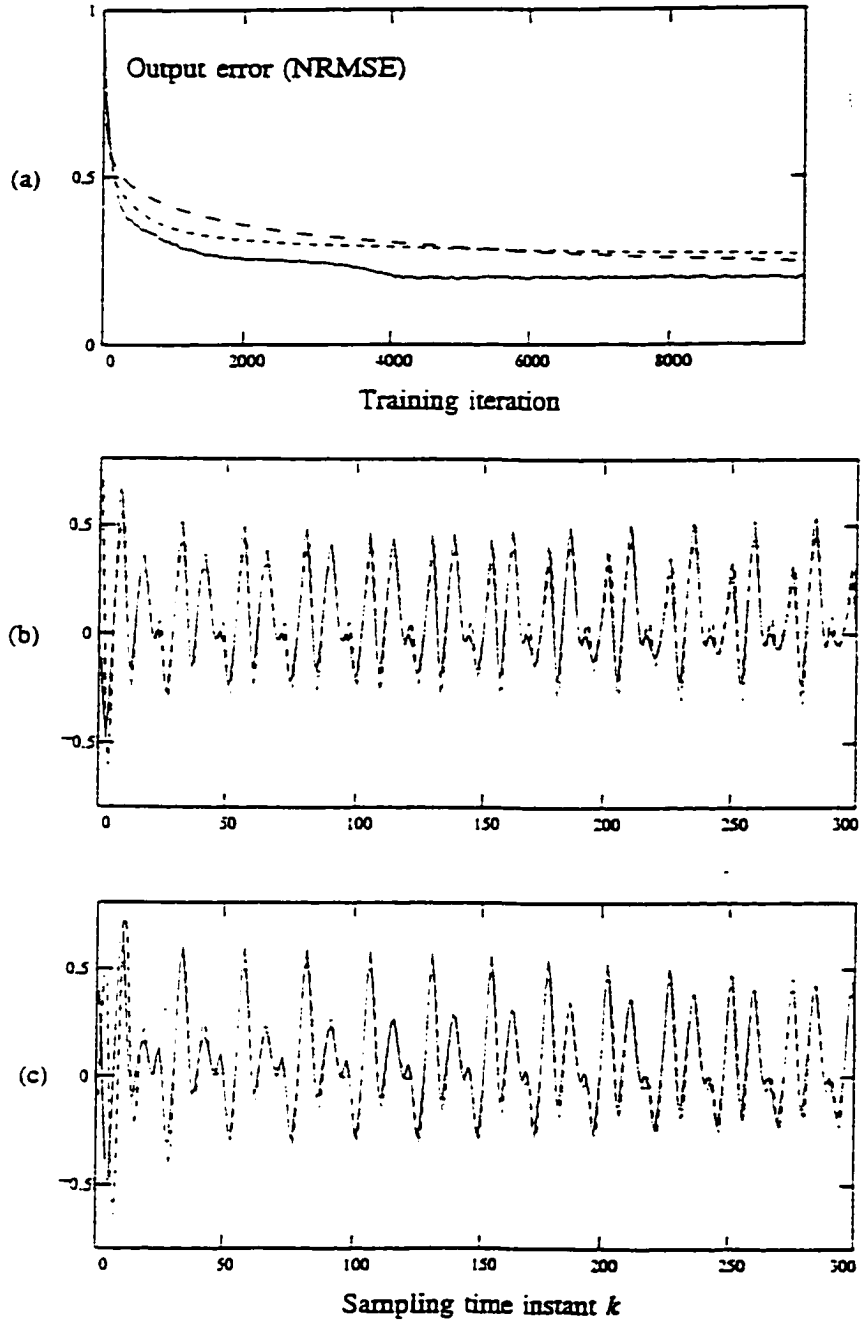


Figure 6.5 Simulation results for modelling a **non-linear** fully recurrent DTRNN plant with a stabilized **eighth** order BDRNN in Section 6.5: (a) output error vs. training iterations for (---) scaled orthogonal, freeform with (- - -) local and (—) global stability. (b) (---) arbitrary plant output compared to (—) output of scaled **orthogonal** BDRNN. (c) (---) arbitrary plant output compared to (—) output of freeform BDRNN with local stability.

As can be seen from Figure 6.5 (a), the freeform BDRNN with local stability compensation out-performs both scaled orthogonal BDRNN and freeform BDRNN with global stability compensation in terms of output error. This performance difference may be attributed to the fact that the freeform BDRNN with local stability compensation has a higher degree of freedom than the other two BDRNNs in modelling the nonlinearity described in (9). However, in several simulation runs for this and other examples, it has been observed that the learning performance of the freeform BDRNN with local stabilization is highly dependent on the choice of learning rate which should be kept small to ensure stable learning. On the other hand, the BDRNNs with global stabilization are more tolerant of higher learning rates.

6.6 MODELLING A NONLINEAR SINGLE-INPUT SINGLE-OUTPUT PLANT WITH A SCALED ORTHOGONAL BDRNN

In this example, a BDRNN is trained to identify a single-input single-output (SISO) plant studied in [Narendra and Parthasarathy 1990] whose output is a nonlinear function of the previous outputs and the previous inputs described by the difference equation:

$$x^p(k+1) = \frac{x^p(k)x^p(k-1)x^p(k-2)u^p(k-1)[x^p(k-2)-1.0]+u^p(k)}{1 + x^p(k-1)^2 + x^p(k-2)^2} \qquad (10)$$

A scaled orthogonal BDRNN with eight state variables, one input unit and one output unit is trained to model (10). The only input $u_1^n(k)$ to the BDRNN consists of the plant input $u^p(k)$ which is assigned random values in the interval [-1,1]. The initial conditions for the network state variables are chosen to be random numbers in the interval [-1.0,1.0]. The training is performed by updating the weights after every epoch of fifty time-steps. For this, μ_1 is chosen to be 4E-3 with a momentum of 2E-3 and μ_2 is chosen to be 32E-4. The number of intermediate storage state vectors to ensure numerical stability is chosen to be five at evenly spaced intervals of each epoch. The stability region is constrained to within

the unit disk by setting $\alpha_1 = 0.95$ in (44) of Chapter 5. The invertibility of W^n is ensured by setting w^{δ}_{min}=0.5 in (28) of Chapter 5. The network converged in 11,000 epochs of training. In Figure (6.6), the output of the plant and of the network are shown for an input $u^p(k) = sin\ (2\pi k/25) + sin\ (2\pi k/10)$. The NRMS output error for this input pattern was observed to be 1.56 at the start of training and was 0.1 at the end of training. Note that this example requires a scaled orthogonal BDRNN with eight state variables and one input compared to the fully connected three layer feedforward network with five input units, twenty units in the first hidden layer, ten units in the second hidden layer and one output unit considered in [Narendra and Parthasarathy 1990] required to model (10) with comparable results. With the scaled orthogonal BDRNN, the number of weights required is eight in the BDRNN layer and eight each in the input and output layers compared to about 300 weights required for the network reported in [Narendra and Parthasarathy 1990].

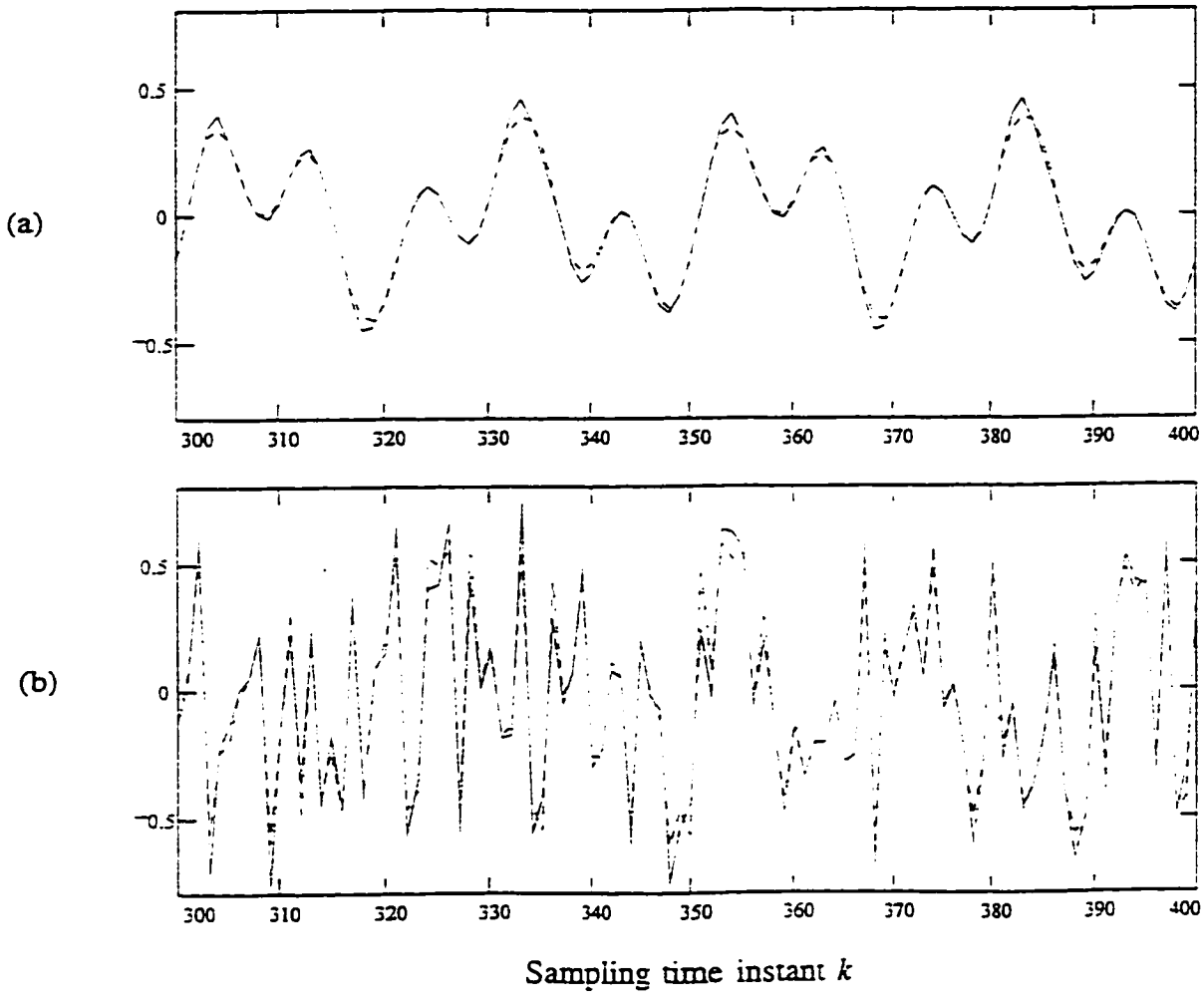


Figure 6.6 SISO Plant (——) and FF-BDRNN (-----) output trajectories in Section 6.6: (a) Output traces for input: $\sin(2\pi k/25) + \sin(2\pi k/10)$ (b) Output traces for random input.

6.7 MODELLING A NONLINEAR MULTIPLE INPUT MULTIPLE OUTPUT (MIMO) PLANT WITH A SCALED ORTHOGONAL FF-BDRNN

In this example, an FF-BDRNN is trained to identify a multiple-input multiple-output (MIMO) plant in which the non-linear dependence of the output on the previous outputs and the previous inputs is considered separable [Narendra and Parthasarathy 1990]. An example of such a plant is the two-input two-output plant given by the difference equations [Narendra and Parthasarathy 1990]:

$$\begin{bmatrix} x_1^p(k+1) \\ x_2^p(k+1) \end{bmatrix} = \begin{bmatrix} \dfrac{x_1^p(k)}{1 + [x_2^p(k)]^2} \\ \dfrac{x_1^p(k)x_2^p(k)}{1 + [x_2^p(k)]^2} \end{bmatrix} + \begin{bmatrix} u_1^p(k) \\ u_2^p(k) \end{bmatrix}, \qquad \begin{bmatrix} y_1^p(k) \\ y_2^p(k) \end{bmatrix} = \frac{1}{5}\begin{bmatrix} x_1^p(k) \\ x_2^p(k) \end{bmatrix} \tag{11}$$

A two input, two output BDRNN with 12 state variables is trained to model this plant. The inputs to the BDRNN consists of the plant input $u_1^p(k)$, $u_2^p(k)$ together with the plant outputs $y_1^p(k)$ and $y_2^p(k)$. The output of the BDRNN is the predicted value of $y_1^p(k+1)$ and $y_2^p(k+1)$. The plant inputs at every time step k are assigned random values in the interval [-2,2]. The initial conditions for the state variables of the network are chosen to be random numbers in the interval [-1.0,1.0]. The training is performed by updating the weights after each epoch of fifty time-steps. For this, μ_1 is set to 16E-3 with a momentum of 8E-3 and μ_2 to 12E-3. The number of intermediate storage state vectors, to ensure numerical stability, is chosen to be five at evenly spaced intervals of each epoch. The stability region is constrained to within the unit disk by setting α_l = 0.95 in (44) of Chapter 5. The invertibility of W^n is ensured by setting w^{δ}_{min}=0.5 in (28) of Chapter 5. The network converges in 100,000 epochs, with NRMS output error of less than 0.35. Increasing the order of the BDRNN network or changing it to freeform with global stability constraint and retraining it, does not significantly improve the performance of the network. Since, for this example, the dependence of the current output on the previous

outputs and the inputs is assumed to be separable [Narendra and Parthasarathy 1990], the BDRNN is augmented with a feedforward network (equations 17 and 18 of Chapter 3) that models the nonlinear mapping between the inputs and outputs of the plant. The feedforward section consists of one hidden layer with sixteen units. The resulting FF-BDRNN, when trained afresh converges in 100,000 epochs, with NRMS output error of less than 0.15. In Figure 6.7, the response of the plant and the FF-BDRNN to a input vector $[1/2\{sin\ (2\pi k/25) + sin\ (2\pi k/10)\},\ 1/2cos\ (2\pi k/25)]^T$ is shown. Note that this performance is comparable to those obtained in [Narendra and Parthasarathy 1990, Parisi et al].

