Asymptotitically Stable Recurrent Neural Networks: Theory and Application

by

John Tadeusz Dorocicz
B. Eng., University of Victoria, 1994

A Thesis Submitted in Partial Fulfillment of the
Requirements of the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

[redacted]

Dr. N.J. Dimopoulos, Supervisor (Department of Electrical & Computer Engineering)

[redacted]

Dr. W-S. Lu, Departmental Member (Department of Electrical & Computer Engineering)

[redacted]

Dr. Z. Dong, Outside Member (Department of Mechanical Engineering)

[redacted]

Dr. M. Serra, External Examiner (Department of Computer Science)

University of Victoria

Supervisor: Dr. Nikitas J. Dimopoulos

ABSTRACT

This thesis will outline the basis for some types of artificial neural networks. In particular a special type of recurrent neural network that has been proven to be asymptotically stable will be studied. The theory required to train such a neural network will be presented. A number of heuristics designed to improve the training of the asymptotically stable recurrent neural network and the results of these heuristics on a variety of data sets will be presented. An application for the asymptotically stable recurrent neural network involving the monitoring of cable television trunk amplifiers will also be described.

Examiners:

[redacted]

Dr. N.J. Dimopoulos, Supervisor (Department of Electrical & Computer Engineering)

[redacted]

Dr. W-S. Lu, Departmental Member (Department of Electrical & Computer Engineering)

[redacted]

Dr. Z. Dong, Outside Member (Department of Mechanical Engineering)

[redacted]

Dr. M. Serra, External Examiner (Department of Computer Science)

Table of Contents

List of Tables

List of Figures

Acknowledgements

The author would like to acknowledge the assistance provided by Dr. N.J. Dimopoulos, and Mr. S.W. Neville in the preparation of this thesis.

The author would also like to acknowledge that this research would not be possible without support from the Canadian Cable Labs Fund and NSERC.

Chapter 1

Introduction

Artificial neural networks have found many applications in recent years [18]. Some artificial neural networks are based on the biological neural networks found in the brain. This chapter describes the biological basis for these artificial neural networks as well as the historical development of artificial neural networks. This chapter also describes some of the various types of artificial neural networks, and some applications for these neural networks.

1.1 Biological Inspiration

A biological neural network consists of groups of neurons (or nerve cells) interconnected through weighted interconnections. The human brain is believed to consist of at least ten billion such neurons [1]. The main parts of a neuron are the dendrites, the soma, the hillock, the axon, and the synapse as shown in Fig. 1.1.

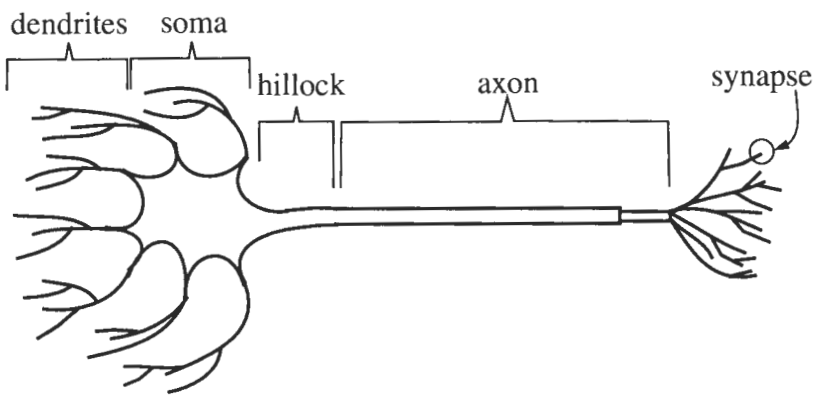

FIGURE 1.1 Sample neuron.

The dendrites receive the input to the neuron. A single neuron can have hundreds to thousands of inputs. The soma does a temporal and spatial summation of the inputs. This summation results in the membrane potential of the neuron. If the membrane potential exceeds a threshold, then a spike (or action potential) is initiated at the hillock. This spike propagates along the axon and affects the dendrites to which the axon is connected. Neurons

interconnect with each other through a structure called a synapse. Chemical synapses[1] transmit information from one neuron to another through the use of neuro-transmitters as illustrated in Fig. 1.2.

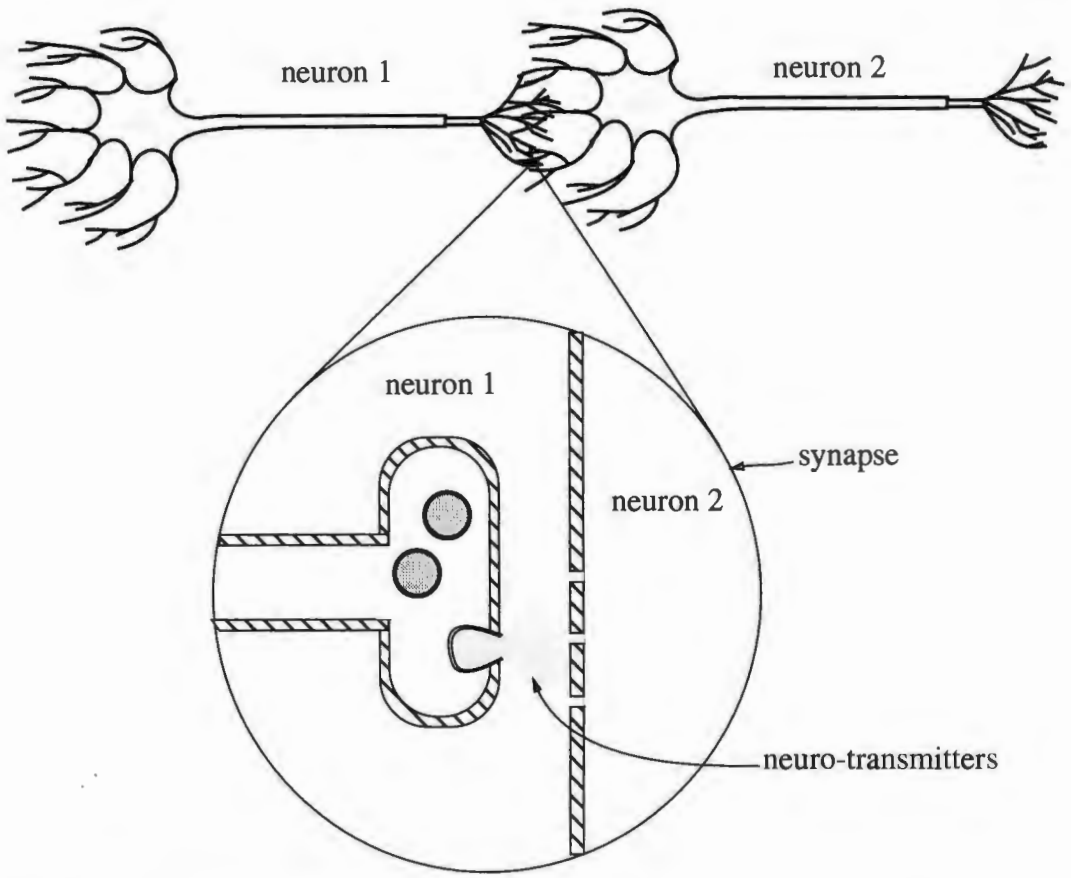

FIGURE 1.2 Chemical synaptic contact.

Synapses are excitatory or inhibitory, but not both. Excitatory synapses contribute to the firing of the target neuron, while inhibitory synapses try to prevent the target neuron from firing.

If the stimulus persists, repeated action potentials are generated as shown in Fig. 1.3.

1. Note that chemical synapses are not the only way neurons interact with each other. Besides chemical synapses there are electrical synapses, and electrochemical synapses.

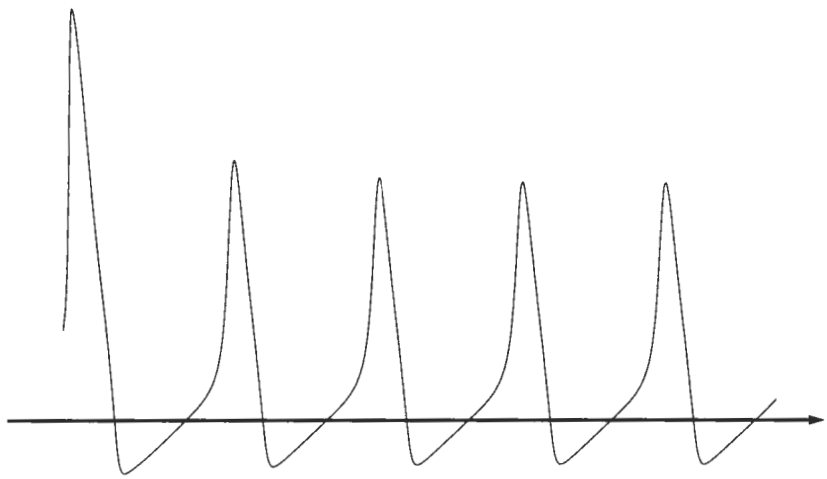

FIGURE 1.3 Repeated action potentials.

The threshold of the second and subsequent action potentials are higher than that of the first one; i.e. the stimulus needs to increase for the second and subsequent spikes as shown in Fig. 1.4.

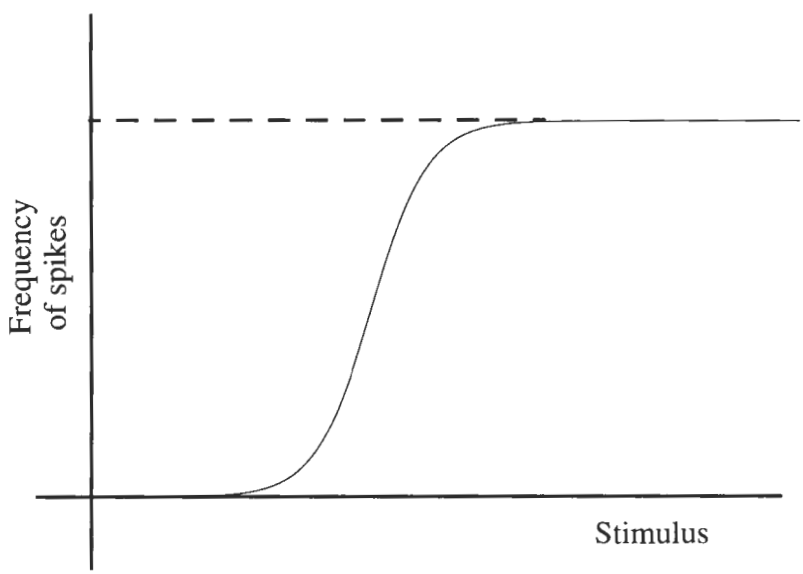

FIGURE 1.4 Threshold function for repeated action potentials.

A natural limit to the firing rate is known as the refractory period. The refractory period imposes an upper bound on the rate a neuron can fire.

Given this biological basis for the neuron, let us examine how we can model this process. This process was originally modelled by Hodgkin and Huxley [8]. We can model the accumulation of the action potentials as

$$v = v_o + v_{\text{syn}} e^{-\alpha t} \tag{1.1}$$

where v_o is the steady state potential, v_{syn} is the synaptic potential, α is a constant, and t is time. Eq. (1.1) indicates that the elicted action potentials decay exponentially with time. The value of v_{syn} depends on the frequency of spikes arriving at the dendrites. Let $f_i(t)$ be the frequency of spikes arriving at the i^{th} synapse and $h_i(t)$ be the effect of a spike arriving at the i^{th} synapse, on the hillock. Given Eq. (1.1) we can write

$$v(t) = v_o + \sum_{i=1}^{k} \int_{-\infty}^{t} h_i(t-\tau) f_i(\tau)\, d\tau \tag{1.2}$$

As the postsynaptic potentials behave exponentially, $h_i(t)$ can be expressed as

$$h_i(t) = w_i e^{-\frac{t}{\mu}} u(t) \tag{1.3}$$

where $u(t)$ is a unit step function. By combining Eqs. (1.2) and (1.3) a neuron can be modelled in the time domain by

$$v(t) - v_o = \sum_{i=1}^{k} \int_{0}^{t} f_i(\tau)\, w_i e^{-\frac{(t-\tau)}{\mu}}\, d\tau \tag{1.4}$$

Differentiating and simplifying yields

$$\mu \frac{d\tilde{v}}{dt} + \tilde{v} = \sum_{i=1}^{k} w_i f_i(t) \tag{1.5}$$

where $\tilde{v} = v - v_o$. As $f_i()$ relates to the potential of the i^{th} neuron through the sigmoid nonlinearity shown in Fig. 1.4, we can generalize by writing

$$\mu \frac{dv_i}{dt} + v_i = \sum_{j=1}^{k} w_{ij} f_j(v_j) \tag{1.6}$$

where v_i is the voltage at the hillock of neuron i and $f_j(v_j)$ is the frequency of spikes from neuron j with respect to the voltage at the hillock of neuron j. Eq. (1.6) gives the differential equation for the neuron that is used in recurrent artificial neural networks.

1.2 Types of Artificial Neural Networks

The first artificial neural networks were mostly attempts by researchers to understand how the human brain and neurons might actually work, rather than problem solving techniques. The first example of an artificial neural network is the McCulloch-Pitts neuron [20]. The paper by McCulloch and Pitts is one of the first attempts to abstract the properties of the biological neuron. The 'McCulloch-Pitts' neuron as it is known, is a simple binary device with a fixed threshold. McCulloch and Pitts showed that these neurons could be constructed to compute any logical function.

Interest in artificial neural networks continued with the advent of learning algorithms. Learning in artificial neural networks drew heavily from the work of psychologist D.O. Hebb [20]. Hebb theorized that the activity of a neuron increased its sensitivity and the sensitivity of neighboring neurons to that particular stimulus.

A significant step in artificial neural networks was brought forth by the perceptron and the perceptron learning algorithm devised by Frank Rosenblat. The perceptron is a single layer network with only one set of weighted arcs as illustrated in Fig. 1.5. (Note that the activation functions are linear). A single layer network of perceptrons as shown in Fig. 1.5 initially created a great deal of excitement as the perceptron was a neural model that was capable of learning complex behavior.

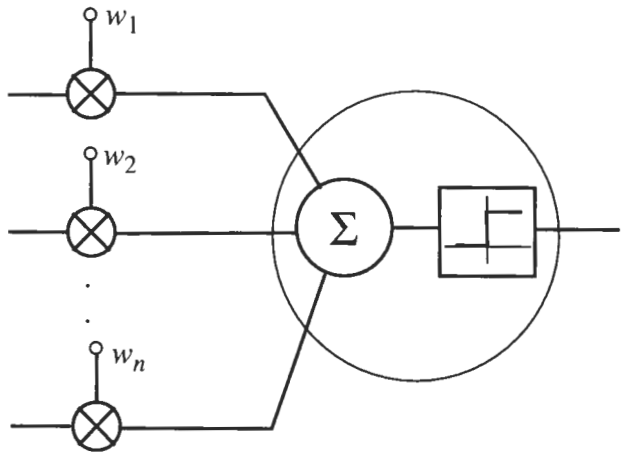

FIGURE 1.5 Example single layer perceptron network.

Interest in neural networks wained when it was proven by Minsky and Papart that perceptrons could only solve a narrow class of problems [20]. Single perceptrons could only differentiate between linearly separable problems; a well known example of a linearly inseparable function is the exclusive-or function. Although multilayer perceptron networks with nonlinear threshold functions as shown in Fig. 1.6 could overcome the problems associated with single layer perceptrons, these networks were generally not usable in the past as there was no straight forward method for training them.

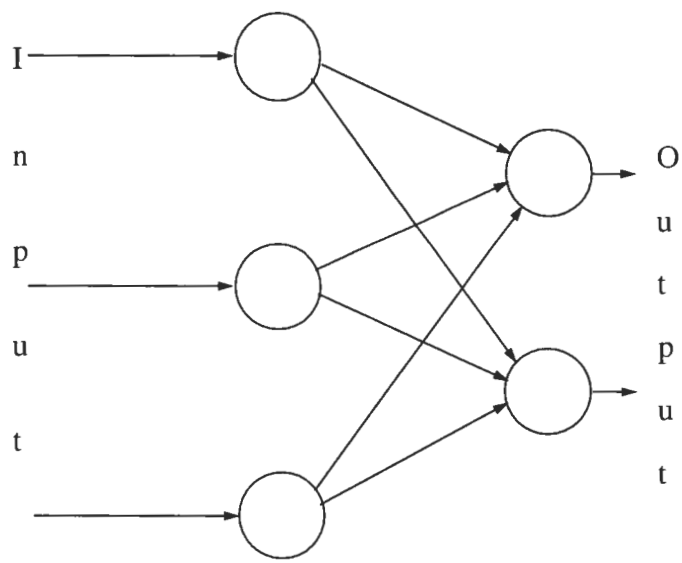

FIGURE 1.6 Example multilayer perceptron network; arrows indicate direction of signal propagation.

The difficulty in training such an artificial neural network is adjusting the weights of the neurons not directly connected to the output. The neural network shown in Fig. 1.6 is also known as a feedforward neural network.

After Minsky and Papart's book, interest in artificial neural networks remained low until 1982 when Hopfield networks came out. Hopfield's paper on artificial neural networks presented a complete theoretical description of how an artificial neural network could operate [12]. Hopfield artificial neural networks are recurrent neural networks in that the neurons can connect to all other neurons with the expectation of self loops i.e. feedback exists within the neural network as shown in Fig. 1.7.

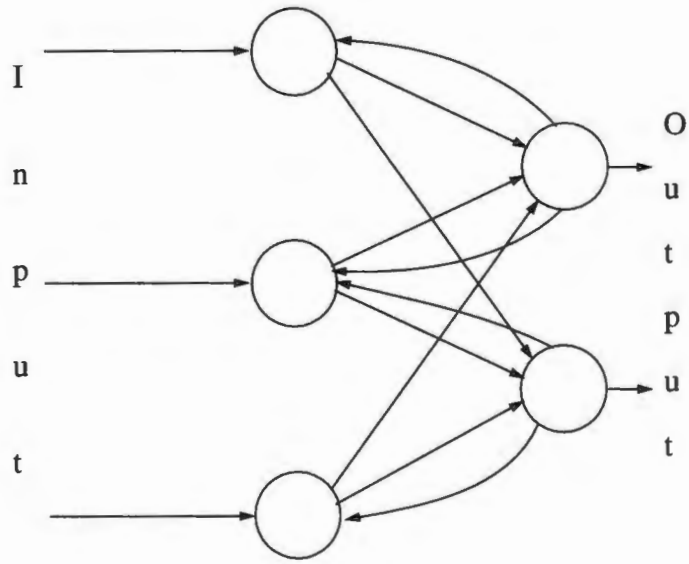

FIGURE 1.7 Example multilayer perceptron network with feedback; arrows indicate direction of signal propagation.

Hopfield neural networks store 'states' in local minima. Note that when feedback is introduced into a system, system stability becomes an important consideration. By ensuring a symmetric weighted connection matrix, a Hopfield neural network is stable.

With the advent of back propagation in 1986, multilayered perceptron networks became feasible [1]. Hopfield neural networks and back propagation renewed interested in research into neural networks. Some examples of such research, are using neural networks for speech recognition [16], pattern recognition [14], and modelling dynamic systems

[9][15] [17][21]. Neural networks are also being implemented in silicon [11]. Note that feedforward neural networks are best at pattern recognition, while recurrent neural networks are best at modelling dynamic systems. The focus of this thesis will be on recurrent neural networks.

1.3 Summary

This chapter provided background theory relating to artificial neural networks. This thesis will examine the training and simulation of a special type of recurrent neural network, as well as the modelling capabilities of such a neural network. The remainder of this thesis is divided into the following chapters:

Ch. 2 describes an asymptotically stable form of recurrent neural networks derived by Dimopoulos. This chapter also derives how to train such a neural network and how such a neural network can be applied to model nonlinear dynamic systems. An original contribution is made with the derivation of the training of the time membrane constants (Sec. 2.3.3).

Ch. 3 gives a brief description of an asymptotically stable recurrent neural network simulator and some improvements made to the simulator. This chapter also describes some heuristics used to improve the training of the simulator as well as the data sets used to test the simulator.

Ch. 4 details an industrial application for the asymptotically stable recurrent neural network. The neural network is used to monitor a cable television amplifier for behavior changes.

Ch. 5 describes how the asymptotically stable recurrent neural network can be used for long-term monitoring of cable television trunk amplifiers.

Ch. 6 presents a summary of the work covered by this thesis, as well as recommendations for future work.

1.4 Trademarks

Because of the software packages employed in developing this thesis, there are far too many instances of trademarks to appropriately note each one when used. Therefore, the following terms are trademarks of their respective owners:

UNIX is a trademark of American Telephone and Telegraph, Inc.

MATLAB is a trademark of The MathWorks, Inc.

C-COR and **C-COR Electronics** are trademarks of C-COR Electronics, Inc.

HP is a trademark of Hewlett Packard Co.

Sun, Sparc, and Sparc Station are trademarks of Sun microsystems, Inc.

Chapter 2

Recurrent Neural Networks

Recurrent neural networks are neural networks that have feedback; whenever feedback is present in a system, stability becomes a concern. Although few researchers have addressed the issue of stability in recurrent neural networks, stability must be considered if recurrent neural networks are used to model dynamic systems. This chapter will study a special type of recurrent neural network that has been proven to be asymptotically stable. By asymptotically stable we mean, if the input to the neural network is set to zero, then the output of the neural network will decay to zero as time goes to infinity. This chapter will also describe how to train such an asymptotically stable neural network using back propagation with steepest descent. Finally, we will show how nonlinear dynamic systems can be modelled using recurrent neural networks. Note that the terms layers and classes will be used interchangeably in this Chapter.

2.1 Recurrent Neural Networks

A recurrent neural network consisting of k neural classes can be described by

$$\dot{\mathbf{O}} = -\mathbf{TO} + \mathbf{W}f(\mathbf{O}) + \mathbf{b}(t) \tag{2.1}$$

where

$$\mathbf{O} = \begin{bmatrix} \mathbf{O}_1 & \mathbf{O}_2 & \dots & \mathbf{O}_k \end{bmatrix}^T \tag{2.2}$$

is the state of the neural network,

$$\mathbf{T} = \begin{bmatrix} \tau_1 & 0 & \dots & 0 \\ 0 & \tau_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \tau_k \end{bmatrix} \tag{2.3}$$

are the membrane time constants, and

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{W}_{12} & \cdots & \mathbf{W}_{1k} \\ \mathbf{W}_{21} & \mathbf{W}_{22} & \cdots & \mathbf{W}_{2k} \\ \vdots & \vdots & & \vdots \\ \mathbf{W}_{k1} & \mathbf{W}_{k2} & \cdots & \mathbf{W}_{kk} \end{bmatrix} \tag{2.4}$$

indicates the weighted connectivity of the neural network and how the classes (layers) are interconnected. $f()$ is the threshold function, and $\mathbf{b}(t)$ is a N x 1 vector that describes the input to the neural network. If there are k neural classes with n_i neurons in each class where $i = 1, 2,...k$ then the total number of neurons in the network is

$$N = \sum_{i=1}^{k} n_i \tag{2.5}$$

The recurrent neural network given in Eq. (2.1) has been proven to be asymptotically stable if certain conditions in the connectivity matrix given in Eq. (2.4) are met [7]. The next section will review the stability proof for the recurrent neural networks derived by Dimopoulos.

2.2 Dimopoulos Stability Criterion for Recurrent Neural Networks

This section will briefly review the stability proof for asymptotically stable recurrent neural networks. Polyexponential functions are used in this proof. Before we examine the proof, several definitions are required first. Neurons are assumed to belong to morphologically distinct classes. Neurons in each class have similar properties (i.e. a class cannot have both excitatory and inhibitory neurons). Neurons in a class connect to neurons of other classes in a predetermined way; this is known as the macroscopic connectivity principle. Neurons connect to neurons (subject to the provisions of the macroscopic connectivity principle) that are physically close to them; this is known as the microscopic connectivity principle.

The basic equation that describes the activation of the asymptotically stable recurrent neural network is given in Eq. (2.1). The vector **O** in Eq. (2.2) gives the 'state' of the neural network. The connectivity matrix **W** given in Eq. (2.4) can be interpreted as fol-

lows: $\mathbf{W}_{ij}$ is zero if the neurons from class i do not receive input from neurons from class j. $\mathbf{W}_{ij}$ is positive if class j is a class of 'excitatory' neurons; $\mathbf{W}_{ij}$ is negative if class j is a class of 'inhibitory' neurons. The function $f()$ describes the transformation of the hillock potential to a frequency of action potentials propagating along the axon. The function $f()$ is defined as a continuous, monotonically nondecreasing function that satisfies a Lipschitz condition. Also $\exists\theta \in \Re^N$ such that $f(\theta) = 0$. An example of such a function is illustrated in Fig. 2.1.

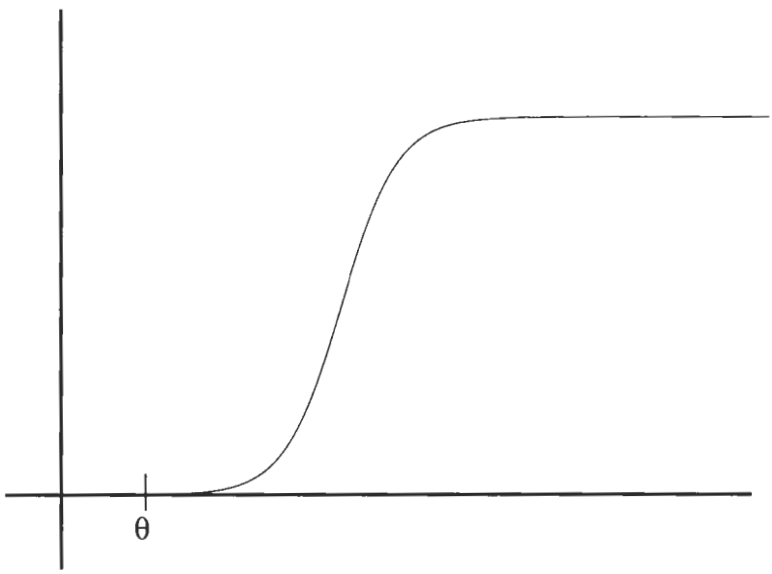

FIGURE 2.1 Example function $f()$.

The function shown in Fig. 2.1 is a sigmoid like function.

Given the definitions in the two previous paragraphs, it can be proven that recurrent neural networks having connectivity matrices where all the excitatory submatrices $\mathbf{W}_{ij}$ are located on the same side of the diagonal, exhibit behavior that is bounded above and below by functions decaying exponentially in time. Furthermore if the thresholds θ are positive, then both bounds reach zero as shown in Fig. 2.2.

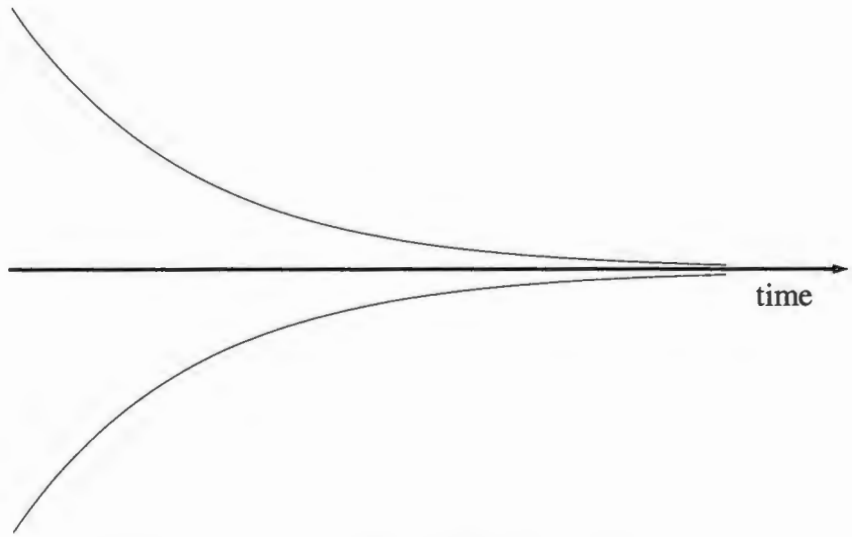

FIGURE 2.2 Asymptotic bounds for recurrent neural network.

In other words, a recurrent neural network will be asymptotically stable if the feedback is inhibitory. In this thesis however, we will only prove that a recurrent neural network based on the cerebellum neural network, shown in Fig. 2.3 is stable.

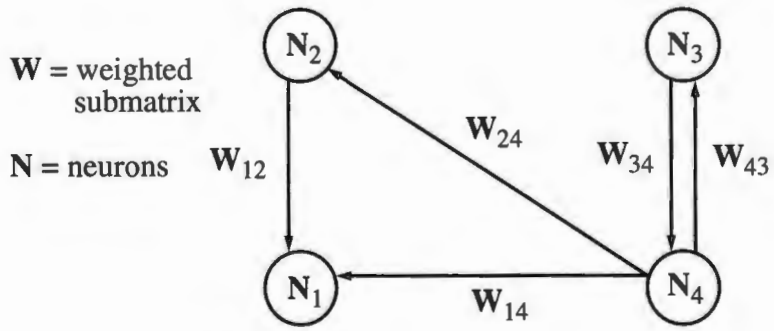

FIGURE 2.3 Cerebellum neural network.

The general stability proof is given in [7]. The connectivity matrix **W** for the network in Fig. 2.3 is

$$\mathbf{W} = \begin{bmatrix} 0 & \mathbf{W}_{12} & 0 & \mathbf{W}_{14} \\ 0 & 0 & 0 & \mathbf{W}_{24} \\ 0 & 0 & 0 & \mathbf{W}_{34} \\ 0 & 0 & \mathbf{W}_{43} & 0 \end{bmatrix} \tag{2.6}$$

where $\mathbf{W}_{12}$ and $\mathbf{W}_{43}$ are inhibitory weight matrices. For the cerebellar network we can write

$$\mathbf{T}_1 \dot{\mathbf{O}}_1 + \mathbf{O}_1 = -\mathbf{W}_{12} f(\mathbf{O}_2) + \mathbf{W}_{14} f(\mathbf{O}_4) \tag{2.7}$$

$$\mathbf{T}_2 \dot{\mathbf{O}}_2 + \mathbf{O}_2 = \mathbf{W}_{24} f(\mathbf{O}_4) \tag{2.8}$$

$$\mathbf{T}_3 \dot{\mathbf{O}}_3 + \mathbf{O}_3 = \mathbf{W}_{34} f(\mathbf{O}_4) \tag{2.9}$$

$$\mathbf{T}_4 \dot{\mathbf{O}}_4 + \mathbf{O}_4 = -\mathbf{W}_{43} f(\mathbf{O}_3) \tag{2.10}$$

To prove that Eqs. (2.7) to (2.10) represent an asymptotically stable system, several Lemmas (without proof) are presented. (For proofs see [7]).

Lemma 2.1: For every function $f()$, every vector polynomial $P(\bullet)$ in t, and all positive diagonal matrices A and B there exists a vector polynomial $\hat{P}(\bullet)$ in t and a positive diagonal matrix $\hat{B}$ such that

$$0 \le e^{-At}\left(\int_0^t e^{As} f\left(e^{-Bs} P(s)\right)\right) ds \le e^{-\hat{B}t} \hat{P}(t) \tag{2.11}$$

is true.

Lemma 2.2: Given Lemma 2.1, the system

$$\mathbf{T}_1 \dot{\mathbf{O}}_1 + \mathbf{O}_1 = f_1(\mathbf{O}_2) \tag{2.12}$$

$$\mathbf{T}_2 \dot{\mathbf{O}}_2 + \mathbf{O}_2 = -f_2(\mathbf{O}_1) \tag{2.13}$$

is asymptotically stable in the large.