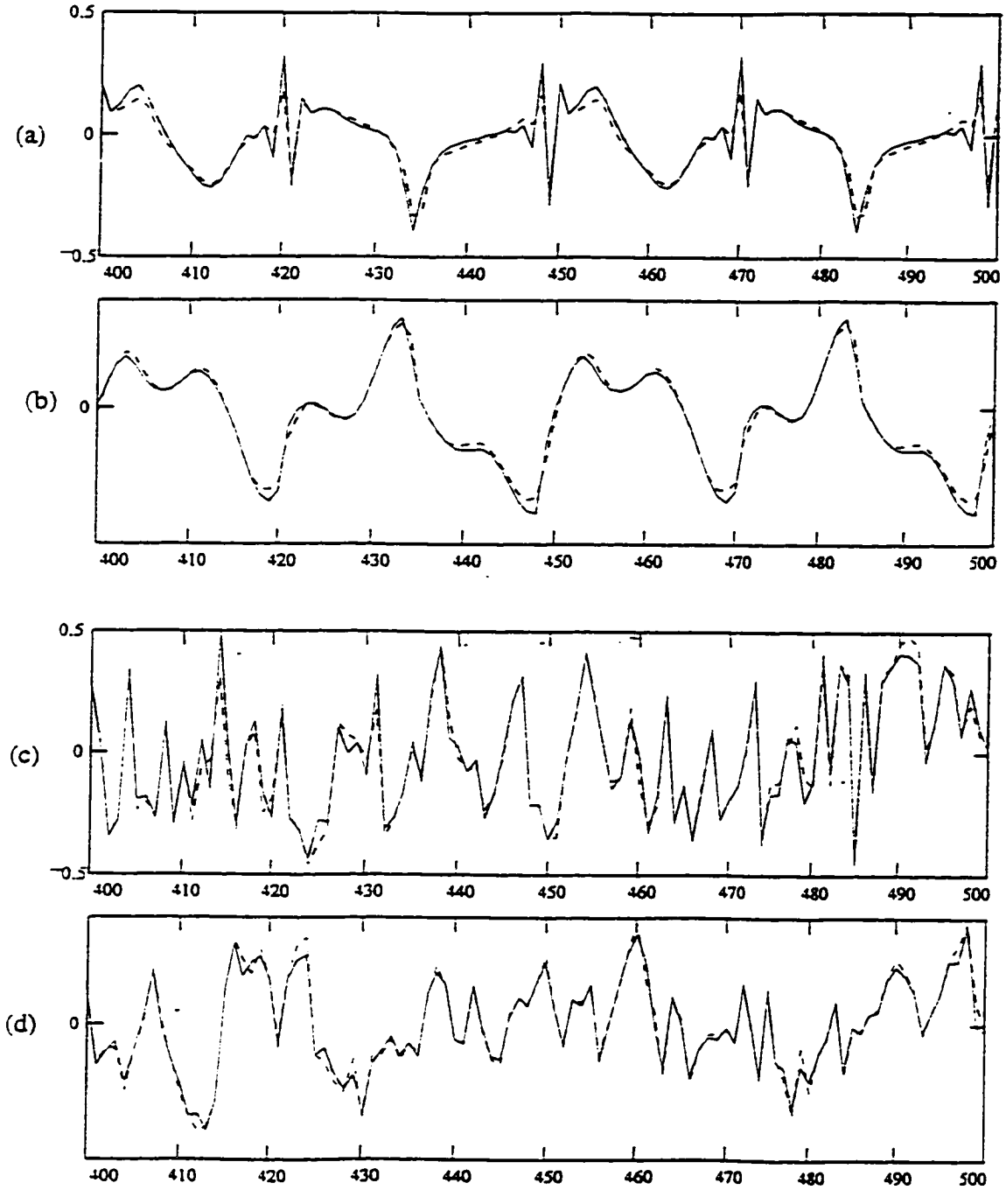


Figure 6.7. MIMO Plant (——) and FF-BDRNN (- - - -) output trajectories in Section 6.7: (a) First output traces for input: [1/2{sin(2πk/25) + sin(2πk/10)}, 1/2 cos (2πk/25)]^T. (b) Second output traces for input: [1/2{sin(2πk/25) + sin(2πk/10)}, 1/2 cos (2πk/25)]^T. (c) First output traces for random input. (d) Second output traces for random input.

6.8 MODELLING A MARGINALLY UNSTABLE PLANT THAT PRODUCES "FIGURE 8" LIMIT CYCLES USING A BDRNN

An autonomous BDRNN network with no input units is trained to reproduce a "figure 8" limit cycle studied in [Pearlmutter 1989, Fang and Sejnowski]. Since the network has no inputs, it is expected that the network must be marginally unstable if its outputs are to move through limit cycles. Given the stability focus of this thesis, this example provides a suitable benchmark to study the conflicting requirement of stable learning versus the reproduction of the output of a marginally unstable plant. The training is expected to be slow and difficult for the following reasons:

i Initially, the network has no specific information on the error at each time step, but has a total consolidated error over the entire length of the trajectory, which could result in slow convergence.

ii Second, as the network has to emulate a marginally unstable plant, the training could become unstable unless suitable measures are taken to ensure stability.

The "figure 8" trajectory is generated by the autonomous plant given by:

$$\begin{bmatrix} y_1^p \\ y_2^p \end{bmatrix} = 0.4 \begin{bmatrix} \sin\frac{2\pi t}{K_p} \\ -\cos\frac{4\pi t}{K_p} \end{bmatrix} \qquad (12)$$

where K_p is the number of samples per cycle of the trajectory. A plot of y_{p1} versus y_{p2} yields the required trajectory. A scaled orthogonal BDRNN with four state variables, no input units and two output units, is considered to reproduce the "figure 8" trajectory with a sampling rate K_p = 32 units of time per cycle. The selection of four state variables is motivated by the fact that the plant has a fundamental and second order dynamics each

of which can be modelled by a 2 x 2 block diagonal submatrix. The error gradient accumulated over the entire length of the trajectory is used to update the weight parameters W^n and C^n. For this, μ_1 is chosen to be 64E-5 with a momentum of 32E-5. As a marginally unstable plant is to be modelled, the penalty function for stabilization is chosen so as to constrain the eigenvalues of W^n to an annular ring around and close to the unit circle. This is achieved by setting α_l = 1.1 in (44) and w^{δ}_{min}=0.9 in (28) in Chapter 5, which also ensures invertibility of W^n. Parameter μ_2 is chosen as 2E-4. During simulations, for the chosen initial conditions and training parameters it is observed that, with no stability constraint the training is unstable. On the other hand, with a very tight stability constraint (α_l < 1.0) the network did not learn as the eigenvalues of W^n were placed well within the unit circle, resulting in near-zero outputs towards the end of the trajectory. The number of intermediate storage state vectors required to ensure numerical stability is chosen to be five at evenly spaced intervals over each epoch. At the start of training, the initial state variables of the BDRNN are set to random values in the interval [-1.0,1.0]. The training is started by presenting only one circuit of the trajectory at each run. As the learning progresses, two, four, five and finally ten circuits of the trajectory are presented at each run while still updating the weights at the end of each circuit. At the end of training, the network outputs are observed to continuously trace the trajectory well beyond ten cycles as shown in Figure (6.8). This technique of gradually extending the length of the epoch is useful in avoiding local minima and hence improving the network performance.

The training performance of the network is evaluated using the normalized root mean square error (NRMSE) magnitude. The initial value of the NRMSE at the output is 2.18 and its final value is 0.025. The NRMSE is down to less than 0.2 by 20,000 epochs of training, however, about 200,000 epochs are required to bring the NRMSE to less than 0.025. Note that this example requires a BDRNN with four state variables compared to a ten hidden unit (state variable) fully connected RNN reported in [Pearlmutter 1989] to produce the "figure 8" limit cycles. With the scaled orthogonal form of the BDRNN, the

number of weights required is just four in the BDRNN layer and eight in the output layer compared to about hundred weights in the hidden layer and ten weights in the output layer considered in [Pearlmutter 1989].

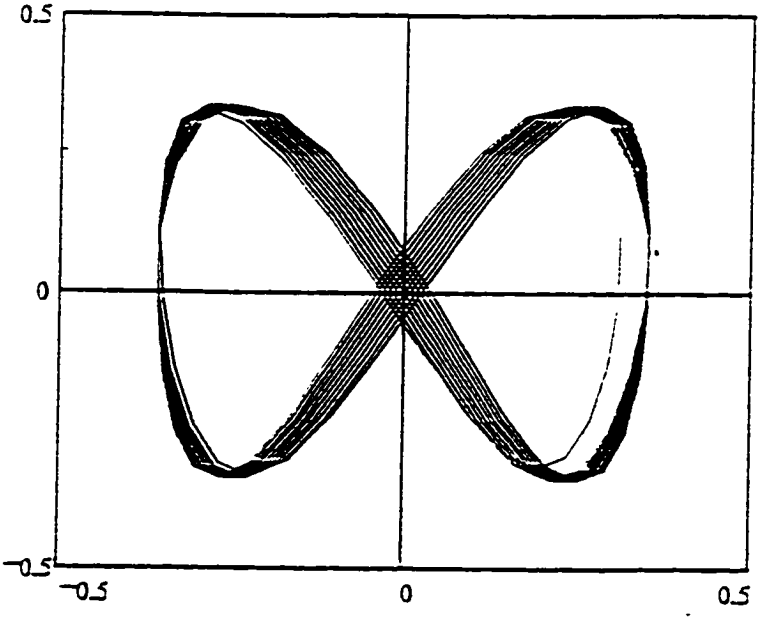


Figure 6.8 Plot of actual outputs y_1^n vs. y_2^n of the BDRNN generating the "figure 8" limit cycle trajectory in Section 6.8.

6.9 MODELLING THE CHAOTIC MACKEY-GLASS DIFFERENTIAL-DELAY EQUATION USING A FF-BDRNN NETWORK

In this example the plant is described by the chaotic Mackey-Glass differential-delay equation [Mackey and Glass]:

$$\dot{x}^p(t) = \frac{0.2\, x^p(t - \tau)}{1 + [x^p(t - \tau)]^{10}} - 0.1\, x^p(t) \qquad (13)$$

This equation is integrated using a fourth-order Runge-Kutta method with a step size of 0.1 to provide values of x^p at discrete time intervals. τ is chosen to be 30 and the initial condition for the plant is $x^p(t) = 0.8$ for $t < 0$. Also the d.c offset of the signal is subtracted from x^p and the BDRNN network is presented with a single input $x^p(t-30)$. The BDRNN output is the predicted value of $x^p(t+6)$. The inputs to the network are taken from the continually evolving time series $x^p(t)$. Initially, the plant is modelled by a 24 state variable, single input, one output scaled orthogonal BDRNN. At the start, the initial conditions for the state variables of the network are chosen to be random numbers in the interval [-1.0,1.0]. Thereafter, after each epoch, the initial values of the state variables are set equal to their final values at the end of the previous epoch. The training is performed by updating the weights after each epoch of hundred time-steps. For this, μ_1 is chosen to be 0.128 with a momentum of 0.064 and μ_2 is chosen to be 0.0512. The number of intermediate storage state vectors, to ensure numerical stability, is chosen to be ten at evenly spaced intervals of each epoch. The stability region is constrained to within the unit disk by setting $\alpha_l = 0.95$ in (44) of Chapter 5. The invertibility of W^n is ensured by setting $w^{\delta}_{min} = 0.5$ in (28) of Chapter 5. With these parameters, the network converges in 20.000 epochs, with a six-step NRMSE of less than 0.33. Increasing the order of the scaled orthogonal BDRNN network or replacing it with a freeform BDRNN with global/local stabilization and retraining it, does not significantly improve the performance of the network. The scaled orthogonal BDRNN network is then augmented with a feedforward subnetwork (equation (17) and (18) of Chapter 3) that consists of six inputs

together with one hidden layer with ten hidden units. The feedforward subnetwork is presented with six input values $x^p(t - 6q)$, q = 0,..5. The resulting FF-BDRNN, when trained afresh with the same parameters as the BDRNN, converges in 30,000 epochs, with a NRMSE at the output of less than 0.03 with a performance comparable to that reported in [Sanger, Day and Davenport]. Figure 6.9 shows the time series dynamics of the plant and the six step prediction output of the FF-BDRNN.