Lemma 2.3: Given Lemma 2.1 and

$$e^{-\tilde{T}t}\tilde{P}(t) \leq x_3 \leq e^{-\hat{T}t}\hat{P}(t) \tag{2.14}$$

the system

$$\mathbf{T}_1 \dot{\mathbf{O}}_1 + \mathbf{O}_1 = f_1(\mathbf{O}_3) \tag{2.15}$$

$$\mathbf{T}_2 \dot{\mathbf{O}}_2 + \mathbf{O}_2 = -f_2(\mathbf{O}_2) + f_3(\mathbf{O}_3) \tag{2.16}$$

is asymptotically stable in the large.

Theorem 2.1: Given Lemma 2.3, the system given in Eqs. (2.7) and (2.8) is asymptotically stable in the large. Given Lemma 2.2, the system given in Eqs. (2.9) and (2.10) is asymptotically stable in the large.

Having reviewed in part Dimopoulos' stability proof, let us look at how we can train such a neural network.

2.3 Training a Recurrent Neural Network

The goal of training a neural network is to get the neural network to exhibit a desired behavior. In the case of a recurrent neural network, we want it to model a dynamic nonlinear system. (Note that Sec. 2.4 describes other methods that can be used to model a dynamic nonlinear system). Fig. 2.4 gives a conceptual visualization of training a recurrent neural network to model a system.

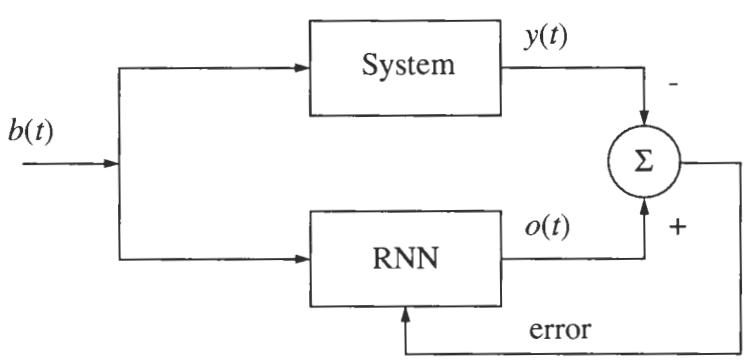

FIGURE 2.4 Training a recurrent neural network.

The recurrent neural network described by Eq. (2.1) is trained by adjusting network parameters such that an error function (defined for the neural network) is minimized. The error (or cost) function for a recurrent neural network defined by Eq. (2.1), is

$$J = \frac{1}{2}(\mathbf{O} - \mathbf{O}_d)^T \mathbf{A} (\mathbf{O} - \mathbf{O}_d) \tag{2.17}$$

where $\mathbf{O}_d$ is the desired state of the recurrent neural network, $\mathbf{O}$ is the actual state of the recurrent neural network, and $\mathbf{A}$ is a matrix used to remove unneeded states. If we assume that the output layer has only one neuron (see Sec. 3.0), then Eq. (2.17) simplifies to

$$J = \frac{1}{2}(o_j - o_{dj})^2 \tag{2.18}$$

where o_j is the activation (output) of the neuron in the output layer, o_{dj} is the desired output, and j is a convenient index.

The most commonly used method to minimize the error function of a neural network is the steepest descent method (SDM) [26]; i.e. the parameters in the neural network are adjusted in a direction opposite to the gradient of the error function (see [3] for more details on SDM). This is mathematically expressed as

$$\frac{d\theta}{dt} = -\eta \frac{\partial J}{\partial \theta} \tag{2.19}$$

where θ is the parameter that is being adjusted, η is the rate of learning, and t is time. The problem of training the neural network now becomes that of finding the value for $\frac{\partial J}{\partial \theta}$. Using the chain rule, the basic equation for calculating the gradient can be expressed as

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial o_j} \frac{\partial o_j}{\partial \theta} \tag{2.20}$$

where o_j is the activation of the neuron j. From Eq. (2.20) the values of $\frac{\partial J}{\partial o_j}$, and $\frac{\partial o_j}{\partial \theta}$ need to be found; the value of $\frac{\partial J}{\partial o_j}$ will be found first.

When adjusting the neural network parameters, the training depends on whether the parameters belong to the output neuron or a non-output neuron. In the case where j is the output neuron, we can write

$$\frac{\partial J}{\partial o_j} = o_j - o_{dj} \tag{2.21}$$

(i.e. the derivative of Eq. (2.18)). For adjusting the parameters of a non-output neuron, $\frac{\partial J}{\partial o_j}$ is found indirectly using gradients previously calculated. This method is analogous to back propagation in feedforward neural networks. For non-output neurons

$$\frac{\partial J}{\partial o_j} = \sum_k \frac{\partial J}{\partial o_k}\frac{\partial o_k}{\partial o_j} \tag{2.22}$$

where o_j is the activation of neuron j, o_k is the activation of neuron k, k is an index that goes through all the neurons that have j as an input, and $\frac{\partial J}{\partial o_k}$ is assumed known. A conceptualization of back propagation is shown in Fig. 2.5.

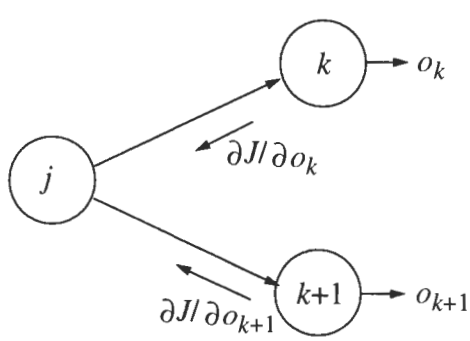

FIGURE 2.5 Back propagation example.

Now the value for $\frac{\partial o_k}{\partial o_j}$ needs to be found. Let $\Delta_{kj} = \frac{\partial o_k}{\partial o_j}$. By using Eq. (2.1) the value of Δ_{kj} can be found. Rewriting Eq. (2.1) in terms of neuron k yields

$$\tau_k o_k + \dot{o}_k = \sum_j w_{kj} f_k(o_j) + b_k(t) \tag{2.23}$$

(Note that w_{kj} means neuron j is an input to neuron k). If we assume that the only external inputs are made to the input layer of the recurrent neural network, then Eq. (2.23) simplifies to

$$\tau_k o_k + \dot{o}_k = \sum_j w_{kj} f_k(o_j) \tag{2.24}$$

for non-input layer neurons and

$$o_k = b_k(t) \tag{2.25}$$

for input layer neurons. If Eq. (2.24) is differentiated with respect to o_j, then

$$\tau_k \frac{\partial o_k}{\partial o_j} + \frac{\partial \dot{o}_k}{\partial o_j} = w_{kj} \frac{\partial f_k(o_j)}{\partial o_j} \tag{2.26}$$

or

$$\dot{\Delta}_{kj} = -\tau_k \Delta_{kj} + w_{kj} \frac{\partial f_k(o_j)}{\partial o_j} \tag{2.27}$$

Solving this differential equation for Δ_{kj} will yield $\frac{\partial o_k}{\partial o_j}$.

Finding $\frac{\partial o_j}{\partial \theta}$ depends on the parameter being adjusted. As we can adjust the weights, the sigmoid shape constants, and membrane time constants, the next three sections show how to solve $\frac{\partial o_j}{\partial \theta}$ for each of these parameters.

2.3.1 Adjustment of Weights

If θ is an interconnection weight w_{ji}, then $\frac{\partial o_j}{\partial w_{ji}}$ needs to be found. Let $\xi_{ji} = \frac{\partial o_j}{\partial w_{ji}}$. If we express Eq. (2.1) in terms of neuron j we get

$$\tau_j o_j + \dot{o}_j = \sum_k w_{jk} f_j(o_k) + b_j \tag{2.28}$$

Differentiating Eq. (2.28) with respect to w_{ji} yields

$$\tau_j \frac{\partial o_j}{\partial w_{ji}} + \frac{\partial \dot{o}_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(\sum_k w_{jk} f_j(o_k) \right) \tag{2.29}$$

or

$$\dot{\xi}_{ji} = -\tau_j \xi_{ji} + f_j(o_i) \tag{2.30}$$

as $\frac{\partial}{\partial w_{ji}}\left(\sum_k w_{jk} f_j(o_k) \right)$ is nonzero only when $i = k$. Solving this differential equation for ξ_{ji} will yield the value of $\frac{\partial o_j}{\partial w_{ji}}$. Thus the weight adjustment formula for the output neuron is

$$\frac{dw_{ji}}{dt} = -\eta (o_j - o_{dj}) \xi_{ji} \tag{2.31}$$

and for non-output neurons is

$$\frac{dw_{ji}}{dt} = -\eta \left(\sum_k \frac{\partial J}{\partial o_k} \Delta_{kj} \right) \xi_{ji} \tag{2.32}$$

It is important to note that weight clamping is required to ensure that the stability criteria are not violated. In other words, weights that are labeled as inhibitory must be checked during training so that the training does not make the weights positive. If the weights do become positive during training, the weights are clamped at zero to ensure stability.

2.3.2 Adjustment of Sigmoid Shape Constants

The shape constant parameter is used to control the shape of the sigmoid function. A truncated sigmoid function is used as the activation function for the neurons. This truncated sigmoid function is

$$f(o) = \begin{cases} \dfrac{2}{1+e^{-\sigma o}} - 1 & o \geq 0 \\ 0 & o < 0 \end{cases} \tag{2.33}$$

The value of σ controls the shape of the sigmoid as illustrated in Fig. 2.6.

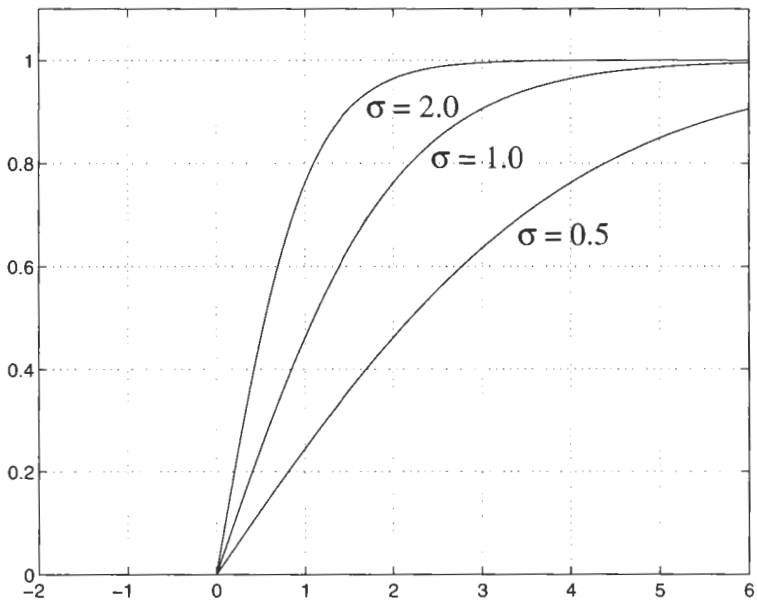

FIGURE 2.6 Example of truncated sigmoid functions.

To derive how to adjust σ, let $\nu_j = \dfrac{\partial o_j}{\partial \sigma_j}$. Taking the derivative of Eq. (2.28) with respect to σ_j yields

$$\tau_j \frac{\partial o_j}{\partial \sigma_j} + \frac{\partial o_j}{\partial \sigma_j} = \sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \tag{2.34}$$

or

$$\dot{\nu}_j = -\tau_j \nu_j + \sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \tag{2.35}$$

Solving this differential equation for ν will yield the value of $\frac{\partial o_j}{\partial \sigma_j}$. Thus the shape constant adjustment formula for the output neuron is

$$\frac{d\sigma_j}{dt} = -\eta (o_j - o_{dj}) \nu_j \tag{2.36}$$

and for non-output neurons is

$$\frac{d\sigma_j}{dt} = -\eta \left(\sum_k \frac{\partial J}{\partial o_k} \Delta_{kj} \right) \nu_j \tag{2.37}$$

2.3.3 Adjustment of Membrane Time Constants

To derive how to adjust the membrane time constants τ_j (given in Eq. (2.3)), let $\beta_j = \frac{\partial o_j}{\partial \tau_j}$.

As the derivation of this formula involves a number of 'clever tricks', the entire derivation is shown for clarity. The derivative of Eq. (2.28) with respect to τ is

$$\frac{\partial}{\partial \tau_j} (\tau_j o_j + \dot{o}_j = \sum_k w_{jk} f_j(o_k) + b_j) \tag{2.38}$$

Substituting yields

$$\tau_j \beta_j + \dot{\beta}_j = \frac{\partial}{\partial \tau_j} (\sum_k w_{jk} f_j(o_k) + b_j) \tag{2.39}$$

As b_j is a constant it can be dropped; Eq. (2.38) then simplifies to

$$\tau_j \beta_j + \dot{\beta}_j = \frac{\partial}{\partial \tau_j} \left(\sum_k w_{jk} f_j(o_k) \right) \tag{2.40}$$

Using the chain rule yields

$$\tau_j \beta_j + \dot{\beta}_j = \frac{\partial \sigma_j}{\partial \tau_j} \cdot \frac{\partial}{\partial \sigma_j} \left(\sum_k w_{jk} f_j(o_k) \right) \tag{2.41}$$

or

$$\tau_j \beta_j + \dot{\beta}_j = \frac{\partial \sigma_j}{\partial \tau_j} \cdot \left(\sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \right) \tag{2.42}$$

Now $\frac{\partial \sigma_j}{\partial \tau_j}$ needs to be found. Applying the chain rule again yields

$$\tau_j \beta_j + \dot{\beta}_j = \frac{\partial o_j}{\partial \tau_j} \cdot \frac{\partial \sigma_j}{\partial o_j} \cdot \left(\sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \right) \tag{2.43}$$

Simplifying

$$\tau_j \beta_j + \dot{\beta}_j = \beta_j \cdot \frac{\partial \sigma_j}{\partial o_j} \cdot \left(\sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \right) \tag{2.44}$$

Now $\frac{\partial \sigma_j}{\partial o_j}$ needs to be found. For o_j greater than or equal to zero

$$f_j(o_j) = \frac{2}{1 + e^{-\sigma_j o_j}} - 1 \tag{2.45}$$

Rearranging yields

$$1 + e^{-\sigma_j o_j} = \frac{2}{f_j(o_j) + 1} \tag{2.46}$$

or

$$e^{-\sigma_j o_j} = \frac{2}{f_j(o_j) + 1} - 1 \tag{2.47}$$

Taking the natural log and rearranging yields the solution for σ_j. We can express this as

$$\sigma_j = \frac{-\ln\left(\frac{2}{f_j(o_j)+1} - 1\right)}{o_j} \tag{2.48}$$

So taking the derivative of Eq. (2.48) yields the solution for $\frac{\partial \sigma_j}{\partial o_j}$

$$\therefore \frac{\partial \sigma_j}{\partial o_j} = \frac{\ln\left(\frac{2}{f_j(o_j)+1} - 1\right)}{o_j^2} = \frac{-\sigma_j}{o_j} \tag{2.49}$$

For o_j less than zero, $\frac{\partial \sigma_j}{\partial o_j} = 0$.

Substituting Eq. (2.49) into Eq. (2.44) yields

$$\dot{\beta}_j = -\beta_j \cdot \left(\frac{\sigma_j}{o_j} \cdot \left(\sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \right) + \tau_j \right) \tag{2.50}$$

for $o_j \geq 0$ and

$$\dot{\beta}_j = -\beta_j \cdot \tau_j \tag{2.51}$$

for $o_j < 0$. But this differential equation for β_j has a closed form solution, given as

$$\beta_j = e^{-kt} \tag{2.52}$$

where

$$k = \begin{cases} \left(\frac{\sigma_j}{o_j} \cdot \left(\sum_k w_{jk} \frac{\partial f_j(o_k)}{\partial \sigma_j} \right) + \tau_j \right) & \text{for } o_j \geq 0 \\ \tau_j & \text{for } o_j < 0 \end{cases} \tag{2.53}$$

The membrane time constant adjustment formula for the output neuron is

$$\frac{d\tau_j}{dt} = -\eta\,(o_j - o_{dj})\,\beta_j \tag{2.54}$$

and for non-output neurons is

$$\frac{d\tau_j}{dt} = -\eta\left(\sum_k \frac{\partial J}{\partial o_k}\Delta_{kj}\right)\beta_j \tag{2.55}$$

Note that τ_j can not be adjusted too fast as τ_j is treated as a constant in Eq. (2.38).