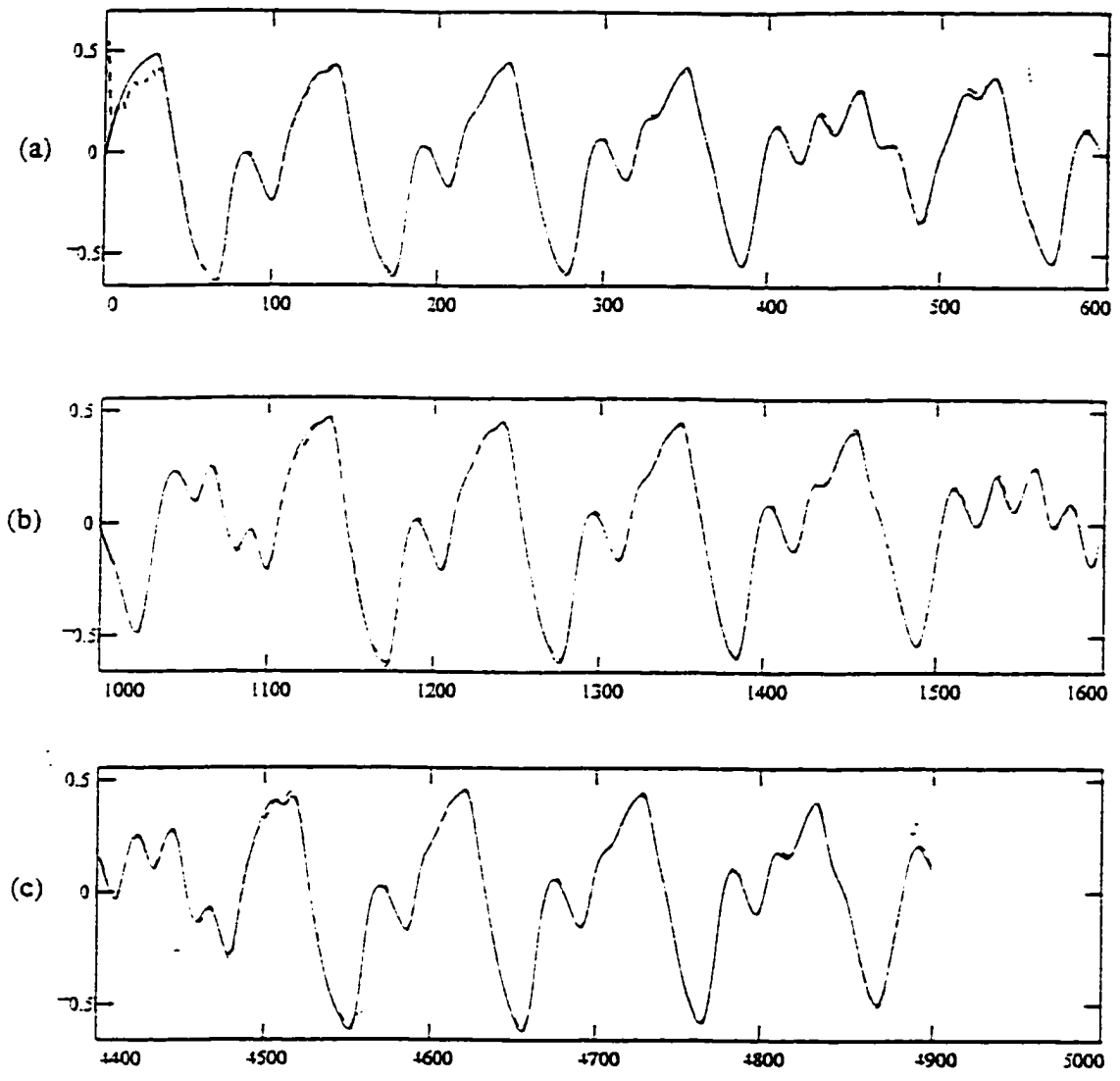


Figure 6.9 Mackey-Glass delay differential plant (——) and FF-BDRNN (——) output trajectories in Section 6.9

When the FF-BDRNN with the scaled orthogonal structure is replaced with a freeform structure with global stabilization and retrained, the resulting NRMSE at the output is observed to be 0.06. The extra restrictions imposed on the weight assignments by the globally stabilized freeform structure may be the reason for the poorer learning of this network when compared to the FF-BDRNN with scaled orthogonal structure. When the global stabilization requirement of the freeform FF-BDRNN is relaxed to local stabilization. the performance of the freeform FF-BDRNN is observed to be identical to that of the scaled orthogonal FF-BDRNN.

For this example, the effect of the number of intermediate storage state vectors on the numerical stability of the recomputation of the state vectors in the backwardpass of the algorithm is also studied. For this, a track of the average number of times the FF-BDRNN encounters numerical instability during the backward pass for varying numbers of intermediate storage is kept. Also, a track of the average number of feedforward computations i.e., recovery computations required to recompute the state variable from the nearest stored intermediate state vector when a numerical instability is encountered, is kept. The results are tabulated in Table 6.1. It is seen from this table that, for this example, when the stored intermediate state vectors exceeds 9% of the total storage required by the conventional BPTT algorithm, no numerical instabilities are encountered. Thus, this example suggests that the proposed algorithm reduces the storage requirement for the state variables by about 90% when compared to conventional BPTT, with no requirement for forward recomputations of state variables in the backward pass. Even when numerical instabilities are encountered the number of feedforward computations required to recompute state vectors are moderate. For example, with 4% intermediate storage state vectors the total average forward computations required in the backward pass are 17.5 for an epoch length of 100. This significantly reduces to 4.5 when the intermediate storage is increased to 5%.

Table 6.1 **Storage vs. computational requirement for state vector recomputation in Example 4**

Number of state vectors stored in an epoch in %	Average no. of numerical instabilities encountered in each cycle with K_p = 100	Average no. of feedforward computations performed each time a numerical instability is encountered
1	4.32	46
2	4	20
3	3.5	12.5
4	3.5	5
5	1.5	3
6	0.65	2.15
7	0.11	1.35
8	0.065	1.6
9	0	0
10	0	0

6.10 EXPERIMENTS WITH SINGLE-DIGIT SPEAKER-DEPENDENT SPEECH DATA

Speech sounds can be characterized by their spectral and temporal properties that depend on the acoustic features of the sound and are manifest in the waveform, the spectra, or both [Rabiner]. In this section, we study the application of FF-BDRNN to raw speech data with a objective of recognizing isolated word utterances. The difficulties encountered in speech recognition are primarily due to the complexity of the speech signal itself. Speech is produced by the vocal tract with slow dynamics, with speech cues distributed over time

in a complex manner. Speech prediction/recognition is hampered by several factors including the variability of the cues caused by coarticulatory effects, dynamic behaviour of the spectral properties, the nonlinear nature of the signal and cues that vary from speaker to speaker.

The current approaches to speech recognition include:

(i) the acoustic approach which directly uses the properties of sound.

(ii) the signal processing/pattern recognition approaches which can be subdivided into
- speech spectra modelling techniques such as filter banks, linear predictive coding (LPC) [Rabiner and Juang].
- statistical modelling techniques like hidden Markov modelling (HMM) [Rabiner 1989], and
- dynamic programming techniques to temporally align and compare speech utterances of differing lengths.

(iii) the neural network approach which includes time-delay neural networks (TDNN) [for example, Waibel et al, Sivakumar 1992a, Sivakumar et al 1992b and 1992c] and adaptive-time delay networks, and self organizing feature finders [Kohonen].

Before speech classification or recognition is attempted, the speech signal is generally preprocessed, to extract important information from the speech waveform, by quantizing, scaling, preemphasising, windowing, filtering, auto/crosscorrelating, fast fourier transforming, time warping etc. The information extracted includes location and transition of formants, location of end points, number of zero-crossings, energy in various spectral bands, frequency content etc.

The performance of the FF-BDRNN architecture introduced in Chapter 3, learning algorithm of Chapter 4, and the stabilization approach of Chapter 5, are used in the six-step ahead prediction of a practical time-series, viz., the speech utterances from "zero" to "nine". The one-step ahead prediction of speech utterances has previously been demonstrated by [Haykin and Li (1995)] on a continuous speech signal using a pipelined recurrent neural network and by [Tsoi and Back] on the utterance "one" using a class of LRGF networks. In this chapter, the speech utterances chosen are those of an adult male with a Boston accent and consist of the ten isolated digits "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" and "zero" which form part of the TIMIT connected-digit speech corpora [TIDIGIT]. The speech in the database is sampled at 20 kHz and a 15 bit linear A/D converter is used to quantize the speech. The total length of the utterance varies from digit to digit and ranges from about 17,000 to 30,000 samples. For the purposes of this simulation a small segment consisting of 2000 samples is sliced out of each utterance. In all the digits, the slicing is done in a section about halfway through the utterance where the energy is generally large. This is done so as to select sections that exhibit strong nonlinear behaviour. The actual speech signal is scaled to between [-0.9,0.9] and is not otherwise preprocessed.

The focus of this simulation is not on how precisely the model predicts the speech waveform, but on how the prediction performance of a network trained on one utterance varies when the input to the network is the waveform of other utterances, and to use this performance variation in speech classification i.e., speech recognition is attempted based on speech prediction techniques [Pearlmutter 1995]. Each digit recognition module consists of a single input, 24 state variable, single output FF-BDRNN with the feedforward section consisting of six input and 10 hidden units. The inputs to the feedforward section consist of the speech signal at six time steps $x'(t - 6q)$, $q = 0,..5$ The BDRNN network is presented with a single input $x'(t-30)$. The output of the FF-BDRNN digit recognition module consists of the predicted value of the speech waveform $x'(t+6)$.

Shown in Figure 6.10 (a) is the waveform of the utterance "zero". Figure 6.10 (b) compares the sliced section of the actual utterance "zero" (in solid line) with the DRM predicted value (in dotted line) when the input to the DRM is "zero". Figure 6.10(c) compares the actual utterance "zero" on which the DRM was trained with the DRM predicted value when the input to the DRM is "two". Figure 6.10(d) compares the actual utterance "zero" on which the DRM was trained with the DRM predicted value when the input to the DRM is "five".

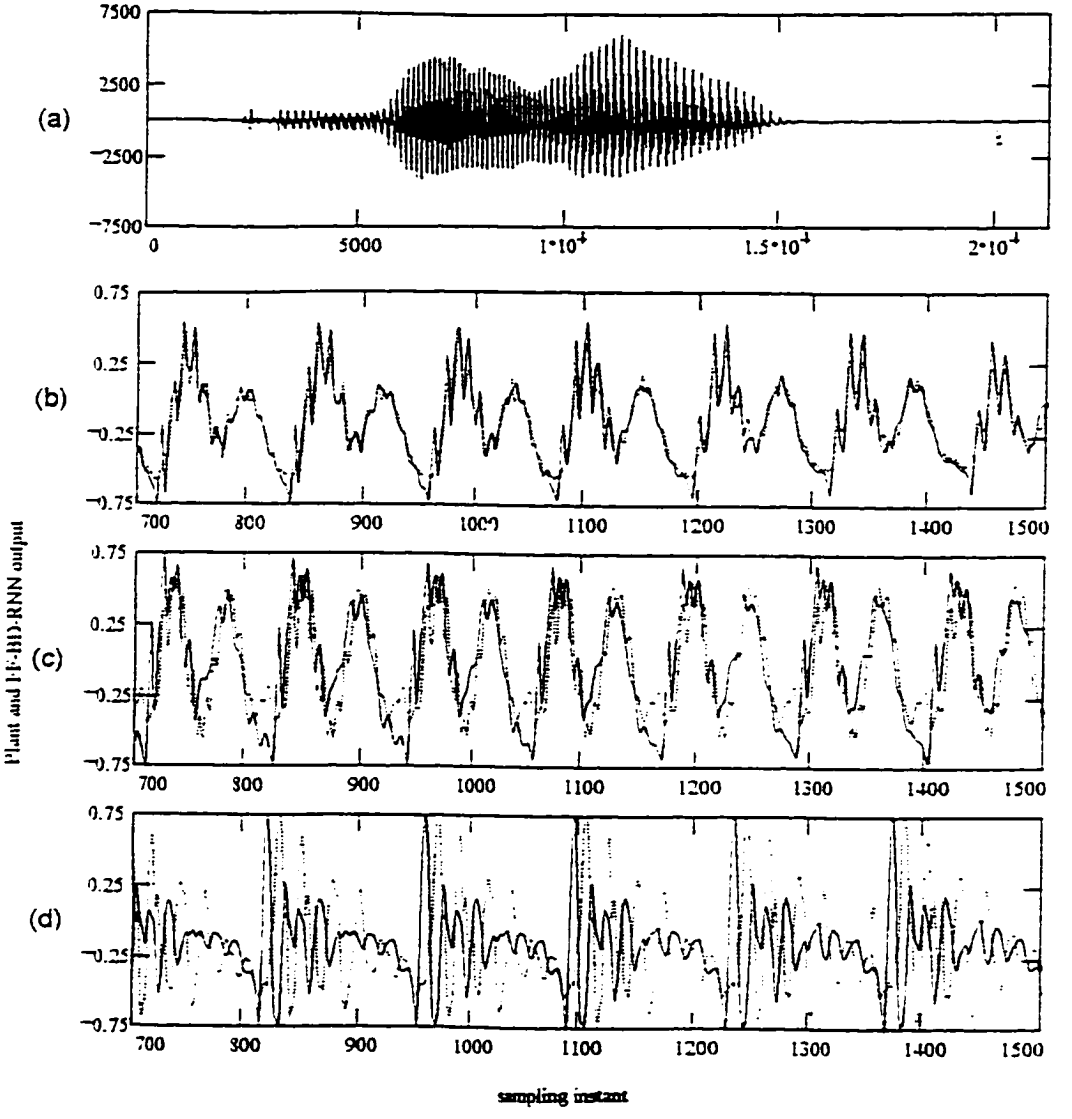


Figure 6.10 Simulation results for isolated word recogntion example in Section 6.10: (a) Waveform of the utterance "zero". (b) Plot of the sliced section of the actual utterance "zero" (——) vs. DRM predicted value (-----) when the input to the DRM is "zero". (c) Plot of the actual utterance "zero" on which the DRM is trained (——) vs. DRM predicted value (-----) when the input to the DRM is "two". (d) Plot of the actual utterance "zero" on which the DRM is trained (——) vs. DRM predicted value (-----) when the input to the DRM is "five".

Ten separate modules are trained to predict the waveforms of the ten digits respectively. Each of these modules is tested on the utterance it was trained on and also on all the other nine utterances on which it is not trained. The results are tabulated in Table 6.2. Each column in Table 6.2 tabulates the NRMSE when each of the digit recognition modules is input the digits from "one" to "zero". The entry in **bold** corresponds to the NRMSE when each module is input the utterance on which it was trained.

Table 6.2 NRMSE when a BDRNN module trained to model a "digit" is tested on digits 0-9.

DRM tested on	Digit Recognition Module trained for									
	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"0"
"1"	**0.37**	1.24	1.42	1.08	1.09	1.60	0.96	1.12	1.11	1.39
"2"	1.26	**0.46**	0.74	0.74	1.69	1.47	2.02	0.66	1.87	0.65
"3"	1.78	0.81	**0.55**	1.08	1.61	1.30	1.84	1.02	1.80	0.88
"4"	0.49	0.74	0.80	**0.34**	1.39	1.46	1.26	0.77	1.30	0.93
"5"	1.63	1.44	1.44	1.38	**0.60**	1.67	1.0	1.39	1.14	1.53
"6"	1.94	1.12	1.05	1.85	2.03	**0.50**	1.9	1.09	1.95	1.29
"7"	1.58	1.11	1.30	0.74	0.84	1.44	**0.38**	1.26	0.80	1.35
"8"	1.08	0.77	1.16	1.76	1.75	1.0	2.58	**0.44**	2.68	1.06
"9"	2.04	1.12	1.13	1.14	0.96	1.65	0.87	1.54	**0.28**	0.99
"0"	1.34	0.73	0.75	0.78	1.2	1.92	1.52	1.11	1.16	**0.33**

Table 6.3 tabulates the performance of the DRM when it is input a second set of digits spoken by the same speaker. The entry that is underlined in column 2 corresponds to the

NRMSE when the "one" recognition module is input the digit "one", that was not used to train the module, and was spoken by the same speaker a second time. Similarly, the other underlined entries correspond to the performance of the other DRMs on the second set of utterances not used in training the respective DRMs.

Table 6.3 NRMSE when each digit recognition module is tested on a second set of "digit" utterances by the same person.

DRM tested on a second set utterances	Digit Recognition Module trained for									
	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"0"
"1"	0.45	1.24	1.42	0.96	1.13	1.34	0.85	1.04	1.09	1.63
"2"	1.26	0.56	0.74	0.70	1.76	1.38	1.69	0.78	1.65	0.70
"3"	1.78	0.81	0.64	0.93	1.59	1.37	1.47	1.21	1.39	0.75
"4"	0.49	0.74	0.80	0.41	1.40	1.45	1.15	0.78	1.13	0.98
"5"	1.63	1.44	1.44	1.33	0.62	1.74	0.95	1.40	1.09	1.48
"6"	1.94	1.12	1.05	1.14	2.15	0.63	2.00	1.3	2.25	1.13
"7"	1.58	1.11	1.30	1.07	1.05	1.26	0.74	1.38	1.05	1.54
"8"	1.08	0.77	1.16	1.47	2.61	1.05	2.50	0.53	2.84	1.1
"9"	2.04	1.12	1.13	1.04	0.88	1.74	0.94	1.46	0.45	0.92
"0"	1.34	0.73	0.75	0.81	1.59	1.54	1.54	1.14	1.45	0.61

It can be seen from Table 6.2 that, the NRMSE is the least when the DRM is input the waveform that it is originally trained on. It is seen that the NRMSE is, in general, 150% to 400% as bad when the DRM is input any other utterance with the exception of the performance of the DRM trained on "one" when tested on "four". This is due to the

similarity in the sliced sections of the two waveforms. Comparing the diagonal entries of Table 6.2 and 6.3, it can be seen that, the performance of the DRM slightly degrades when it is input the second set of digits on which it was not trained.

A possible scheme for recognition of the digits can be formulated in the following fashion:

(i) Design ten modules to perform the six-step prediction of the digits "one" to "zero" respectively.

(ii) Quantize the utterance to be recognized and then slice out a section of samples corresponding to a maximum in speech energy. Scale the sample.

(iii) Input the scaled utterance to all the ten modules and observe their NRMSE.

(iv) The module that predicts the input utterance with the least NRMSE has "recognized" the utterance.

Table 6.4 **Summary of network and training parameters for the examples of sections 6.2 - 6.10.**

Plant	network	μ_1	μ_2	mom	K_p	$w^{\delta}_{min}, \alpha_1$	N_s	iteration
6.2	FF-BDRNN	0.025	---	0.012	32	---	4	10000
6.3	BDRNN - O & F	0.025	0.016	0.012	32	0.5,0.95	4	10000
6.4	BDRNN - O & F	0.025	0.016	0.012	48	0.5,0.95	5	10000
6.5	BDRNN - O & F	0.032	0.025	0.016	48	0.5,0.95	5	10000
6.6	BDRNN - O & F	0.004	0.0032	0.002	50	0.5,0.95	5	11000
6.7	FF-BDRNN - O & F	0.016	0.012	0.008	50	0.5,0.95	5	100000
6.8	BDRNN -O	64 E-5	0.0002	32 E-5	32	0.9, 1.1	5	200000
6.9	BDRNN -O FF-BDRNN -O	0.128	0.05	0.064	100	0.5,0.95	10	20000 30000
6.10	FF-BDRNN -O	0.160	0.063	0.080	125	0.5,0.95	10	60000

Note: O - scaled orthogonal, F- freeform

6.11 CONCLUSION

The scaled orthogonal BDRNN in which the global and local stability constraints are equivalent is the better of the two architectures for modelling nonlinear plants with oscillatory modes as it does not impose undue restrictions on weight assignments. Simulation studies presented indicate the viability of the proposed FF-BDRNN architecture with stabilization together with the learning algorithm for modelling plants with a variety of nonlinear dynamics.

In all the examples presented in this chapter, the stability of learning was also assessed without imposing the network stability constraints. In all such cases, in many instances, the learning was observed to be unstable. One reason for this instability was traced to the choice of the initial weights of the network (note that the weights of W^n were assigned random numbers chosen such that the eigenvalues of the 2 x 2 submatrices of the network were on the unit circle) in relation to the choice of learning parameters. Also, even when the initial weight matrix was chosen to be stable, in some cases the network was observed to migrate into instability when the stability constraint was not imposed during training. However, with the stability constraint in place, in all cases the learning was observed to be stable for the same network and training parameters.

7 CASCADING CONSTRUCTIVE TRAJECTORY LEARNING IN BLOCK-DIAGONAL RECURRENT NEURAL NETWORKS

7.1 INTRODUCTION

The size and architecture of DTRNN are generally determined by trial and error. However, this technique results in training a number of networks and then selecting the smallest network that meets the required performance criterion. This chapter proposes a constructive method of designing BDRNNs to model trajectories. The motivation for such an approach is derived from the possibility that a nonlinear dynamic trajectory can be decomposed into a combination of a dominant dynamic and a series of progressively less dominant subdynamics. Blocks of BDRNNs can be used, with each block employed to learn the dominant dynamic or one of the subdynamics. The advantage of such an approach is that the size of the BDRNN is determined methodically with better error performance and a faster learning time in comparison with the larger equivalent network that may be used to learn the same dynamic without construction.

As discussed in Chapter 2, Waibel et al [Waibel et al] have demonstrated a construction technique for feedforward networks used for a speech recognition application by using "glue" units that integrate the output of network submodules assigned to recognize subclasses of consonants. In this technique, the weights of these submodules are first trained and frozen and only the weights that connect the glue units to the submodules are trained during construction. Fahlman [Fahlman 1991] has constructed cascading feedforward neural networks with self recurrent neurons, by adding one hidden neuron at a time. Giles et al [Giles et al 1995] also have used similar techniques to construct fully recurrent DTRNN with hard threshold neurons by adding one neuron to the recurrent layer after a fixed number of epochs to learn a number of finite state automata. Draelos et al [Draelos and Hush] have constructed a composite feedforward network whose each stage consists of a single hidden-layer feedforward subnetwork with the output of the composite

network approximating the target function to be modelled. The hidden layer at any stage of the composite network is trained on the residual error between the most recent estimate and the target function.

As considered in the references cited above, any construction algorithm must address issues related to the following:

(i) Addition of construction units: a construction unit may consist of either a single neuron or a module made up of several neurons. The purpose of construction is either to improve the performance of, or to add new information to, an already trained network; and hence, improve the generalization capability of the neural network.

(ii) Weight update strategy: one of the following two approaches may be adopted for updating the weights: (a) update all the weights by completely retraining the entire network after the addition of each neuron/module, (b) freezing the weights of the existing network and train only the weights associated with the added neuron/module by partial retraining of the network. The complete retraining method is computationally much more expensive than the freeze and retrain method, but has a larger degree of freedom than the latter in finding the global solution. Hence, the freeze and retrain method may need more neurons/modules resulting in non-minimal networks.

(iii) Choice of performance index for training: at each construction stage, the weight updating may be performed to effect the reduction of either the output error of the composite network or the residual error which is the output error at that stage of construction. In general, the output error reduction method is adopted together with the addition of a single neuron at a time. The residual error reduction method is adopted whenever a new packet of information is to be learnt.

(iv) Criterion to be used for starting network growth: there are two commonly used criterions. In the first, a neuron/module is added after an arbitrary number of epochs of training, until convergence. In the second, a neuron/module is added each time the output training error shows no significant improvement, until convergence. Either technique may lead to a large number of neurons/modules being added especially towards the end of training.

7.2 CONSTRUCTION IN THE CONTEXT OF BDRNN

Although successful modelling with FF-BDRNNs of several nonlinear dynamic trajectories has been demonstrated in the previous chapters, there exist situations where the effectiveness of learning is restricted by the presence of dominant dynamic components within the target system being modelled, and their degree of dominance over other less dominant dynamics. This type of "decomposition" is analogous to a waveform in a linear system, which can be decomposed into a fundamental and several harmonics of diminishing magnitude. In a nonlinear dynamic system, a desirable strategy for learning would be:

(i) the decomposition of the target system, if possible, into their constituent dynamic components,

(ii) classification of these components into groups based on the degree of the dynamic dominance and

(iii) individual stable learning of each of these groups.

A construction strategy which addresses these objectives is presented in this chapter.

In the context of BDRNNs, let us define the dominant dynamic of a target system as that obtained with a single BDRNN that provides the best output error performance, i.e., the trajectory output by the BDRNN when no improvement or a deterioration in its performance is achieved with an increase in its order. At this performance level, one may

say that the BDRNN has learnt the dominant dynamic in the target system. The output error trajectory then represents the subdynamics that remains after subtraction of the dominant dynamic from the target system. The above definition of dominant dynamic assumes that its order is known *apriori* and is matched with the order of the BDRNN. However, in practical situations, the order of the dominant dynamic is not known *apriori* and can be found only by trial and error, which is computationally expensive. In such situations, the best that can be done is to assume an arbitrary sufficiently large order BDRNN to model the dominant dynamic of the target system in a reasonable fashion. Let us now define the estimate of the dominant dynamic as the output of a single BDRNN of a specific order with the best error performance. The BDRNN output error then represents the sum of the error in estimating the dominant dynamic and the remaining subdynamics of the target system. Assuming that the estimate error is small, the output error can now be suitably scaled and presented to a second BDRNN module as its target trajectory. The second BDRNN similarly extracts an estimate of the dominant dynamic in its target trajectory. This process can be continued until the desired output performance goal is achieved.