2.4 Modelling Nonlinear Systems Using Recurrent Neural Networks

Nonlinear dynamic systems are typically difficult to model. Methods of nonlinear identification such as Volterra series or Weiner series are limited in their usefulness [15]. Similarly using a linear system to model the nonlinear system about certain operating points is also limited. In this respect, recurrent neural networks seem to be a viable alternative for modelling nonlinear systems. We want to prove that recurrent neural networks can be used to model arbitrary dynamic nonlinear systems. Ideally we would like to prove that this modelling can be done to arbitrary precision. Unfortunately such proofs currently do not exist for the asymptotically stable recurrent neural networks. However such proofs do exist for feedforward neural networks. Hornik *et al.* proved that multilayer feedforward neural networks consisting of three layers with sigmoid units on the hidden layer are universal approximators [13]. Other papers by Carrol and Dickensin; Cybenko; and Funahashi derived a similar proof only using different methods [5]. All these papers are concerned with approximating to a continuous functions defined on a compact set in $\Re^N$.

Narendra and Parthasarathy argue that feedforward multilayer neural networks can be 'considered as versatile nonlinear maps' and that there is a very close relationship between feedforward multilayer neural networks and recurrent neural networks [21]. Following this line of reasoning, Narendra and Parthasarathy used simulations to illustrate that recurrent neural networks can be used to identify and control nonlinear dynamical systems.

In a recent paper, [5] Chen and Chen explore the question 'can we give a neural network model which could be used to approximate the output of some dynamic system as a whole?' However Chen and Chen also state 'the problem of a neural network's capability in approximating nonlinear operators with its related application in computing the output as a whole of a dynamic system however remains open.' Chen and Chen derive a theorem that indicates a way of constructing a specific type of recurrent neural network models for identifying dynamic systems.

In another recent paper [17], Kosmatopoulos *et al.* proved that recurrent high-order neural networks (RHONN) are capable of approximating arbitrary dynamic systems. Given that we can represent recurrent neural networks using

$$\dot{x}_i = -a_i x_i + b_i \sum_j w_{ij} y_j \tag{2.56}$$

then a RHONN is represented by

$$\dot{x}_i = -a_i x_i + b_i \left[\sum_{k=1}^{L} w_{ik} \prod_{j \in I_k} y_j^{d_j(k)} \right] \tag{2.57}$$

where $d_j(k)$ are non negative integers and y_j is the j^{th} input to the above neuron. Kosmatopoulos *et al.* proves that given a sufficiently large number of high-order connections, a neural network consisting of neurons defined in Eq. (2.56), can approximate any nonlinear dynamical system to any degree of accuracy.

Although the above papers do not specifically prove that the asymptotically stable recurrent neural network can approximate an arbitrary dynamic system to any degree of accuracy, they do give an indication that such a proof may exist. However, in this thesis we explore the modelling capabilities of the asymptotically stable recurrent neural network through simulation.

2.5 Chapter Summary

In this chapter we studied recurrent neural networks and some of their properties. We proved that if certain conditions in the connectivity matrix are met then the recurrent neural network will exhibit asymptotically stable behavior. We derived how such a recurrent neural network can be trained by using a variation of back propagation. Finally we analyzed how a recurrent neural network can be used to model nonlinear systems.

Chapter 3

Simulating a Recurrent Neural Network

A recurrent neural network simulator based on the asymptotically stable recurrent neural network was written by C. Jubien as part of his master's thesis [15]. Although Jubien's simulator appeared to give valid results, Jubien did not explore the possibility of using heuristics to improve training. Also after some examination, it was clear that Jubien's simulator contained coding errors. The rewriting of the simulator took over six months, as many different ideas and heuristics were tried in an attempt to improve the training of the recurrent neural network. This chapter briefly describes the simulator. (A detailed description is given in Appendix A). This chapter also reviews some of the heuristics and improvements in training for the simulator and the data sets used to test the simulator. In this Chapter the term neural network refers to the asymptotically stable recurrent neural network unless otherwise stated.

3.1 Recurrent Neural Network Simulator

The recurrent neural network simulator is written in C and is currently executed on Sun and HP, UNIX based workstations. The simulator is based on the theory presented in Ch. 2. Essentially the simulator trains an asymptotically stable recurrent neural network to model (or track) a dynamic system provided by the user. After training the asymptotically stable recurrent neural network, the simulator provides the user with a set of weights so that the neural network can be simulated repeatedly without having to retrain.

The recurrent neural network that the simulator uses is fairly restrictive, as the simulator makes some major assumptions about the neural network configuration. These assumptions are made primarily to simplify the simulator. The simulator assumes the neural network has a minimum of four layers structured as follows: layer 0 is the input layer; layer 1 is a scheduler layer; and the final layer is the output layer. The state layer (or layers) resides in between the scheduler layer and the output layer. The simulator can only track single input, single output (SISO) systems, or dual input, single output systems. The

number of inputs is dictated by the number of neurons in the input layer (i.e. one neuron indicates one input and two neurons indicates two inputs). There can be no input connections from other layers to the input layer. The input data to the simulator must be normalized to between zero and one. The simulator always assumes there is a scheduler layer. The scheduler layer is a 'special layer' used to turn the neurons in the state layer on or off (as detailed in Sec. 3.2.1). The weights, sigmoid shape constants, and membrane time constants in the scheduler layer are fixed. Any interconnections made from the scheduler layer to another layer must have negative interconnection weights to ensure stability. As there is only one output, the output layer consists of only a single neuron.

The simulator neural network is typically setup as indicated in the Figure 3.1.

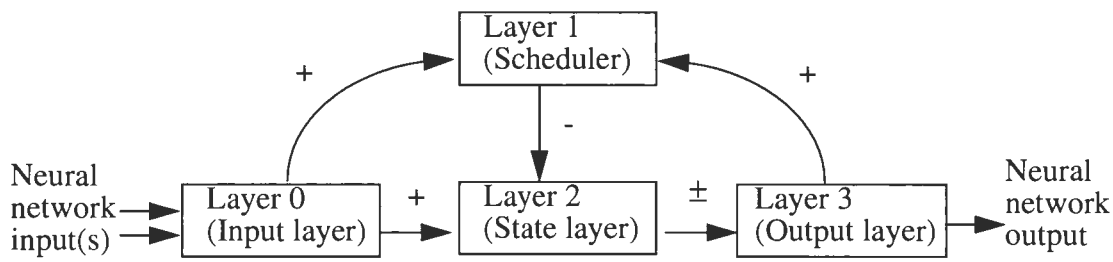

FIGURE 3.1 Typical neural network configuration.

(The plus and minus signs indicate the polarity of the weights; the ± sign indicates that the weights can be positive or negative).

3.1.1 User Input

The simulator gets almost all user input from four data files; simulator.cfg, neural_net.cfg, training_dir.cfg, and the training data file. Table 3.1 indicates the parameters specified in the file simulator.cfg.

TABLE 3.1 Configuration parameters for neural network simulator

Parameter	Description
total_samples	Total number of data points that the neural network will train and / or simulate on
epochs	Number of epochs to train for

TABLE 3.1 Configuration parameters for neural network simulator

Parameter	Description
learning_rate	η in Eq. (2.19)
data_file_name	File that contains the data
sampling_interval	Used in solving the differential equations
num_layer	Number of layers in the neural network

Although these parameters are explained in the Appendix A, the learning rate and the sampling interval deserve some mention here. The learning rate regulates the speed of training (see Eq. (2.19) in Sec. 2.3). If the learning rate is too high, the neural network training may become unstable and the training error (the difference between the neural network output and the desired output) may oscillate as shown in Fig. 3.2.

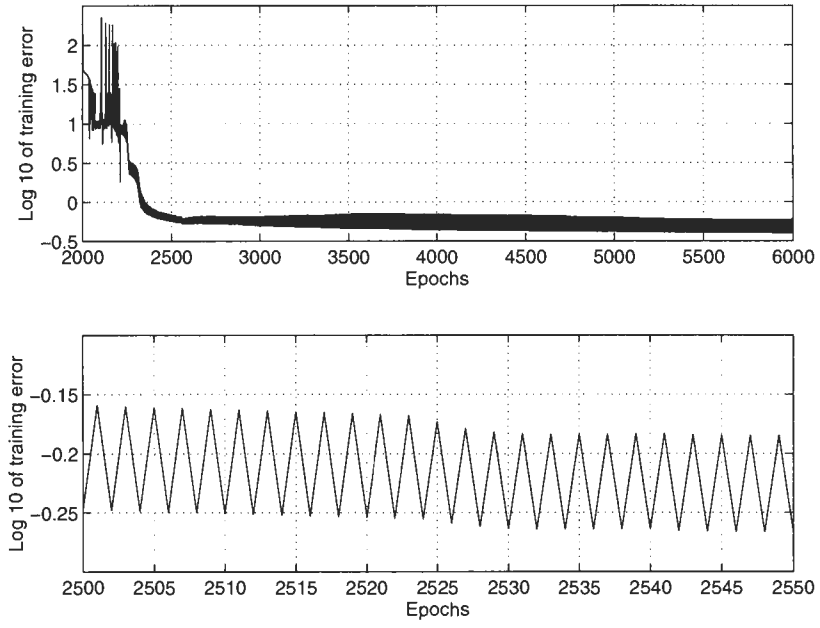

FIGURE 3.2 Training error oscillations.

However if the learning rate is too low, then the neural network will train very slowly. An empirical rule of thumb from [10] suggest values in the range 0.1 to 0.5 should be used for the learning rate. Zurada [26] suggested that the learning rate is dependent on the problem being solved with rates ranging from 10^{-3} to 10 having been reported as successful.

The sampling interval is used in solving the differential equations given in Eqs. (2.23), (2.27), (2.30), and (2.34). The simulator uses Euler's method to solve numerically

the differential equations. Although Euler's method is generally considered too inaccurate for practical use [4], based on experimental results it seems to work well in the simulator. The sampling interval is not based on the data, but rather found empirically. There is currently no known method to find the optimal sampling interval [4].

The configurable parts of the neural network are specified by the user through an input file called neural_net.cfg. The user can specify the layer interconnections (within the restrictions noted in Sec. 3.1), and the number of neurons in the scheduler layer and state layer(s). Training_dir.cfg specifies the directories containing the other setup files. Finally, the file containing the training data is specified by the user in the file simulator.cfg. Obviously it is important that the information contained in the four input files correspond to each other, otherwise the simulator will either not work or give meaningless results. The four setup files are explained in greater detail in the appendix.

One parameter that the simulator does get directly from the user is a random number seed. This number is used to seed a pseudo random number generator. As the interconnection weights are initially random, the seed will dictate what the initial weights will be. It is important to note that the neural network training is some what sensitive to the initial weights. Although the initial weights are only one of many factors in determining how well a neural network will train, most neural networks are sensitive to the initial weights [26].

3.1.2 Simulator Output

There are three types of files written out by the simulator: debugging files, internal use files, and training and simulation result files. The debugging files are useful for code development and debugging. The internal use files are used by the simulator to store the state of the neural network. For the user, the most important files are the results files.

For each epoch the simulator calculates

$$s = \sum_{i=0}^{n} (o_i - o_{di})^2 \tag{3.1}$$

where n is the number of samples, o_i is the neural network output for sample i, and o_{di} is the target output for sample i. This value is written to the file NN_error.dat for each epoch (i.e. 500 epochs mean 500 file entries). By looking at this file in MATLAB, the user can see how well the neural network trained. A typical example is shown in Fig. 3.3.

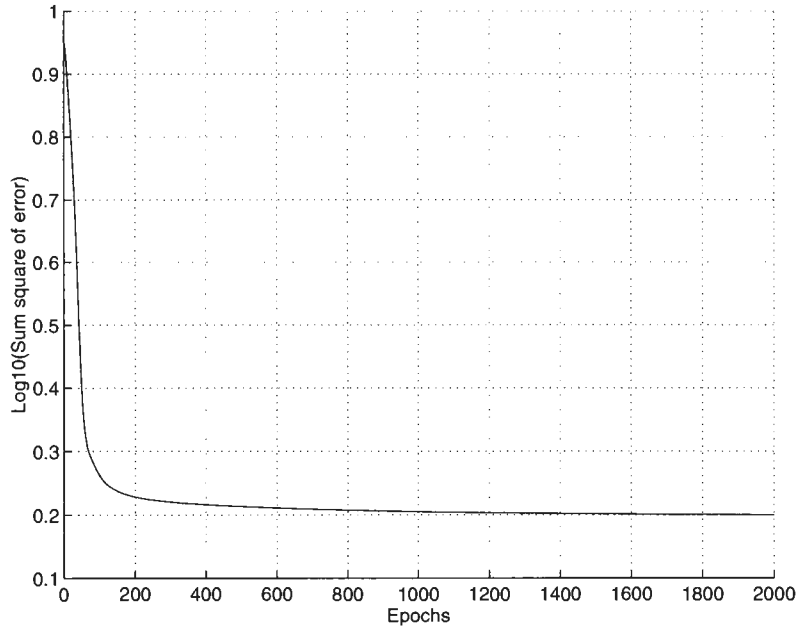

FIGURE 3.3 Sample training error.

Typically if the neural network trained well the error graph should appear as shown in Fig. 3.3.

After the neural network is trained, simulations can be run. In the simulation the input data (from the training data file) is applied to the input of the neural network, and the output of the neural network is calculated. For every sample, the input, target output, and actual output are written to file. Using a software package such as matlab, the user can see how well the neural network tracked the system.

3.2 Changes to Original Simulator

Besides the correction of simple coding bugs, there were four major changes made to the original simulator that caused a significant improvement in the results. The first change was changing the activation function for the neurons in the scheduler layer from a notch to

a peak. The second change was making the sampling interval to a user defined parameter. The third change was making the truncated sigmoid y axis range from 0 to 1 instead of 0 to 0.5. The fourth and final change was making the initial condition of the neurons non-zero. The next four sections explain these changes in greater detail.

3.2.1 Scheduler Layer

As mentioned in the introduction to this chapter, the neural network simulator always has a scheduler layer. The scheduler layer is used to turn neurons in the state layer on or off. As the scheduler layer always inhibits the state layer, the neurons in the scheduler layer must have a special response. For the scheduler layer, we want a neural network that can generate a peaked response. The architecture for such a neural network is shown in Fig. 3.4 with the activation function shown in Fig. 3.5.

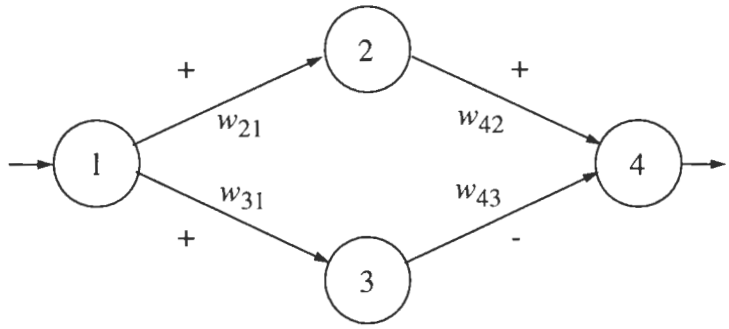

FIGURE 3.4 Architecture for scheduler neural network.

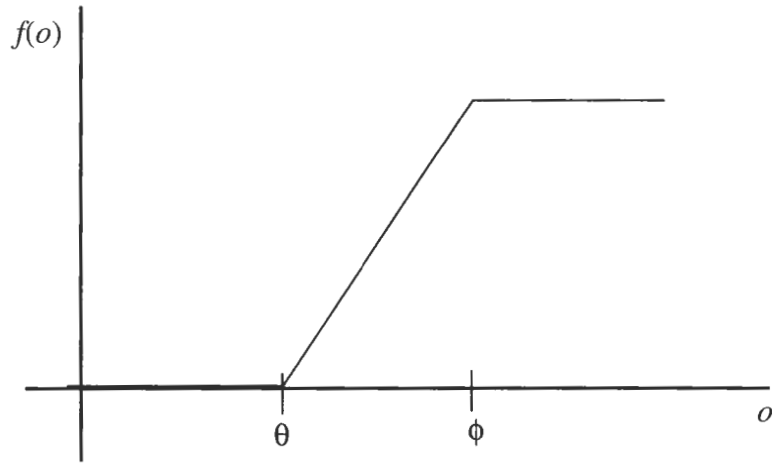

FIGURE 3.5 Activation function for neurons in the scheduler neural network.