The proposed construction algorithm [Sivakumar et. al. 1996b] starts with a basic module consisting of N_1 block-diagonal units capable of modelling dynamic modes corresponding to N_1 eigenvalue pairs of the system being modelled. Learning is performed with this basic architecture until there is no improvement in the performance criterion, at which stage, the weights of the existing architecture are frozen and incremental construction is accomplished by cascading a BDRNN module made up of N_2 block-diagonal units to the existing architecture and training only the weights in the new module. This process is continued until the desired output performance criterion is achieved or a selected maximum number of modules have been added. The block diagram of the cascading constructive learning architecture is illustrated in Figure 7.1.

7.3 MATHEMATICAL FORMULATION OF THE CONSTRUCTION PROBLEM

7.3.1 Motivation

Let us assume that the nonlinear trajectory $g_1(x,u,t)$ to be modelled can be written as a sum of several dynamic trajectories ordered in descending order of dominance, i.e.,

$$g_1(x,u,t) = k_1(f_1(x,u,t) + k_2(f_2(x,u,t) + \ldots + k_{N_c}(f_{N_c}(x,u,t)) \ldots)) \quad (1)$$

with k_q is a constant < 1, representing the degree of dominance of the constituent dynamic trajectory $f_q(x,u,t)$. Let us assume that $f_1(x,u,t)$ can be extracted from $(1/k_1)g_1(x,u,t)$ by a BDRNN with reasonable accuracy. Then what remains is the weighted sum of the subdynamics of the target trajectory and is given by:

$$g_2(x,u,t) \approx k_2(f_2(x,u,t) + k_3(f_3(x,u,t) + \ldots + k_{N_c}(f_{N_c}(x,u,t))\ldots)) \quad (2)$$

Let us again assume that $f_2(x,u,t)$ can be extracted from $(1/k_2)g_2(x,u,t)$ by another BDRNN with reasonable accuracy. Continuing this process, the target trajectory at stage q is given by:

$$g_q(x,u,t) \approx k_q(f_q(x,u,t) + k_{q+1}(f_{q+1}(x,u,t) + \ldots + k_{N_c}(f_{N_c}(x,u,t))\ldots)) \quad (3)$$

This process can be continued until all the subdynamics in the R.H.S of (1) is extracted. Let $\hat{f}_q(x,u,t)$ be the output of the q-th BDRNN module that estimates $f_q(x,u,t)$. The composite error at the q-th stage is then given by:

$$\frac{1}{k_q}g_q(x,u,t) - \hat{f}_q(x,u,t) = \zeta_q(t) + k_{q+1}(f_{q+1}(x,u,t) + k_{q+2}(f_{q+2}(x,u,t) + \ldots)) \quad (4)$$

where $\zeta_q(t)$ is the error in estimating $f_q(x,u,t)$. The implementation of the construction technique, will be successful if:

$$\|\zeta_q(t)\| \le \|k_{q+1}(f_{q+1}(x,u,t))\| \tag{5}$$

is satisfied where $\|.\|$ is a suitably defined norm, for e.g., Euclidean norm. If (5) is not satisfied, the subdynamic constituents of the target trajectory of the q+1-th BDRNN may be corrupted by the estimate error in the q-th construction stage resulting in poor performance thereafter.

7.3.2 Construction algorithm description

The algorithm starts by constructing a basic module made up of N_I block-diagonal submatrices with M inputs and N_o outputs. The output $y_h^q(k)$ of the q-th module is given by:

$$\begin{aligned} x_i^q(k+1) &= f_a\left(\sum_{j=v}^{v+1} w_{ij}^q x_j^q(k) + \sum_{j=1}^{M} b_{ij}^q u_j(k)\right) \qquad i=1,..,N_q \\ &\textit{where} \quad v = i, \quad \textit{if } i \textit{ is odd}, \\ &\qquad\quad v = i-1, \quad \textit{if } i \textit{ is even}. \\ y_h^q(k) &= f_b\left(\sum_{j=1}^{N} c_{hj}^q x_j^q(k)\right), \qquad h=1,...,N_o \end{aligned} \tag{6}$$

The input $u_j(k)$ is the external input to the plant. The residual error from the q-th stage is computed as:

$$e_h^q(k) = d_h^q(k) - y_h^q(k), \qquad h=1,...,N_o \tag{7}$$

A scaled version of the frozen residual error in the q-1-th stage is the desired trajectory to the module in the q-th stage. The desired value $d_h^q(k)$ is given by:

$$\begin{aligned} d_h^q(k) = \vartheta_{q-1} e_h^{q-1}(k) &= \vartheta_{q-1}\left(d_h^{q-1}(k) - y_h^{q-1}(k)\right), \\ h=1,...,N_o, \quad k &= q,...,K_p \end{aligned} \tag{8}$$

where $d_h^1(k) = d_h(k)$ is the desired value of the target trajectory and ϑ_{q-1} is an amplifying scaling factor ≥ 1.0.

The composite estimate $\theta_h^q(k)$ of the desired trajectory at the q-th module at any instant k and is given by:

$$\theta_h^q(k) = y_h^1(k) + \frac{1}{\hat{v}_1}\left(y_h^2(k) + \frac{1}{\hat{v}_2}\left(y_h^3(k) + \ldots + \frac{1}{\hat{v}_{q-1}}\left(y_h^q(k)\right)\right)\ldots\right), \quad h=1,\ldots,N_o \tag{9}$$

Each BDRNN module is trained with the algorithm described in Chapter 4 with the stability compensation technique described in Chapter 5, with the objective of minimizing the residual output error $e_h^{q-1}(k)$. The algorithm starts with a basic module made up of N_I block-diagonal submatrices with M external inputs and N_o outputs.

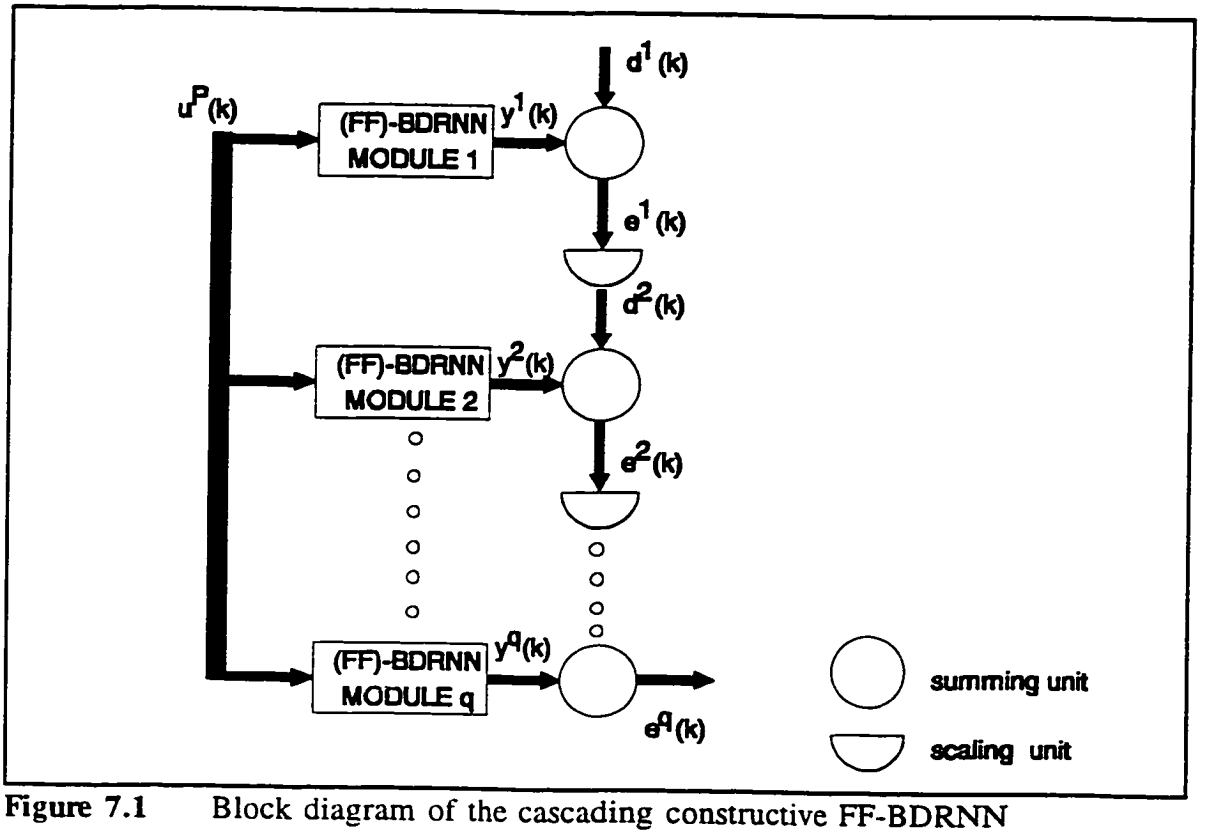


Figure 7.1 Block diagram of the cascading constructive FF-BDRNN architecture.

7.3.3 Stability of the composite network

From (1) it can be seen that the stable learning of the trajectory requires that each of the individual trajectories given by the R.H.S of (1) can be learned by individually stable BDRNN modules. Thus, the stability of the composite BDRNN can be ensured by ensuring the individual stability of each module of the composite network. At any instant in the training of the q-th module, the weights of the previous q-1 modules have already been frozen (and are assumed to be stable). Hence, stability of the composite network and its learning requires the individual stability of each q-th module during its training. The local stability of the composite BDRNN can be ensured by:

$$| \lambda_i(W^q) | \leq \frac{2}{a} \qquad \begin{matrix} i = 1,...,N_Q \\ q = 1,..,N_c \end{matrix} \tag{10}$$

where $\lambda_i(W^q)$ is the i-th eigenvalue of the square matrix $W^q = \{w^q_{ij}\}$. For the *global* stability of the composite BDRNN with a nonsingular feedback weight matrix W^q

$$\left[\lambda_{max}((W^q)^T W^q) \right]^{1/2} < \frac{2}{a} \qquad q = 1,..,N_c \tag{11}$$

where $\lambda_{max}(.)$ is the maximum eigenvalue of the matrix. These conditions can be directly ensured by using the methodology discussed in Chapter 5, when training each module of the composite BDRNN.

7.3.4 Analysis and discussion

7.3.4.1 Strengths

The cascading constructive trajectory learning algorithm offers the following advantages over conventional learning techniques:

(i) This technique has faster learning time and good generalization capabilities as

each module focusses on learning the dominant dynamics of its desired trajectory. The various modules do not interact with each other. Each module is presented with the same input data and uses the residual error from the previous module as its desired trajectory. Hence, at any instant, any one BDRNN module is being trained by propagating the error signals through the connection weights in that module and not through all the connections in the composite network. This reduces training time and complexity.

(ii) The problem of ensuring network and learning stability is also decomposed, as the various modules do not interact and are not being trained simultaneously. At any instant, the stability of the composite network is addressed by addressing the stability requirements of only the module that is being trained. This simplifies the stability problem.

7.3.4.2 Limitations

(i) This method of construction presupposes that the nonlinear dynamic trajectory being learnt is decomposable into subdynamics in decreasing order of dominance as defined by (1). This technique may fail if used for problems requiring fixed-point learning such as in modelling finite state machines. For such problems it might not be possible to "decompose" the finite state dynamics into meaningful subdynamics.

(ii) This technique will also fail when the error $\zeta_{si}(t)$ in estimating the dynamics already learnt at any stage is comparable to the dynamics yet to be learned. Under such conditions, the estimation error $\zeta_{si}(t)$ introduces an unwanted dynamic that is as dominant as, if not more dominant than, the residual error $e_k^q(k)$ dynamics, and may result in the module toggling between learning the unrequired estimation error dynamic and the required residual error dynamics, resulting in poor overall

learning performance.

(iii) While this technique results in a collection of small decomposed networks, the resultant network at the end of training may not be the *optimal* size of the network for a given problem.