In [15] Jubien proves that such a neural network can be trained to have a peaked response. Rather than training such a neural network, we can devise neurons that have the desired peaked response and thus the neurons in the scheduler layer are 'special purpose' neurons i.e. these neurons do not follow the state equation given in Eq. (2.23). Despite Jubien stating in his thesis the scheduler neurons have a peaked response, the original simulator used 'notch' response. The scheduler neuron state equation originally used by the simulator program written by Jubien is

$$o_j = 0.1 - 0.1e^{\left(-\left(\frac{\left(\sum_i w_{ji}o_i\right) - x}{y}\right)^2\right)} \tag{3.2}$$

where o_j is the activation of scheduler neuron j, x determines the midpoint of the notch, and y determines the width of the notch. This response is shown in Fig. 3.6.

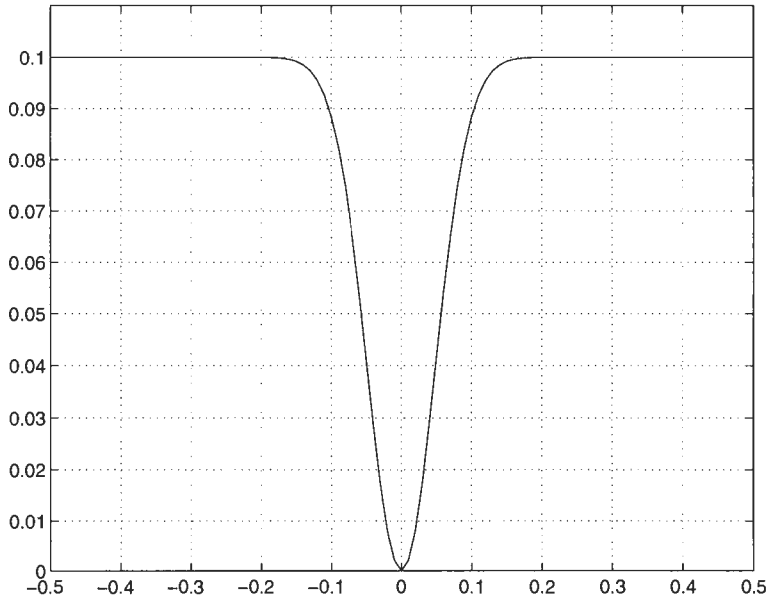

FIGURE 3.6 Notch response of scheduler neurons, x= 0, y = 0.0682.

In the new simulator though, experiments indicated that this 'notch' type response was responsible for causing poor results. Furthermore, Jubien's own thesis indicated that he used a peaked response and not a notch response. The formula for this type of response is

$$o_j = e^{\left(-\left(\frac{\left(\sum_i w_{ji} o_i\right) - x}{y}\right)^2\right)} \tag{3.3}$$

The peaked response is shown in Fig. 3.7. Note the difference in the scale of the y axis compared to Fig. 3.6.

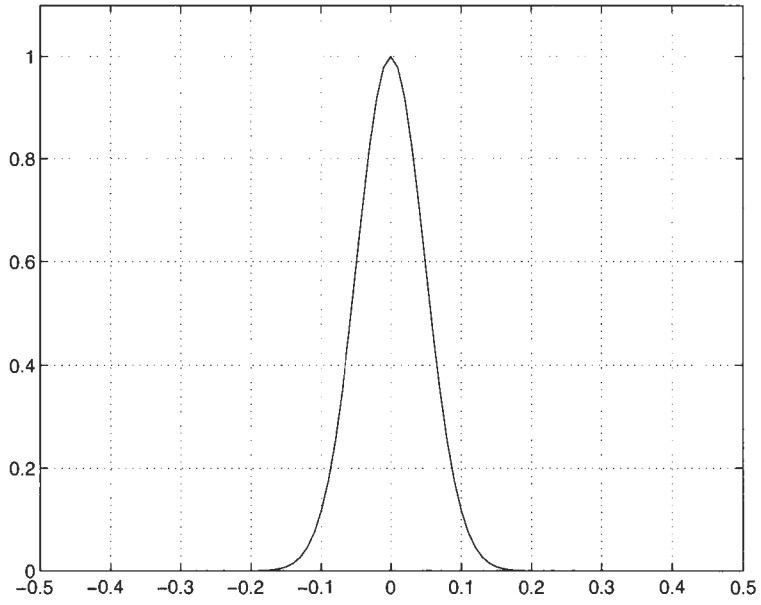

FIGURE 3.7 Peaked response of scheduler neurons, x= 0, y = 0.0682.

The reader may note that Eq. (3.3) is the formula for a gaussian pulse. At the 'top' of the gaussian pulse, the scheduler neuron will inhibit any destination neuron(s) in the state layer (i.e. the neuron(s) in the state layer will turn 'off'). The neurons in the scheduler layer are setup so that the pulses will overlap for smooth 'on/off' transitions in the state layer as illustrated in Fig 3.8.

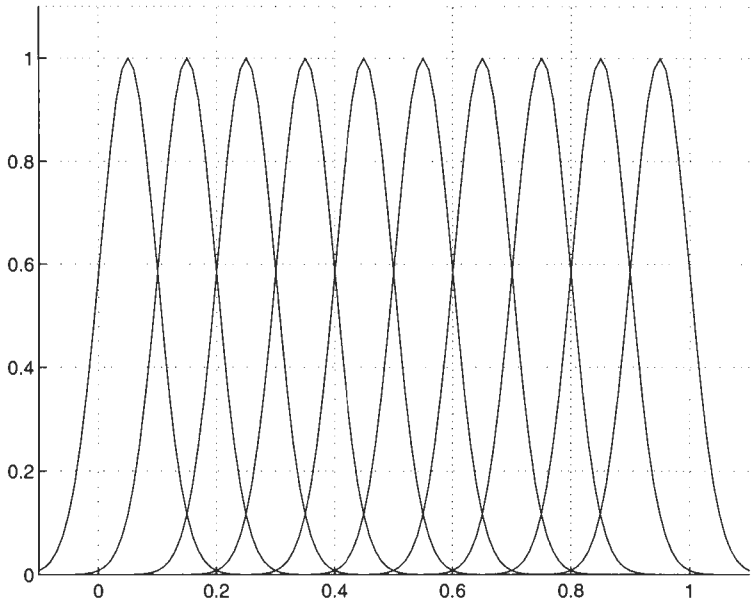

FIGURE 3.8 Total response of scheduler layer.

When the simulator used the gaussian pulse response, there was a significant improvement in the results.

3.2.2 Sampling Interval

As stated in Sec. 3.1.1, the sampling interval is used in solving the differential equations. In the original simulator the sampling interval was hard coded to 0.24. Experiments indicated that not only was this value too high, but that it was also dependent on the data set. As there is no known way to find the optimal sampling interval, this must be found empirically. Because of this, the sampling interval was changed to a user defined value.

3.2.3 Sigmoid Axis Change

In the original simulator, the formula for the truncated sigmoid function was

$$f(o) = \begin{cases} \dfrac{1}{1+e^{-\sigma o}} - \dfrac{1}{2} & o \geq 0 \\ 0 & o < 0 \end{cases} \tag{3.4}$$

Changing the truncated sigmoid to the formula given in Eq. (2.32) improved results.

3.2.4 Neural Network Initial Conditions

One simple improvement to the original simulator was to make the initial activation in the neural network nonzero if necessary. Fig. 3.9a illustrates how the neural network has to slowly rise to the correct level when the system it is tracking starts off at a nonzero value. However, if the neural network is initially set to the first value of the system, the neural network immediately tracks correctly as shown in Fig. 3.9b.

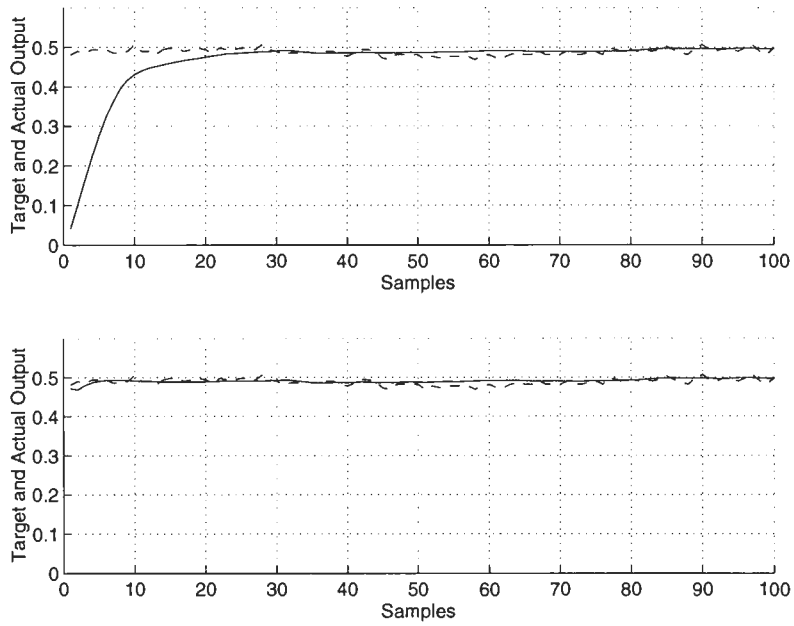

FIGURE 3.9 Effect of non zero initial conditions. Solid line neural network; dashed line system.

This simple improvement is analogous to precharging the capacitors in a filter.

3.3 Training Heuristics

Once the code had been rewritten, the effort was made to improve the neural network's tracking of nonlinear systems and to increase the rate at which the neural network trained. To achieve this, we need to look at what makes learning slow in the first place. If we visualize the error function of the neural network in two dimensions as shown in Fig. 3.10,

then we can see that using steepest descent for training the neural network, the solution can get trapped in a local minimum.

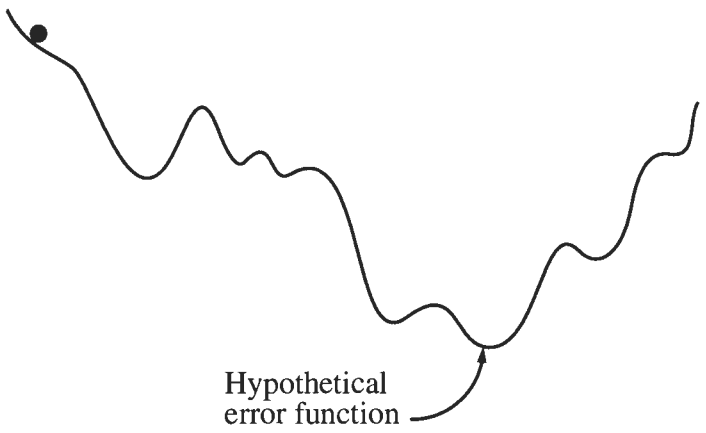

FIGURE 3.10 Hypothetical two dimensional error function with many local minima.

When the function being optimized using SDM reaches a local minimum, the gradient will be zero and thus the optimization will stop at this suboptimal value. We can improve the training by avoiding shallow local minima.

The heuristics discussed in this chapter are all various methods of avoiding shallow local minima. Various ideas were tried, included an initial path search, a momentum term, an adaptive learning rate, and a closed form learning rate. Note that for these training runs the simulator only trained the weights. Training the shape and membrane time constants using the theory given in Ch. 2, along with training the weights was also tried.

3.3.1 Initial Path Search

Initially the interconnection weights are set to random values between zero and one. These random values can greatly affect the training. A 'poor' set of initial weights can prevent the neural network from training. To avoid this problem a simple heuristic is applied to the simulator. Rather than taking the initial weights and doing a complete training run with them, ten initial weights are tested for 10% of the total number of training epochs each. The initial path that yields the lowest error is then followed for the remaining number of

training epochs. This heuristic allows the simulator to 'explore' the error space and take the path that would seem to yield the most promising results.

3.3.2 Momentum Term

The idea of the momentum term is to force the optimization over local minima. (Note that the momentum term is only applied to the weights). Basically the weights are adjusted using a fraction of the previous weight change plus a fraction of the weight change suggested by back propagation. The idea is that the momentum will force the optimization past local minima as illustrated in Fig. 3.11.

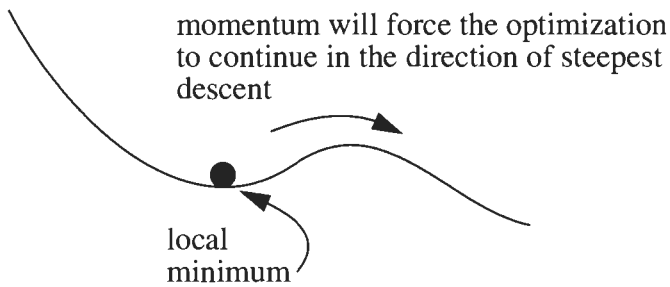

FIGURE 3.11 Momentum example.

The momentum term is applied as

$$\Delta W_{k+1}(i,j) = mc\Delta W_k(i,j) + (1-mc)\frac{dw_{ji}}{dt} \tag{3.5}$$

$$\Delta W_1(i,j) = \frac{dw_{ji}}{dt} \tag{3.6}$$

where $\Delta W_k()$ is the change in the weight matrix at time k, mc is the momentum constant (typically 0.95 [6]), and $\frac{dw_{ji}}{dt}$ is the weight change suggested by the back propagation (Eqs. (2.13) and (2.14)).

3.3.3 Dynamic Learning Rate

As stated in Sec. 3.1 there is currently no known way to find the optimal learning rate. Although the steepest descent method gives the direction in which to proceed to minimize the function, η determines how far in this direction the optimization should proceed. A simple two dimensional example of this is illustrated in Fig. 3.12.

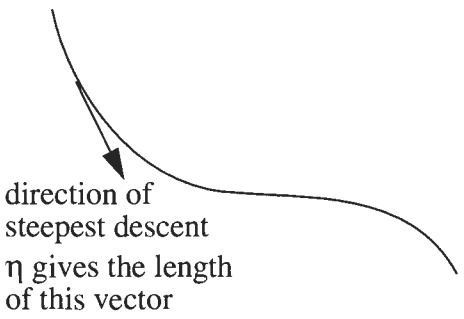

FIGURE 3.12 Steepest descent example.

One alternative to a fixed learning rate is to adjust it according to some gradient. Unfortunately there is no simple way to determine the optimal value of η dynamically for training neural networks; it is not possible to do a line search as there is no direct relationship between the neural network error (Eq. (2.17)) and η. Two possible solutions to this problem are suggested below.

3.3.3.1 Adaptive Learning Rate

While the simulator is training the neural network, the simulator calculates the error of the neural network as

$$e = o - o_d \tag{3.7}$$

where o is the output of the neural network, and o_d is the target output (specified by the training data). A simple procedure for finding $\frac{\partial e}{\partial \eta}$ that has been suggested by [19] is

$$\frac{\partial e}{\partial \eta} = \frac{e(\eta + \delta) - e(\eta)}{\delta} \tag{3.8}$$

where $e(\eta + \delta)$ is the error at $\eta + \delta$, $e(\eta)$ is the error at η, and δ is a small perturbation (i.e. 10^{-5}). Eq. (3.7) is an approximation of $\frac{\partial e}{\partial \eta}$ obtained through first principles. η can then be adjusted using

$$\eta_{k+1} = \eta_k - \alpha \frac{\partial e}{\partial \eta} \tag{3.9}$$

Essentially the steepest descent method is being applied to η. The problem again becomes how far along the direction given by $\frac{\partial e}{\partial \eta}$ to traverse. In the case of these experiments α was set to one; this value was chosen to simplify computations.