7.4 SIMULATION RESULTS ON CONSTRUCTIVE LEARNING

7.4.1 Six-step prediction of the MacKey-Glass delay differential equation with BDRNN

The plant considered in this example is the Mackey-Glass delay differential equation described by (13) of Chapter 6. The objective here is to first simulate the six-steps ahead prediction results of the Mackey-Glass delay differential equation with only BDRNN construction modules with no feedforward networks. A comparison in performance between a single-input 24-state variable, single-output BDRNN and a constructively trained BDRNN consisting of four modules is provided in Table 7.4.1. The training parameters and performance details for these two networks is listed in this table. The conditions under which the BDRNN networks are trained is identical to that described in section 6.9 of Chapter 6. From Table 7.4.1, it can be seen that the output NRMSE for the single-input, 24-state variable, single-output BDRNN network converges to 0.33 after training it for 20,000 iterations. Increasing the order of the BDRNN to 32-state variable or larger and retraining has not yielded appreciably better results. Next, a BDRNN network is constructive built from scratch by first, beginning with a single-input, 8-state variable, single output BDRNN as the basic module. During training the output NRMSE reduces steadily from a initial 0.73 to a final 0.464 and does not reduce any further when training is continued. At this stage, a second BDRNN block consisting of 8-state variables is added. The input to the second block is the output error of the basic BDRNN module. On training the NRMSE of the second stage reduces to 0.574, yielding an effective

NRMSE of 0.266 for the composite network consisting of these two cascading blocks. At this stage, a third block also consisted of a 8-state variable BDRNN is added. The input to the third stage is the output error from the second stage scaled by a factor of 2. During training the NRMSE of the third stage reduces to 0.83, yielding an effective NRMSE of 0.22 for the composite network consisting of the three cascading blocks. At this stage, a fourth module again consisting of an 8-state variable BDRNN is added. The input to the fourth stage is the output error from the third stage scaled by a factor of 2. During training the NRMSE of the fourth stage reduces to 0.94, yielding an effective NRMSE of 0.153. It is seen that with constructive learning there is significant improvement in trajectory prediction performance and the effective NRMSE of the composite network consisting of four cascading BDRNN blocks is twice as better as the 24-state variable "monolithic" BDRNN. The six step prediction results of the 24-state BDRNN and the cascading constructive four stage BDRNN are shown in Figure 7.2 along with the plant output. It is also seen that training the cascading constructive BDRNN requires approximately 56% of the training time required for the monolithic 24 state variable BDRNN.

Table 7.1 Training parameters and performance for BDRNN construction

Network inputs (M), states (N), outputs (N_o)	μ_1	μ_2	momentum	NRMSE	iterations
BDRNN without construction $M = 1$, $N = 24$, $N_o = 1$					
BDRNN	0.128	0.0512	0.064	0.82 to 0.33	20,000
BDRNN construction $M = 1$, $N_1 =$, $N_2 = N_3 = N_4 = 8$, $N_o = 1$					
Module 1	0.128	0.0512	0.064	0.73 to 0.464	200
Module 2	0.128	0.0512	0.064	0.286	1000
Module 3	0.128	0.0512	0.064	0.22	2500
Module 4	0.128	0.0512	0.064	0.153	7500

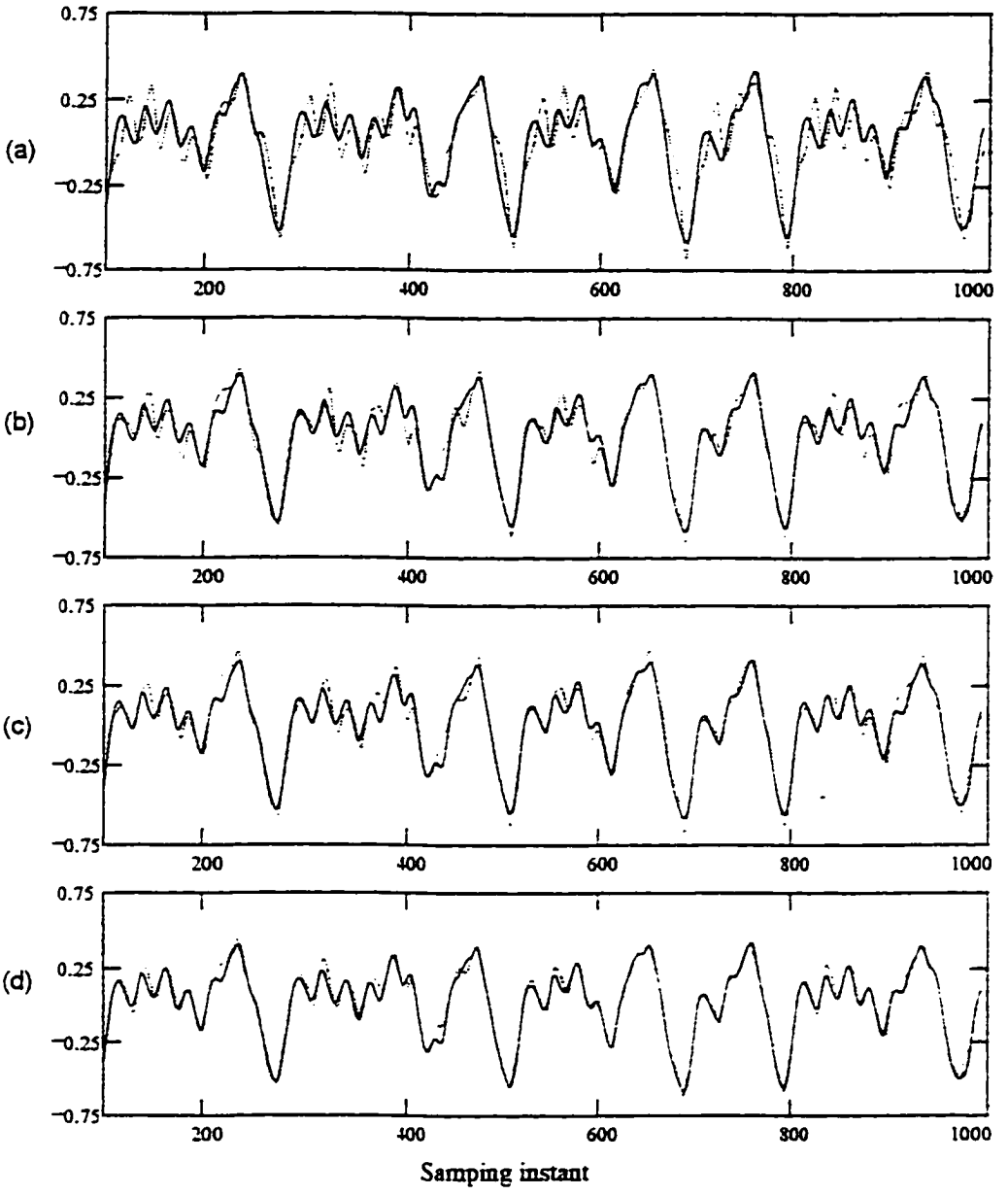


Figure 7.2 Mackey-Glass delay differential plant (_____) and constructive BDRNN (----) output trajectories - (a) first stage (b) first and second stages (c) first to third stages (d) first to fourth stages -

7.4.2 Six-step prediction of the MacKey-Glass delay differential equation with FF-BDRNN

The plant considered in this example is the Mackey-Glass delay differential equation described by (13) of Chapter 6. The objective here is to simulate the six-steps ahead prediction results of the Mackey-Glass delay differential equation with FF-BDRNN construction modules. A comparison in performance between constructively trained FF-BDRNN consisting of four modules is provided in Table 7.4.2. The training parameters and performance details for these constructively trained networks is listed in this table. The conditions under which the FF-BDRNN networks are trained is identical to that described in section 6.9 of Chapter 6. From section 6.9 of chapter 6, it can be seen that the output NRMSE for the single-input, 24-state variable, single-output FF-BDRNN with a feedforward subnetwork (equation (17) and (18) of Chapter 3) that consists of six inputs together with one hidden layer with ten hidden units, converges to 0.03 after training it for 30,000 iterations. Next, a FF-BDRNN network is constructive built from scratch by first, beginning with a single-input, 8-state variable, single output FF-BDRNN with the feedforward section consisting of six inputs together with one hidden layer consisting of ten hidden units as the basic module. On training the basic module the output NRMSE reduces steadily from a initial 1.92 to a final 0.23 and does not reduce any further when training is continued. At this stage, a second FF-BDRNN block that is identical to the basic module in architecture is added. The input to the second block is the output error of the basic FF-BDRNN module scaled by a factor of four. On training the NRMSE of the second stage reduces to 0.475, yielding an effective NRMSE of 0.11 for the composite network consisting of these two cascading FF-BDRNN blocks. At this stage, a third block also identical to the basic FF-BDRNN module is added. The input to the third stage is the output error from the second stage scaled by a factor of 2.5. During training the NRMSE of the third stage reduces to 0.85, yielding an effective NRMSE of 0.095 for the composite network consisting of the three cascading blocks. At this stage, a fourth FF-BDRNN module again identical to the basic module is added. However, a significant

improvement in the output NRMSE performance is not observed on the addition of the fourth module. Hence construction was stopped with a composite architecture consisting of three cascading FF-BDRNN modules. However, it is noted that with constructive learning there is no significant improvement in trajectory prediction performance and the effective NRMSE of the composite network consisting of three cascading FF-BDRNN blocks when compared to the "monolithic" 24-state variable, FF-BDRNN with a feedforward subnetwork of six-input and ten hidden units. The six step prediction results of the cascading constructive three stage FF-BDRNN are shown in Figure 7.2 along with the plant output. It is also seen that training the cascading constructive FF-BDRNN requires approximately 57% of the training time required for the monolithic 24 state variable FF-BDRNN.

Table 7.2 Training parameters and performance for FF-BDRNN construction

Network inputs (M), states (N), outputs (N_o)	μ_1	μ_2	momentum	NRMSE	iterations
FF-BDRNN construction $M = 1$, $N_1 = , N_2 = N_3 = N_4 = 8$, $N_o = 1$ with the **feedforward section** input = 6, hidden units in the first hidden layer = 10.					
Module 1	0.128	0.0512	0.064	1.92 to 0.202	2000
Module 2	0.128	0.0512	0.064	0.112	12000
Module 3	0.128	0.0512	0.064	0.095	3000

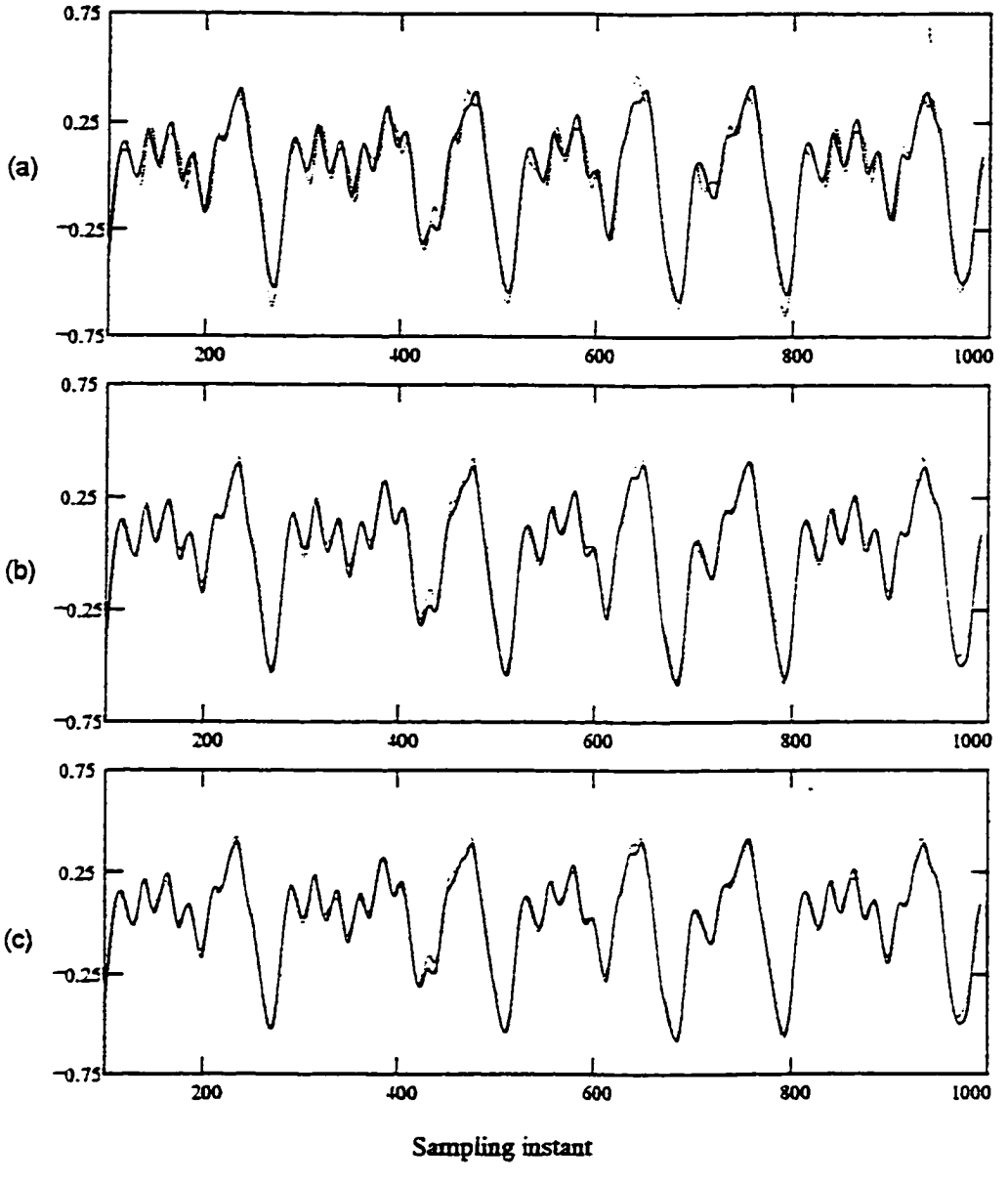


Figure 7.3 Mackey-Glass delay differential plant (——) and constructive FF-BDRNN (---) output trajectory - (a) first stage (b) first and second stages (c) first, second and third stages.