3.3.3.2 Closed Form Learning Rate

In [3] a closed form formula is given for calculating the length (η) of the direction vector for the SDM (see Fig. 3.12). Theoretically this value will result in the greatest reduction of the error, for a given direction. Based on this formula, a closed form expression for η is

$$\eta_k \approx \frac{g_k^T g_k \hat{\eta}^2}{2\left(\hat{o} - o_k + \hat{\eta} g_k^T g_k\right)} \tag{3.10}$$

$$\hat{o} = N(w_k - \hat{\eta} g_k) \tag{3.11}$$

where $\hat{\eta}$ is an estimate of η_k, o_k is the output of the neural network at time k, $\hat{o}$ is the output of the neural network as given in Eq. (2.23), N is an operator representing the valuation of the neural network for a given set of weights at time k, w_k is a vector of weights at time k, and g_k is a gradient vector of the cost function with respect to the weights (see Eq. (2.20)). A suitable value for $\hat{\eta}$ is η_{k-1}; a suitable value for the first iteration would be the user specified value for η. Note that the closed form value of η is calculated with respect to the weights.

3.3.4 Sigmoid Shape Constant and Membrane Time Constant

In the simulator, the sigmoid shape constants are set to one and the membrane time constants are set to a random number between zero and five for all layers other than the scheduler layer. It is possible that adjusting these values using the methods given in Ch. 2, may result in improved tracking of the system by the neural network.

3.4 Test Data

Initially three sets of data were used to test and simulate the simulator; data from a motorized boat, data from a PUMA 560 robot, and data from a cable television trunk amplifier. The next three sections give a brief explanation of each data set, and their configuration parameters.

3.4.1 Boat Data

This data was from a motorized boat moving through the ocean with a constant velocity. The boat is treated as a single input, single output (SISO) system. The boat's rudder angle is the system input, and the boat's heading is the system output. This data covers approximately twelve minutes. Fig. 3.13 illustrates the data set.

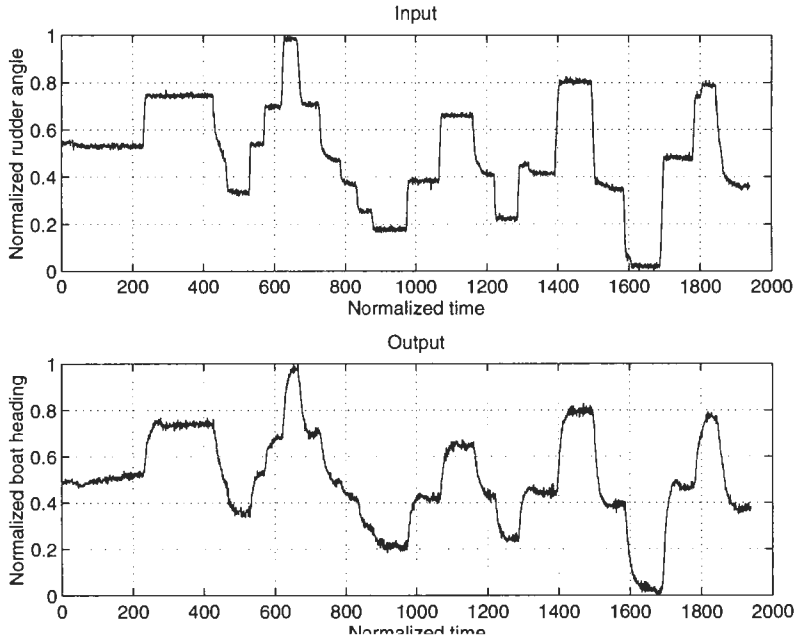

FIGURE 3.13 Boat data.

For all the boat data training runs, the configuration parameters were as shown in Table 3.2.

TABLE 3.2 Simulator configuration for boat data

Parameter	Value
samples	800
epochs	3000
learning_rate	0.8
num_layers	4
sampling_interval	0.09

The neural network was setup using the configuration shown in Fig. 3.14 and Table 3.3.

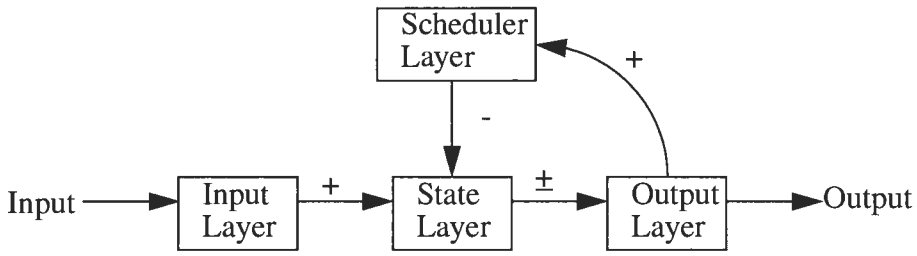

FIGURE 3.14 Neural network configuration for boat data.

TABLE 3.3 Neural network configuration for boat data

Layer	Number of neurons	Number of input neurons per destination neuron
Input Layer	1	0
Scheduler Layer	10	1 from Output layer
State Layer	5	1 from Input layer 6 from Scheduler layer
Output Layer	1	5 from State layer

3.4.2 Robot Data

This data is from a two link PUMA 560 robot. Although this is really a two input, two output system, only one output is identified. The two inputs are control voltages for the joint motors, and the two outputs are the joint angles. Fig. 3.15 illustrates the data set.

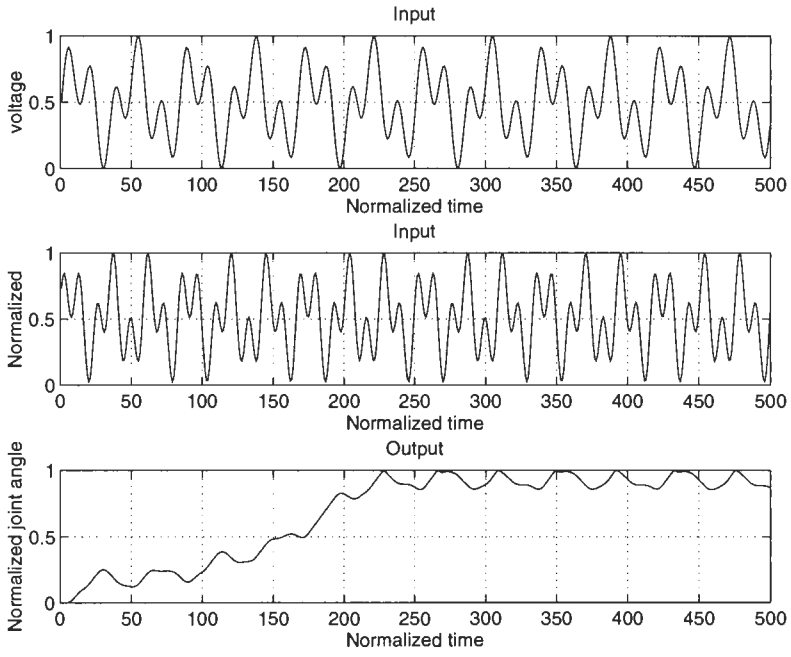

FIGURE 3.15 Robot data.

For all the robot data runs, the configuration parameters were as shown in Table 3.4.

TABLE 3.4 Simulator configuration for robot data

Parameter	Value
samples	500
epochs	10000
neta	0.1
num_layers	4
sampling_interval	0.04

The neural network was setup using the configuration shown in Fig. 3.16 and Table 3.5

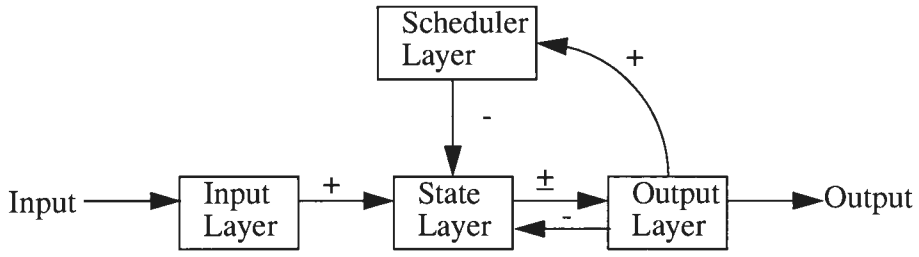

FIGURE 3.16 Neural network configuration for robot data.

TABLE 3.5 Neural network configuration for robot data

Layer	Number of neurons	Number of input neurons per destination neuron
Input Layer	1	0
Scheduler Layer	10	1 from Output layer
State Layer	5	1 from Input layer 6 from Scheduler layer 1 from Output layer
Output Layer	1	5 from State layer

3.4.3 Cable Television Trunk Amplifier Data

This data is from a cable television trunk amplifier. The amplifier is treated as a SISO system. The input to the system is the amplifier temperature, and the output from the system is the amplifier forward pilot signal. Although the data covers approximately a week of operation, the neural network was trained and simulated using half of day of data covering normal amplifier operation. Note that the amplifier data is quite noisy. The data is illustrated in Fig. 3.17.

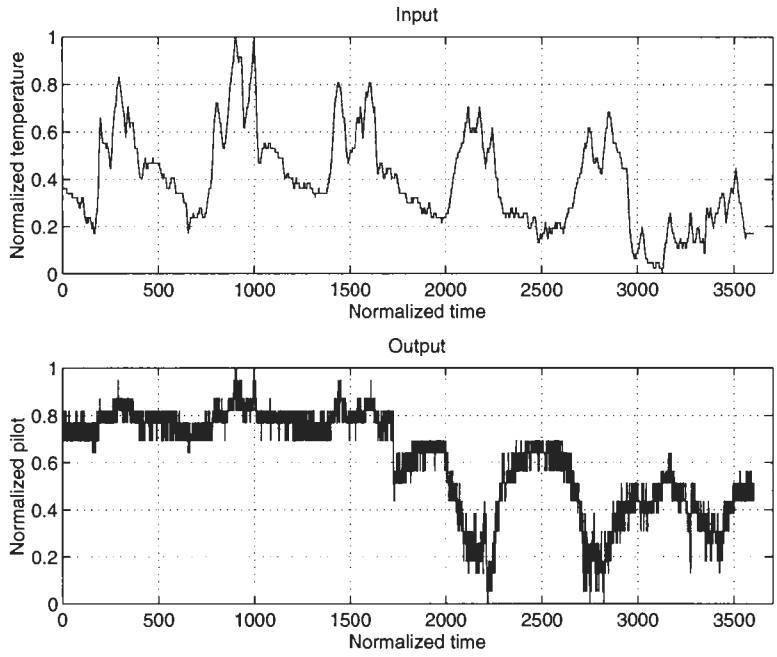

FIGURE 3.17 Amplifier data.

For all the amplifier data runs, the configuration parameters were as shown in Table 3.6.

TABLE 3.6 Simulator configuration for amplifier data

Parameter	Value
samples	1000
epochs	2000
learning_rate	0.3
num_layers	4
sampling_interval	0.09

The neural network was setup using the configuration shown in Fig. 3.18 and Table 3.7.

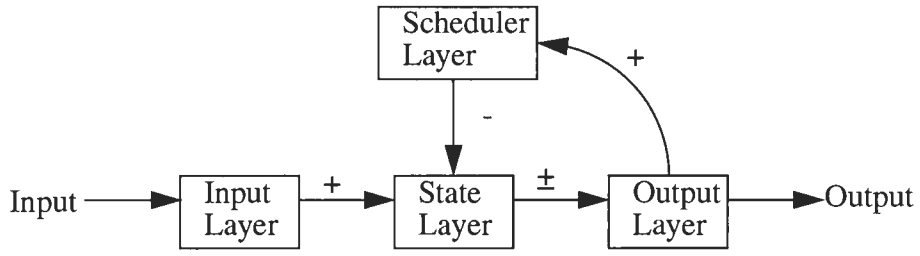

FIGURE 3.18 Neural network configuration for amplifier data.

TABLE 3.7 Neural network configuration for amplifier data

Layer	Number of neurons	Number of input neurons per destination neuron
Input Layer	1	0
Scheduler Layer	10	1 from Output layer
State Layer	5	1 from Input layer 6 from Scheduler layer
Output Layer	1	5 from State layer

This data is important as it can illustrate how that the neural network can be used for to detect behavior changes. The neural network is trained on data where the amplifier is behaving correctly and then simulated on all the data. A large difference between the neural network and the forward pilot indicate a behavior change. This is the basis for the work in Ch. 4.

3.5 Simulations

This section compares the training results obtained using the heuristics, to the benchmark training results to see if the heuristics made any improvement in the training. A set of training runs consisted of training each data set with ten different initial random weights. For each of the ten runs the training error per sample for the final epoch is calculated using

$$e = \sqrt{\frac{\sum_{i=0}^{n} (o_i - o_{di})^2}{n}} \tag{3.12}$$

The average, maximum, minimum, and variance of Eq. (3.12) is calculated for each set of training runs. (Note that the same ten seeds were used with each of the data sets so that comparisons could be made). The benchmark training results for the three data sets (i.e the training without heuristics) are given in Table 3.8.

TABLE 3.8 Benchmark simulations

Parameter	Boat	Robot	Amplifier
Min. error	0.01865	0.01443	0.03980
Max. error	0.03501	0.3139	0.04749
Ave. error	0.02637	0.08198	0.04136
Variance	2.3361e-3	0.01296	5.4906e-6

The next five sections give the results of the various training heuristics. The parameters that we are the most intersted in are the average error and the variance.

3.5.1 Initial Path Search

This section describes the results of implementing the initial path search. Table 3.9 illustrates the training results using ten initial paths for the three data sets.

TABLE 3.9 Initial path search simulations with ten initial paths

Parameter	Boat	Robot	Amplifier
Min. error	0.01865	0.01443	0.03957
Max. error	0.02604	0.08357	0.04015
Ave. error	0.02160	0.03461	0.03987
Variance	5.8265e-6	5.9381e-4	4.2104e-8

Table 3.10 illustrates the training results using three initial paths.

TABLE 3.10 Initial path search simulations with three initial paths

Parameter	Boat	Robot	Amplifier
Min. error	0.01728	0.01437	0.03961
Max. error	0.03102	0.5866	0.04361
Ave. error	0.02311	0.09361	0.04055
Variance	2.1903e-5	0.03049	1.8601e-6

With ten initial paths all three data sets showed improvement over the benchmark. Even with only three initial paths both the boat and amplifier data sets show considerable improvements over the benchmark. The robot data shows a slight improvement over the minimum error.

3.5.2 Momentum Term

This section describes the results of implementing the momentum term, as described in section 3.2.3. The momentum constant was set to 0.95. Table 3.11 illustrates the training results for the three data sets.

TABLE 3.11 Momentum simulations

Parameter	Boat	Robot	Amplifier
Min. error	0.02117	0.02645	0.03979
Max. error	0.07489	0.6054	0.07605
Ave. error	0.04133	0.1963	0.04417
Variance	3.7304e-4	0.03857	1.2617e-4

Interestingly, the momentum term heuristic made no improvement for either the boat or robot data and only a slight improvement in the minimum error for the amplifier data over the benchmark.

3.5.3 Dynamic Learning Rate

Both forms of the dynamic learning rate were implemented, and the results are described in the next two sections.

3.5.3.1 Adaptive Learning Rate

Table 3.12 illustrates the training results for the three data sets.

TABLE 3.12 Adaptive learning rate simulations

Parameter	Boat	Robot	Amplifier
Min. error	0.01992	0.02004	0.04076
Max. error	0.03586	3.2577	0.07453
Ave. error	0.02834	0.7796	0.04828
Variance	2.8687e-5	1.5123	1.0293e-4

The adaptive learning rate did not result in any improvement over the benchmark results.

3.5.3.2 Closed Form Learning Rate

In the initial experiments done with the closed form learning rate on the boat data and the robot data, the closed form solutions obtained for η were close to zero ($\sim 10^{-8}$). Because of this, the neural network did not train in any of the runs, and no statistics were collected.

3.5.4 Sigmoid Shape Constant Adjustment

Table 3.13 illustrates the training results for the three data sets.

TABLE 3.13 Sigmoid shape constant simulations

Parameter	Boat	Robot	Amplifier
Min. error	0.01834	0.01759	0.03967
Max. error	0.03520	0.6866	0.04472
Ave. error	0.02604	0.2186	0.04099
Variance	2.2074e-5	0.06787	3.5350e-6

Both the boat and amplifier data sets show improvements over the benchmark. The robot data shows no improvement over the benchmark.

3.5.5 Membrane Time Constant Adjustment

Table 3.14 illustrates the training results for the three data sets.

TABLE 3.14 Membrane time constant simulations

Parameter	Boat	Robot	Amplifier
Min. error	0.01882	0.04283	0.03980
Max. error	0.07949	0.4410	0.04247
Ave. error	0.03144	0.2703	0.04053
Variance	3.0592e-4	0.01428	8.9507e-7

The amplifier data sets show considerable improvements over the benchmark. The robot data and the boat show no improvement over benchmark.

3.5.6 Summary of Results Using Heuristics

Of all the heuristics used, only the initial path search improved upon the result of the benchmark for all data sets in at least one of the categories. The best training results for the all data sets in all categories were obtained using the initial path search. These results for the boat, robot, and amplifier are displayed in Figs. 3.19, 3.20, and 3.21 respectively.

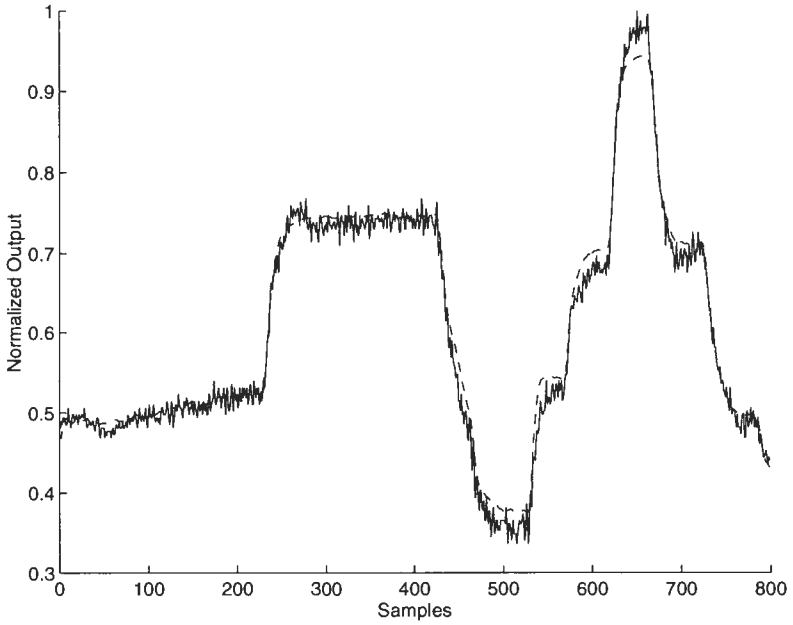

FIGURE 3.19 Best boat data results. Solid line boat; dashed line neural network.

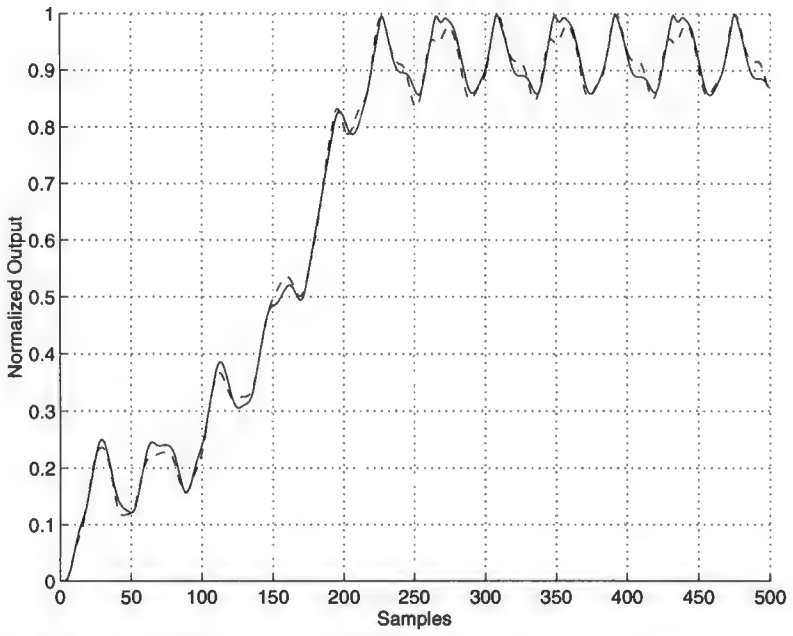

FIGURE 3.20 Best robot data results. Solid line robot; dashed line neural network.

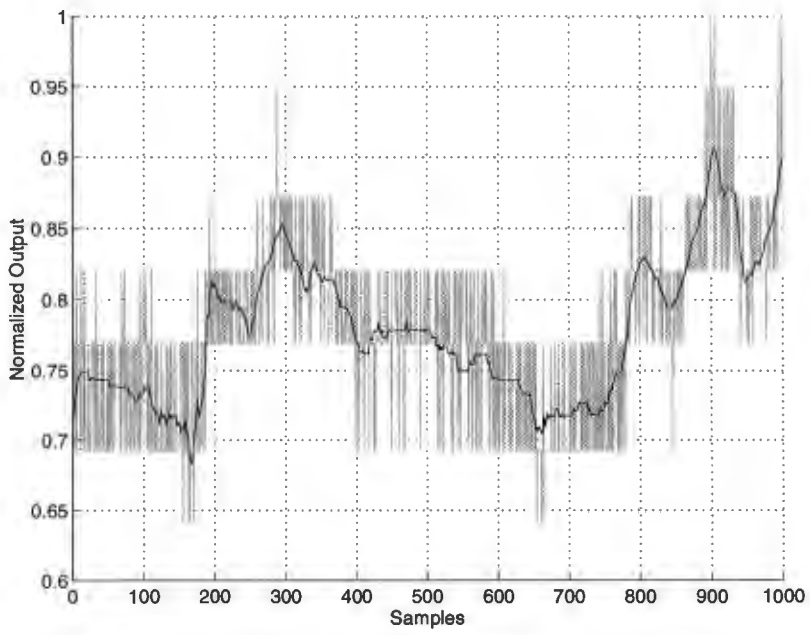

FIGURE 3.21 Best amplifier data results. Light gray trace forward pilot; dark line neural network.

Surprisingly the momentum term made either no difference or produced slightly worse training results in all the data sets. There are two possible reasons for this unexpected result. The error hypersurface could have very few local minima; the local minima that do exist are so deep that the momentum term is unable to work the optimization out. A simple two dimensional example of this is illustrated in Fig. 3.19.

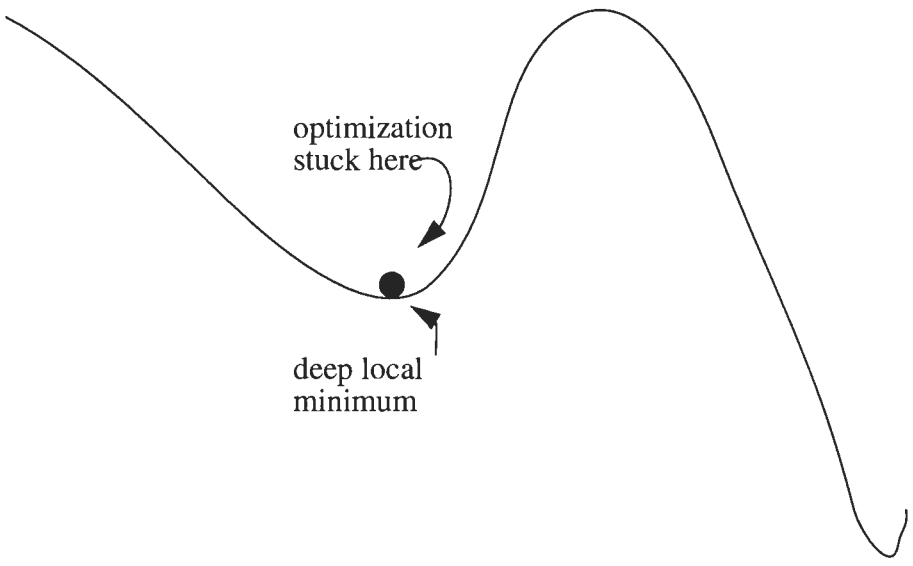

FIGURE 3.22 Deep local minimum example.

Another possibility is the error hypersurface may be very nearly flat, and thus the momentum will be very small. A simple two dimensional example of this is illustrated in Fig. 3.20.

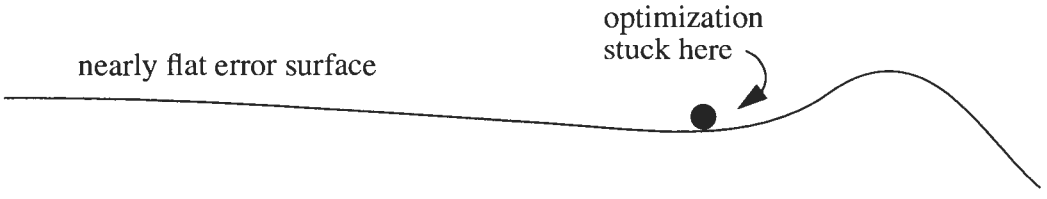

FIGURE 3.23 Flat error surface example.

The closed form learning rate failed, as the small values it calculated for η prevented the neural network from training. Two possible causes for this could be a nearly flat error hypersurface as shown in Fig. 3.20, or the optimization was immediately trapped in a

local minimum. Two possible reasons for the failure of the adaptive learning rate are the Eq. (3.8) may not be a good approximation of the derivative. Also the value chosen for α in Eq. (3.9) may not be appropriate.

As for the sigmoid shape constant and membrane time constant, it would seem that by adjusting the weights the fixed and random values of these parameters can be compensated for.

As stated previously, for the initial path search ten initial random weights were followed for 10% of the total number of training epochs and then the path with the lowest error was followed for the remaining training epochs as illustrated in Fig. 3.24.

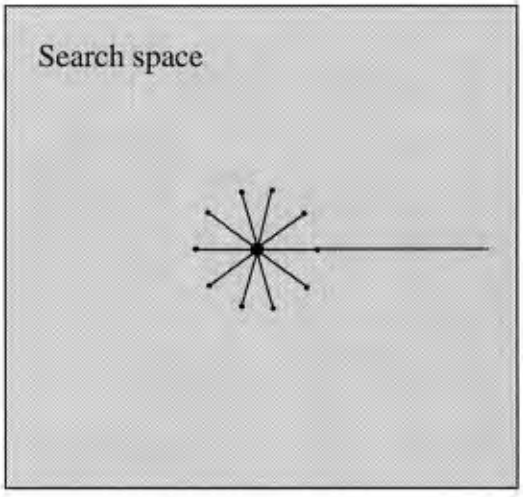

FIGURE 3.24 Conceptualization of initial path search.

We are assuming that the path with the lowest error will continue to remain that way for the remaining training epochs. To test this assumption, for the amplifier data each initial path was followed for all the training epochs as shown in Fig. 3.25 and the results were compared to that of the initial path search.

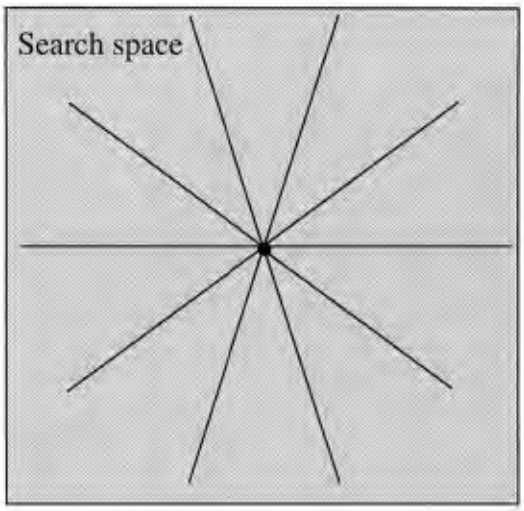

FIGURE 3.25 Conceptualization of searching all paths.

As there were ten runs with ten initial paths each, this results in ten groups of ten runs of 2000 training epochs each. For each group we can calculate the average final training error, the minimum final training error, and the maximum final training error. Table 3.15 compares these results to the initial path error. The column initial path error is the final training error calculated in Sec. 3.5.1 using the initial path search described in Sec. 3.3.1.

TABLE 3.15 Comparison of ten initial paths

Group No.	Ave. error	Min. error	Max. error	Initial path error
1	0.04142	0.03980	0.04749	0.03980
2	0.04119	0.03962	0.04361	0.03962
3	0.04024	0.03861	0.04262	0.03976
4	0.04099	0.03966	0.04341	0.04011
5	0.04372	0.03970	0.06098	0.03972
6	0.04272	0.03979	0.05813	0.03992
7	0.1988	0.03966	0.6165	0.04007
8	0.04588	0.03957	0.07457	0.03957
9	0.04041	0.03989	0.04131	0.04015
10	0.04065	0.03974	0.04462	0.03997

Table 3.15 indicates that with ten initial paths, choosing lowest error at 10% of the training epochs leads to the minimum error three times out of ten. Even if the error is not the minimum, the error is always below average. Using three initial paths, the results are compared in Table 3.16.

TABLE 3.16 Comparison of three initial paths

Group No.	Ave. error	Min. error	Max. error	Initial path error
1	0.04031	0.03986	0.04093	0.03986
2	0.04087	0.03980	0.04236	0.03980
3	0.04330	0.03988	0.04749	0.04217
4	0.04135	0.03977	0.04361	0.04361
5	0.04142	0.04109	0.04191	0.04125
6	0.04131	0.03962	0.04231	0.03962
7	0.03996	0.03970	0.04034	0.03970
8	0.03999	0.03961	0.04039	0.03961
9	0.04085	0.03976	0.04262	0.03976
10	0.04005	0.03985	0.04016	0.04016

With only three initial paths, the minimum results six times out of ten.

3.6 Summary

This chapter gave an introduction to the simulator for the asymptotically stable recurrent neural network. The most important concepts related to the simulator were described. A number of different training heuristics were described as well as the three data sets used to test the simulator. The results of applying the training heretics were presented and compared. The initial path heuristic showed the most improvement in the training over the benchmark.

Chapter 4

Recurrent Neural Network Application: Background

As the experiments in Ch. 3 indicated that the new neural network simulator is stable and reliable it became feasible to use it in an application. One of the applications for the neural network simulator is to model and detect behavior changes in the trunk amplifiers in a large scale cable television distribution network. To determine if it is possible to use the neural network to monitor the trunk amplifiers many experiments had to be performed. This chapter will give some background on the cable television trunk amplifiers, why monitoring the trunk amplifiers is important in this network, and how the recurrent neural network simulator can be applied to monitor the trunk amplifiers. Although most of the trunk amplifiers used in this experiment are from a Roger's cable plant in Newmarket Ontario, the results should be valid for any telecommunication network.

4.1 Trunk Amplifiers

This section will give some background on the trunk amplifiers used in the Newmarket cable television distribution network. Some information on the basic operation of the trunk amplifier and on modelling the trunk amplifier will be given as well as information on why monitoring is important in the trunk amplifiers. A more detailed description of the trunk amplifiers can be found in [22].

4.1.1 Background

Cable television distribution networks are used to distribute cable signals from a 'head-end' to individual subscribers' homes and businesses. The basic structure of the network consists of several hundred to a few thousand cable television trunk amplifiers interconnected by coax cable in a tree type network as shown in Fig. 4.1.