7.5 CONCLUSION

This chapter deals with the idea of constructively designing an FF-BDRNN on the assumption that the target nonlinear dynamics can be decomposed into a dominant dynamic and a series of less dominant subdynamics. The construction technique employs cascaded BDRNN/FF-BDRNN modules to learn the dominant and each of the subdynamics. The size of the network arrived at by this technique may not be the optimal one required to model a given dynamic trajectory, however, this technique results in faster learning time in comparison with a single FF-BDRNN network that is trained to model the same trajectory without construction.

8 CONCLUSION AND DIRECTIONS FOR FUTURE RESEARCH

In this thesis a new DTRNN architecture, called the FF-BDRNN, is introduced, which with its sparse and block-diagonal structure, (i) is capable of effectively modelling a class of nonlinear dynamic trajectories, (ii) provides an efficient framework for developing a technique for the stabilization of the network and its training, (iii) is conducive to designing a BPTT based learning algorithm with significantly reduced storage requirement and (iv) provides a framework for network construction using modules. The applicability of the proposed architecture, its stabilization, the training algorithm and the construction method are demonstrated for several examples encompassing the following: production of limit cycles, emulation of autonomous plants with fully-connected DTRNN architecture, prediction of the outputs of single-input single-output and multiple-input multiple-output nonlinear plants, chaotic time series and speech recognition using speech prediction techniques.

8.1 MAJOR CONTRIBUTIONS

The major contributions of this thesis are the following:

(i) **BDRNN / FF-BDRNN architecture**: A block diagonal recurrent neural network architecture (BDRNN) is proposed in which the *complex eigenvalues* of the block-diagonal submatrices of the feedback weight matrix are better suited, than purely self-recurrent networks, to model the oscillatory dynamic modes of non-linear trajectories. This architecture is extended to include feedforward connections (FF-BDRNN) to provide a framework for modelling both static and dynamic characteristics in a unified fashion.

(ii) **Recalculated state vector modified BPTT learning algorithm**: A learning algorithm that has a significantly reduced storage requirement when compared to the conventional BPTT algorithm is proposed. The algorithm reduces the requirement for state vector storage by recalculating them in the backward pass. Successful implementation of the algorithm requires that the computations are numerically stable and the feedback weight matrix is invertible. In the selected BDRNN architectures the occurrence of numerical instability is significantly reduced due to the reduced interaction between the state variables and invertibility is easily monitored by simple manipulation of the feedback weight matrix elements. More importantly, numerical instability is continuously monitored by a scalar shadow error and when the occurrence of one is detected, numerical stability is restored by performing recovery computations. The invertibility of the feedback weight matrix is ensured by introducing a penalty term in the cost function being minimized during training.

(iii) **Stabilizing technique**: The stability of the FF-BDRNN and its training is ensured by including, in the cost function being minimized during training, a suitable penalty term that is a function of the relative stability of the feedback weight matrix. For this purpose, closed form conditions for local and global stability of the FF-BDRNN are derived using the location of eigenvalues in relation to the unit circle. The relative stability is thus quantified and formulated as a stability function that measures how far this condition is met or violated at each weight update. The stability function is implemented as a constrained feedforward neural network, termed stabilizing feedforward neural network (SFNN), which augments the FF-BDRNN architecture. This approach guides the feedback weight matrix towards a stable configuration while simultaneously reducing the system output error at each weight update.

(iv) **Construction methodology**: The construction methodology is based on the notion that if a nonlinear dynamic trajectory can be decomposed into a combination of a dominant dynamic and a series of progressively less dominant subdynamics, then blocks of FF-BDRNNs can be used with each block employed to learn the dominant dynamic or one of the subdynamics. The construction method uses several cascaded FF-BDRNN modules to construct a composite network whose output is required to approximate the target function being modelled. The FF-BDRNN module at any stage of construction is trained with the scaled residual error between the most recent estimate of the target trajectory and the target trajectory as its desired trajectory.

8.2 CONCLUSIONS ON THE BDRNN / FF-BDRNN ARCHITECTURE

The BDRNN architecture is capable of effectively modelling nonlinear dynamic systems as demonstrated in the examples presented in Chapter 6. This architecture, without feedforward connections, is particularly suited to accurately model trajectories generated by systems with strong autonomous dynamic behaviour. This is demonstrated in the example of section 6.8 of Chapter 6, where a BDRNN accurately generates the co-ordinates of "figure 8" trajectory without any external inputs. This is also demonstrated in the example of section 6.6 of Chapter 6 where a BDRNN models a single-input single-output non-linear plant with a comparable accuracy to, but with a much smaller network size than, that reported previously in literature.

The FF-BDRNN architecture is more suitable to modelling dynamic systems where, in addition to autonomous dynamic behaviour, there exists a strong direct correlation between the system output trajectory and the external inputs. This is demonstrated for the multiple-input multiple-output plant of the example considered in section 6.7 of Chapter 6, where the FF-BDRNN is observed to outperform the BDRNN with an output NRMSE

of less than 50% of the latter. A similar, but better, performance is also observed in the six-step prediction of the Mackey-Glass delay differential equation considered in section 6.9 of Chapter 6, where the FF-BDRNN, with an output NRMSE of 0.03, is observed to clearly outperform the BDRNN whose output NRMSE is 0.33.

The following conclusions are drawn with regard to the two special structures for the BDRNN feedback weight matrix presented: one with scaled orthogonal submatrices and the other with freeform submatrices.

(i) Each submatrix in the freeform feedback weight matrix is suitable for modelling a second order dynamic system with a real pair or a complex-conjugate eigenvalue pair in the linear system sense using four distinct elements.

(ii) Each submatrix in the scaled orthogonal feedback weight matrix is suitable for modelling a second order dynamic system with a complex-conjugate eigenvalue pair in the linear system sense using only two distinct elements.

(iii) The BDRNN with freeform submatrices has a larger degree of freedom in weight assignments than the BDRNN with scaled orthogonal submatrices under local stability conditions and performs marginally better than the latter for a given problem provided that the learning rate is chosen suitably small. However, the global stability condition for the freeform BDRNN imposes more restrictions on the placement of its weights than the corresponding condition for the scaled orthogonal BDRNN. Hence, under global stability conditions the scaled orthogonal BDRNN outperforms the freeform BDRNN for a given problem with a better tolerance to higher learning rates.

8.3 CONCLUSIONS ON THE RECALCULATED STATE VECTOR MODIFIED BPTT ALGORITHM

The block-diagonal structure of the FF-BDRNN is well suited to afford recalculation of the state vectors during the backward pass of the BPTT algorithm. The occurrence of numerical instability normally associated with such recalculation is considerably reduced for the FF-BDRNN in comparison with the fully recurrent DTRNN, as the recurrent connections are limited to between pairs of state variables. Moreover, the block-diagonal structure of the FF-BDRNN is exploited to address various aspects of the state vector recalculation task as follows:

(i) calculating the inverse of the state feedback weight matrix is accomplished using spatially local computations with simple manipulation and scaling of its elements.

(ii) ensuring feedback weight matrix invertibility by implementing the invertibility criterion as a penalty term in the cost function being minimized during training.

(iii) ingraining the invertibility criterion in the FF-BDRNN architecture as a constrained feedforward neural network.

(iv) monitoring and detecting numerical instability using a shadow output error,

(v) restoring numerical stability by performing recovery computations using the intermediate state vectors previously stored during the forwardpass.

It is shown that for this algorithm

(i) the storage requirement for the state vectors is a fraction of that required for conventional BPTT as only intermediate state vectors are stored in the forwardpass

for numerically stable recomputations of the state vectors in the backwardpass. For example, as shown in modelling the Mackey-Glass delay differential equation considered in section 6.9 of chapter 6, no numerical instabilities are encountered when 9% of the state vectors are stored at evenly spaced time intervals in the forwardpass.

(ii) the reduction in storage requirement is achieved by a tradeoff between state vector storage and recovery computations performed to restore numerical stability. For example, as shown in modelling the Mackey-Glass delay differential equation considered in section 6.9 of chapter 6, the storage requirement for intermediate state vectors is reduced to 5%, with an average allowance of less than 5 recovery computations per epoch of length 100.

(iii) the spatially local nature of all computations involved has been retained while still maintaining the exact nature of gradient computation as is the case with conventional BPTT.

8.4 CONCLUSIONS ON STABILIZATION OF FF-BDRNN

The local and global stability conditions for the scaled orthogonal and the freeform BDRNNs are given by:

(i) A BDRNN with scaled orthogonal submatrices is both globally and locally stable if the elements of each of its submatrices satisfy the condition that its determinant is less than or equal to 1.

(ii) A BDRNN with freeform submatrices is globally stable if the elements of each of its submatrices satisfy the condition that the sum of the square of its elements less

the square of its determinant is less than or equal to 1.

(iii) The local stability of a BDRNN with freeform submatrices depends on whether the eigenvalues of each of its submatrices are real or complex conjugates. Restricting the eigenvalues of the submatrix to be real and equal or complex conjugates, the local stability condition is shown to be that the magnitude of the trace should be less than or equal to 2 or that the magnitude of the determinant should be less than or equal to 1, respectively.

The closed form of the local and global stability condition of the scaled orthogonal FF-BDRNN and the global stability condition of the freeform FF-BDRNN allows the formulation of the stability functions in a constrained feedforward neural network framework using first order perceptrons. In the freeform FF-BDRNN with the eigenvalues of its feedback weight matrix restricted to be real and unequal, the neural network implementation of the stability function corresponding to the local stability condition requires the use of higher order perceptrons; however, restricting these eigenvalues to be complex-conjugate pairs allows the use of first order perceptrons in the neural network implementation of the stability function.

Based on the simulation examples presented in Chapter 6, for most cases, the scaled orthogonal BDRNN / FF-BDRNN performs better than or as well as the freeform BDRNN / FF-BDRNN with global stabilization (see e.g., section 6.9 of Chapter 6). This is attributable to the fact that although the freeform architecture has a larger degree of freedom than the scaled orthogonal architecture, the global stability condition imposes more restrictions on the weight placements of the former. On the other hand, with local stability compensation a better output error performance is obtainable with the freeform BDRNN / FF-BDRNN as seen in the example of section 6.5 of Chapter 6 that models the fully recurrent DTRNN plant.

8.5 CONCLUSIONS ON CONSTRUCTIVE TRAJECTORY LEARNING

Based on the example presented in Chapter 7 on modelling the Mackey-Glass delay differential equation, it is seen that training with constructive learning is at least 40% faster than training a single BDRNN / FF-BDRNN. It is also seen that, constructive learning with BDRNN without feedforward connection outperforms learning with a single BDRNN described in section 6.9 of Chapter 6. In the case of constructive learning with FF-BDRNN the output error performance, although is better than the corresponding BDRNN with constructive learning, is marginally worse than that of the single FF-BDRNN described in section 6.9 of Chapter 6. This is attributable to the fact that as the NRMSE of the composite network reduces to small values, the dynamics of the resulting residual error become increasingly difficult to be modelled by the BDRNN / FF-BDRNN modules, hence, resulting in poor performance improvement.

8.6 DIRECTIONS FOR FUTURE RESEARCH

(i) A mathematical characterization of the tradeoff between state vector storage and number of recovery computations performed to restore numerical stability of the BDRNN architecture using stochastic or deterministic methods would be useful. This would result in a methodical study of the numerical instability behaviour and can later be extended to fully connected DTRNN as well.

(ii) It would be worthwhile to perform an in-depth study of the special architectures of the FF-BDRNN (scaled orthogonal and freeform) with particular relevance to their applicability to model target trajectories with special characteristics.

(iii) It would be interesting to rigorously apply the FF-BDRNN architectures to practical problems such as speech recognition and speech generation. With suitable

preprocessing of the speech signal such as dynamic time warping, the modelling capability of these architectures can be better utilized.

REFERENCES

Almeida L.B., (1998) "Backpropagation in perceptrons with feedback", NATO ASI series, Vol. F41, Neural Computers, Edited by R.Eckmiller and Ch. v. d. Malsburg, Springer-Verlag Berlin Heidelberg 1988, pp 199-208.

Angeline P.J., Saunders G.M., Pollack J.B., (1994) "An evolutionary algorithm that constructs recurrent neural networks", IEEE Transactions in Neural Networks, vol 5., no 1., Jan 1994, pp 54-64.

Ash T., (1989) "Dynamic node creation in backpropagation networks", Connection Science, vol. 1, No. 4, pp 365-375, 1989.

Azimi-Sadjadi M.R., Sheedvash S., Trujillo F.O., (1993) "Recursive dynamic node creation in multilayer neural networks", IEEE Transactions on Neural Networks, vol 4, no.2, March 1993, pp 242-56.

Baldi P., (1995) "Gradient descent learning algorithm overview: a general dynamical systems perspective." In IEEE transactions on neural networks, vol. 6, No. 1, Jan. 1995.

Barnes C.W., (1985) "A parametric approach to the realization of second-order digital filter sections," IEEE transactions on circuits and systems, vol. cas-32, No. 6, June 1985.

Beaufays F. and Wan E.A., (1994) "Relating real-time backpropagation and backpropagation through time: an application of flow graph interreciprocity", Neural Computation, Vol. 6, 1994, pp 296-305.

Bruck J, (1990) "On the convergence properties of the Hopfield model", Proceedings of the IEEE, Vol. 78, No. 10, Oct. 1990, pp. 1579-1585.

Cichocki A., Unbehauen R., (1994) Neural networks for optimization and signal processing. John Wiley and sons, New York, 1994.

Day S.P., Davenport M.R., (1993) "Continuous-time temporal back-propagation with adaptive time delays", IEEE transactions on neural networks, vol. 4, No. 2, Mar. 1993.

Draelos T., Hush D., (1996) "A constructive neural network algorithm for functional approximation", International Conference on Neural Networks, Washington D.C., vol, 1, pp. 50-55, 1996.

Elman J.L., (1990) "Finding structure in time", Cognitive Science 14, pp179-211, 1990.

Fahlman S.E., Lebiere C., (1990) "The cascade-correlation learning archietecture", Advances in Neural Information Processing systems 2, 1990. D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp 524-32.

Fahlman S.E., (1991) "The recurrent cascade-correlation architecture", Advances in Neural Information Processing systems 3, 1991. D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp 190-196.

Fang Y., Sejnowski T.J, (1990) "Faster learning for dynamic recurrent backpropagation", Neural Computation, Vol. 2, 1990, pp 270-273.

Frean M., (1990) "The upstart algorithm: A method for constructing and training feedforward neural networks", Neural Computation, vol 2, 1990, pp 198-209.

Giles C.L., Miller C.B., Chen D., Chen H.H., Sun G.Z., Lee Y.C., (1992) "Learning and extracting finite state automata with second-order recurrent neural networks." Neural Computation, 4(3): 393-405, 1992.

Giles C.L., Chen D., Chen H.H., Lee Y.C., and Goudreau M.W., (1995) "Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution", IEEE Transactions on Neural Networks vol. 6, no. 4, July 1995, pp 829-36.

Gori M., Bengio Y., De Mori R., (1989) "BPS: a learning algorithm for capturing the dynamic nature of speech", International joint conference on neural networks 1989, vol. 2, pp 417-423.

Hampshire II J.B., Vijaya kumar B.V.K., (1992) "Shooting craps in search of an optimal strategy for training connectionist pattern classifiers", Advances in Neural Information processing systems 4, 1992, pp 1125-1132.

Hanson S.J., (1990) "Meiosis networks", Advances in Neural Information Processing systems 2, 1990. D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp 533-41.

Haykin S., (1994) Neural networks: A comprehensive foundation, IEEE computer society press, Macmillan New York 1994.

Haykin S., Li L., (1995) "Nonlinear adaptive prediction of nonstationary signals", IEEE Transactions on Signal Processing, vol 43, No.2, pp 526-535, Feb. 1995.

Hirose Y., Yamashita K., Hijiya S., (1991) "Backpropagation algorithm which varies the number of hidden units", Neural Networks, vol 4. 1991, pp 61-66.

Hunt K.J., Sbarbaro D., Zbikowski R., Gawthorp P.J., (1992) "Neural networks for control systems - a survey", Automatica, vol 28, No. 6, pp 1083-1112, 1992.

Hush D.R., Horne B.G., (1993) "Progress in supervised neural networks - What's new since Lippmann ?", IEEE signal processing magazine, Jan 1993, pp 8-39.

Jin L., Nikiforuk P.N., Gupta M.M., (1994) "Absolute stability conditions for discrete-time recurrent neural networks", IEEE Transactions on Neural Networks, Vol. 5, No. 6, Nov. 1994, pp 954-964.

Jin L., Gupta M.M., (1996) "Globally asymptotical stability of discrete-time analog neural networks", IEEE Trans. on Neural Networks, Vol. 7, No. 4, July 1996, pp 1024-1031.

Karnin E.D., (1990) "A simple procedure for pruning back-propagation trained neural networks", IEEE Transactions on Neural Networks, vol 1., June 1990, pp 239-44.

Kelly D.G., (1990) "Stability in contractive nonlinear neural networks", IEEE Transactions on Biomedical Engineering, vol. 37, No. 3, pp 231-242, March 1990.

Kohonen T., The "neural" phonetic typewriter", Computer, March 1988, pp. 11-22.

Kosmatopoulos E.B., Polycarpou M.M., Christodoulou M.A., Ioannou P.A., (1995) "High-order neural network structures for identification of dynamical systems", IEEE Trans. on Neural Networks, Vol. 6, No. 2, March 1995, pp. 422-431.

Ku C.C., Lee K.Y., (1995) "Diagonal recurrent neural networks for dynamic systems control", IEEE Trans. on Neural Networks, Vol. 6, No.1, Jan. 1995, pp. 144-156.

Kung S.Y., (1993) Digital Neural Networks, Chapter 6. Deterministic temporal neural networks. pp 219-224, Prentice Hall, Englewood Cliffs, NJ, 1993.

Li L.K, (1992) "Fixed point analysis for discrete time recurrent neural networks", Proceedings of the Intl. Joint Conference on Neural Networks, June 1992, Vol. 4, pp. 134-139.

Mackey M.C. and Glass L., (1987) "Oscillation and chaos in physiological control systems," Science, vol. 197, pp 287-289, July 1987.

McDonnell J.R., Waagen D., (1994) "Evolving recurrent perceptrons for time-series modeling", in IEEE Transactions on Neural Networks, vol 5, no 1, Jan 1994, pp 24-38.

Moody J.O., Antsaklis P.J., (1996) "The dependence identification neural network construction algorithm", IEEE Transactions on Neural Networks, vol 7., Jan 1996, pp 3-15.

Mozer M.C., Smolensky P., (1989) " Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing systems I, 1989. D. S.Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp 107-115.

Narendra K.S, Parthasarathy K, (1990) "Identification and control of dynamical systems using neural networks", IEEE Trans. on Neural Networks, Vol. 1, No. 1, pp. 4-27, March 1990.

Narendra K.S., Parthasarathy K., (1991) "Gradient methods for optimization of dynamic systems containing neural networks", IEEE transactions on neural networks, vol. 2, No. 2, March 1991.

Nerrand O, Roussel-Ragot P, Urbani D, Personnaz L, Dreyfus. G, (1994) "Training recurrent neural networks: why and how? An illustration in dynamical process modelling", IEEE Trans. on Neural Networks, Vol. 5, No. 2, March 1994, pp. 178-184.

Obradovic D., "On-line training of recurrent neural networks with continuous topology adaptation", in IEEE Transactions on Neural Networks, vol 7, no 1, Jan 1996, pp 222-28.

Parisi R., Di Claudio E.D., Rapagnetta A., Orlandi G., (1996) "Recursive least squares approach to learning in recurrent neural networks", Proceedings of the International Conference on Neural Networks 1996, Vol. 2, pp. 1350-1354.

Pearlmutter B.A, (1989) "Learning state-space trajectories in recurrent neural networks", Neural Computation, Vol. 1, 1989, pp. 263-269.

Pearlmutter B.A., (1995) "Gradient calculations for dynamic recurrent neural networks: a survey", IEEE transactions on neural networks, vol. 6, No. 5, Sept. 1995.

Pineda F.J, (1989) "Recurrent backpropagation and the dynamical approach to adaptive neural computation", Neural Computation, Vol. 1, 1989, pp. 161-172.

Rabiner L.R., (1989) "A tutorial on Hidden Markov Models and selected applications in speech recognition", Proceedings of the IEEE, pp 257-285, Feb 1989.

Rabiner L.R., Juang B.H., (1993) Fundamentals of Speech Recognition, Prentice Hall Signal Processing series Englewood Cliffs, New Jersey 07632, 1993.

Sanger T.D., (1991) "A tree structured adaptive network for function approximation in high dimensional spaces", IEEE transactions on neural networks, vol. 2, No. 2, Mar. 1991.

Sato M, (1990) "A real time learning algorithm for recurrent analog neural networks", Biological Cybernetics, Vol. 62, 1990, pp. 237-241.

Schmidhuber J., (1992) "A fixed size storage O(n3) time complexity learning algorithm for fully recurrent continually running networks", Neural Computation, vol 4, 1992, pp 243-248.

Simard P.Y., Rayzs J.P., Victorri B., (1990) "Shaping the state space landscape in recurrent networks," in Advances in Neural Information processing systems 3, 1991, pp 105-112.

Sivakumar S.C., (1992a) "Consonant recognition using neural networks", M.A.Sc Thesis in Electrical Engineering, Technical University of Nova Scotia, Halifax, Nova Scotia.

Sivakumar S.C., Robertson W., MacLeod K., (1992b) "Improving temporal representation in TDNN structure for phoneme recognition", International Joint Conference on Neural Networks, Washington D.C., vol. 4, pp 728-733, June 1992.

Sivakumar S.C., Robertson W., Phillips W.J., (1992c) "Enhancing neural network based consonant recognition using formant transition in neighboring vowels", International Symposium on Information Technology and Applications, Singapore, vol. 3, pp , 1992.

Sivakumar S.C., Robertson W., Phillips W.J., (1995) "On-line stabilization of block-diagonal recurrent neural networks", Submitted to IEEE Transactions on Neural Networks. October 1995.

Sivakumar S.C., Robertson W., Phillips W.J., (1996a) "A modified BPTT algorithm for trajectory learning in block-diagonal recurrent neural networks", Submitted to 1997 Canadian Conference on Electrical and Computer Engineering, October 1996.

Sivakumar S.C., Robertson W., Phillips W.J., (1996b) "A cascading constructive trajectory learning algorithm for block-diagonal recurrent neural networks", Submitted to 1997 Canadian Conference on Electrical and Computer Engineering, October 1996.

Stewart G.W., Introduction to Matrix Computations, Academic Press Inc., San Diego, 1973.

Sun G.Z., Chen H.H., Lee Y.C., Giles C.L., (1991) "Turing equivalence of neural networks with second order connection weights." In Proceedings of the International Joint Conference on Neural Networks, vol 2, pp 357-362, 1991.

Sun G.Z., Chen H.H., Lee Y.C., (1992) "Green's function method for fast on-line learning algorithm of recurrent neural networks," Advances in Neural Information processing systems 4, 1992, pp 333-340.

TIDIGITS, (1991) Studio Quality Speaker-Independent Connected-Digit Corpus with ISO 9660 Data Format on CD-ROM Media, The National Institute of Standards and Technology, Made in USA by American Helix, Feb. 1991

Tsoi A.C., Back A.D., (1994) "Locally recurrent globally feedforward networks: a critical review of architectures", IEEE transactions on neural networks, vol. 5, No. 2, Mar. 1994.

Uchiyama T., Shimohara., Tokunaga Y., (1989) " A modified leaky integrator network for temporal pattern processing", International joint conference on neural networks 1989, vol. 1, pp 469-475.

Unnikrishnan K.P., Hopfield J., and Tank D., (1991) "Connected digit speaker dependent speech recognition using a neural network with time delayed connections," IEEE Trans. on Signal Processing, Vol. 39, 1991, pp 698-713.

Vidyasagar M, (1993) "Location and stability of the high-gain equilibria of nonlinear neural networks", IEEE Trans. on Neural Networks, Vol. 4, No. 4, July 1993, pp. 660-672.

Waibel A., Hanazawa T., Hinton G., Shikano K., Lang K., (1989) "Phoneme recognition using time-delay neural networks," IEEE Trans. on Acoustics, Speech and Signal Processing, vol. 37, No. 3, 1989, pp 328-339.

Werbos P.J, (1990) "Backpropagation through time: What it does and how to do it", Proceedings of the IEEE, Vol. 78, No. 10, Oct. 1990, pp. 1550-1560.

Widrow B., (1990) "30 years of adaptive neural networks: perceptron, madaline, and backpropagation", Proceedings of the IEEE, pp. 1415-1441, Sept. 1990.

Williams R.J., Peng J., (1990) "An efficient gradient-based algorithm for on-line training of recurrent network trajectories", Neural Computation, vol. 2, 1990, pp 490-501.

Williams R.J, Zipser.D, (1989) "A learning algorithm for continually running fully recurrent neural networks", Neural Computation, Vol. 1, 1989, pp. 270-280.

IMAGE EVALUATION TEST TARGET (QA-3)

150mm

6″

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989