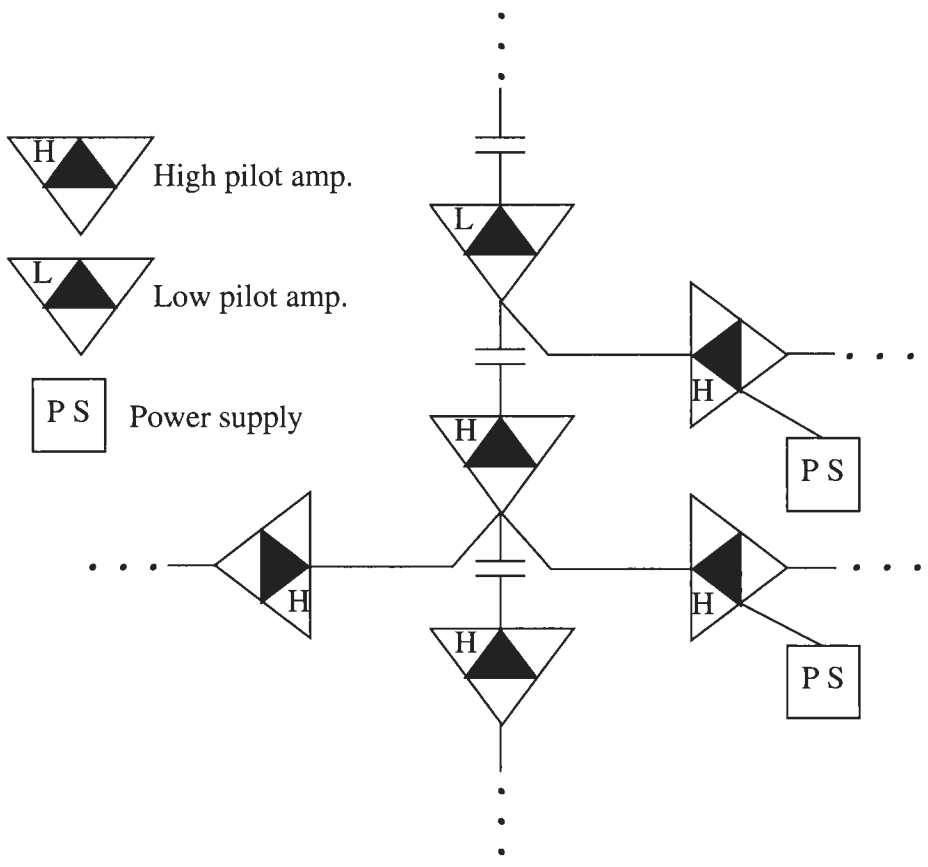

FIGURE 4.1 Section of amplifier network in Victoria, B.C. [22].

At the leaf nodes of this tree there are smaller spans of distribution amplifiers; taps from these amplifiers feed cable signals to the subscribers. As up to twelve distribution amplifiers can be supplied by a single trunk amplifier, the failure of a single trunk amplifier can affect hundreds of subscribers. To further compound the problem, when one trunk amplifier has a fault[1] this fault tends to propagate down the tree. Currently only the trunk amplifiers are monitored at the headend.

The cable television trunk amplifiers in Newmarket Ont. and Victoria B.C. are manufactured by C-COR Electronics Inc., and are designed specifically to amplify cable television signals. The trunk amplifiers are capable of amplifying signals over a range of

1. By fault we mean that some component of the amplifier has failed and the amplifier is in need of service.

50 to 550 MHz. The trunk amplifiers have two way communications capability; the forward path of the trunk amplifier distributes cable signals, while the reverse path is currently used for returning status information. The main purpose of these trunk amplifiers is to restore the strength of the cable signal, as the signal is attenuated by the spans of coax cable between the amplifiers. Two types of trunk amplifiers are required to maintain the desired signal strength; high pilot amplifiers and low pilot amplifiers. The high pilot amplifiers operate at 379.25 MHz or 499.25 MHz with the signal level at 38.0 dBmV. The low pilot amplifiers operate at 67.25 MHz with the signal level at 35.0 dBmV. The ratio of high pilot amplifiers to low pilot amplifiers is approximately five to one. Using these high and low pilot amplifiers, a flat signal level across the entire 500 MHz bandwidth is maintained. The losses incurred by the cable signal are dependent on the cable spans length and physical properties; typical losses are in the range of 10 to 15 dBmV. Ordinarily the trunk amplifier's feedforward amplifier module can compensate for these losses. Because of each span's unique attenuation properties, each trunk amplifier needs to be balanced by a technician before it is used.

If for some reason the feedforward amplifier module can not amplify the signal up to the desired level, a special circuit called the Automatic Level Control (ALC) will attempt to compensate. The ALC can only amplify a signal by up to 5 dBmV. In the event of a complete amplifier failure, the amplifier will go into a bypass mode. In bypass mode the signal passes through the amplifier's metal case. Although the signal does not get amplified, the ALC of the downstream amplifiers will compensate.

Key to monitoring the trunk amplifiers is the status information returned by the status monitoring transponder (SMT) in the trunk amplifier. The SMT module monitors and returns upon request the values of the following status parameters: temperature, regulated power supply voltage (B+ voltage), unregulated power supply voltage (raw DC voltage), forward pilot level, reverse carrier pilot level, and current. The trunk amplifiers are continuously polled in order of their SMT number by the status monitoring software (called INMS) at the headend. In the Newmarket plant, this results in a given trunk amplifier being polled once every 77 seconds [23]. This results in approximately 1100 samples per trunk amplifier every twenty four hours. The polling period is dependent on the num-

ber of trunk amplifiers in the network; the greater the number of amplifiers, the greater the time between polls.

For this application we are only interested in the forward pilot and the temperature. The temperature is a measure of the temperature in Celsius within the trunk amplifier's housing. The forward pilot level gives the value of the strength of the cable signal in dBmV as indicated in Fig. 4.2.

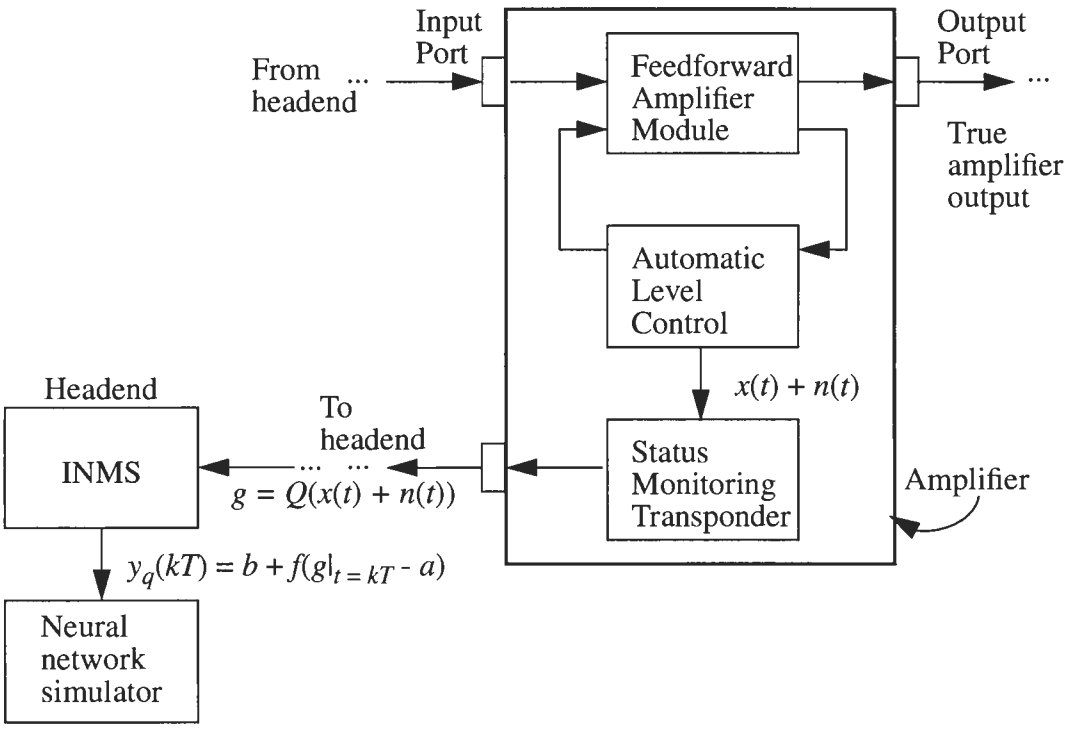

FIGURE 4.2 Forward pilot data used by the neural network simulator.

As Fig. 4.2 indicates, the forward pilot data reached by the neural network may be considerably different from the true output of the trunk amplifier, as the SMT module both quantizes and samples the data, and the INMS software does temperature correction on the data. We can model the forward pilot at INMS as

$$y_q(kT) = b + f(Q([x(t) + n(t)]|_{t = kT}) - a) \tag{4.1}$$

where a and b are constants, Q is a quantization function, f is a nonlinear function, $n(t)$ is noise, and $x(t)$ is the forward pilot as measured by the ALC. Although the quantization

steps are not uniform for either the forward pilot or the temperature, it is reasonable to assume that the quantization step for the forward pilot is approximately 0.5 dBmV and for the temperature is 0.5 °C [23]. Research indicates that both the forward pilot and temperature are not only under sampled but under quantized as well [23]. As a result of the under quantization of the forward pilot an enormous amount of noise is added to the signal. This further complicates the problem of monitoring the trunk amplifiers.

Other problems exist with the trunk amplifier's status information besides the under sampling and under quantization. As both the forward pilot and temperature are analog signals, these signals contain analog noise. This noise is compounded by the quantization noise. Another problem with the trunk amplifier's status information is occasional communication outages between the trunk amplifier and the headend. During these 'SMT outages' no status information is received from the trunk amplifier. Such outages must be compensated for in the monitoring software. Another problem in the status information are 'spikes' in the data; as these spikes are artifacts, they must be filtered out of the data. These problems indicate that the forward pilot and temperature must be preprocessed before being fed to the neural network.

4.1.2 Modelling the Trunk Amplifiers

As stated in Sec. 3.4, the recurrent neural network can model the trunk amplifier using the temperature as an input and the forward pilot as an output. This section will review why the trunk amplifiers are modelled in such a manner. Although the temperature is compensation is done in both the trunk amplifier circuitry and INMS, the forward pilot still varies with the temperature. Empirical evidence indicates that in a normally behaving trunk amplifier, the forward pilot is positively correlated with the temperature, i.e. if the temperature increases (decreases) the forward pilot level will increase (decrease) as indicated in Fig. 4.3. As can be seen in Fig 4.3, peaks in the forward pilot correspond to peaks in the temperature. In what are believed to be faulty trunk amplifiers[1], empirical evidence indi-

1. We are assuming that a significant change in the behavior of the forward pilot of the trunk amplifier is indicative of a fault.

cates that this behavior is usually reversed; i.e. the forward pilot is negatively correlated with temperature as indicated in Fig. 4.4.

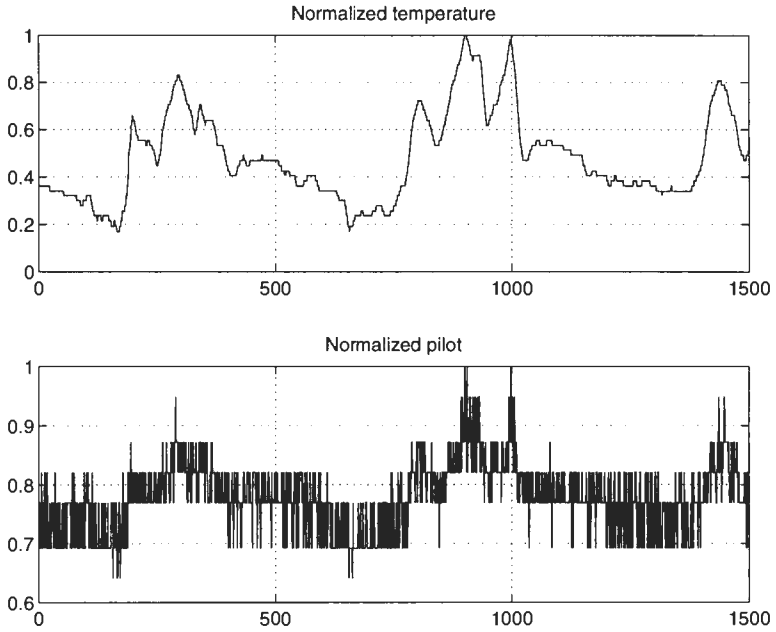

FIGURE 4.3 Example of correlated forward pilot variation with temperature.

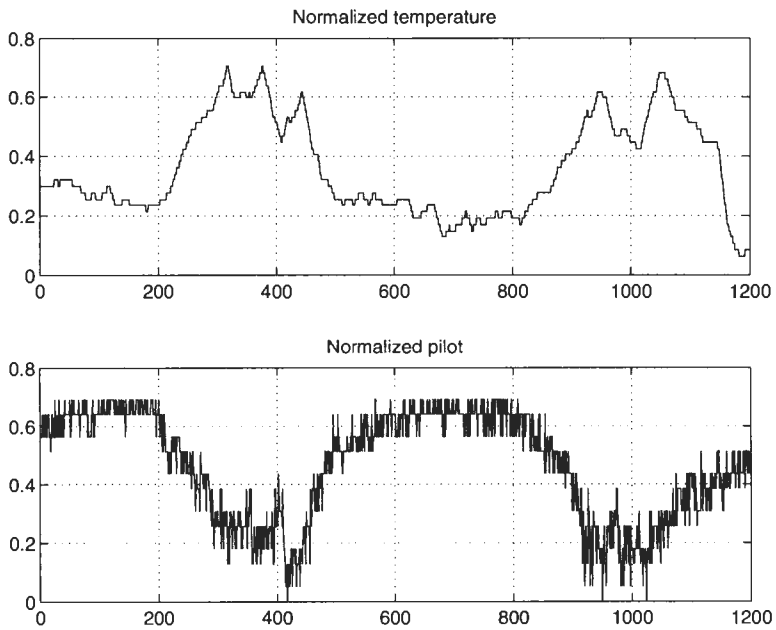

FIGURE 4.4 Example of anticorrelated forward pilot variation with temperature.

In this case, the peaks in the temperature correspond to drops in the forward pilot. This anticorrelation seems to be a feature present in many abnormally behaving amplifiers. In some other faulty trunk amplifiers there is a seemingly random mix of correlated and anti-correlated behavior as illustrated in Fig. 4.5.

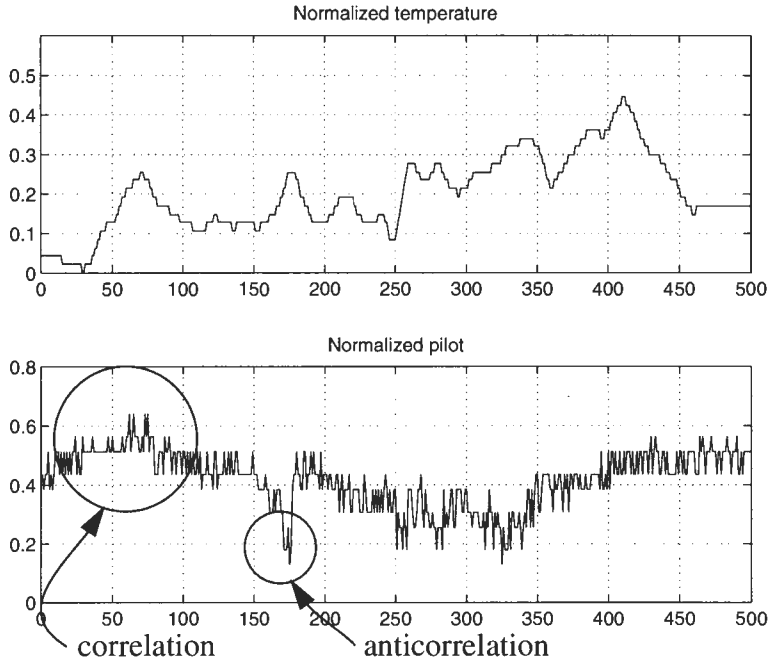

FIGURE 4.5 Example of mixed correlated and anticorrelated forward pilot variation with temperature.

Given this strong empirical evidence of the relationship between the temperature and the forward pilot, it was felt that it was best to model the amplifier using the temperature and forward pilot as the input / output relationship. The reader may feel that intuitively it would seem reasonable to use the input signal to the trunk amplifier as the input to the neural network and model the amplifier using this parameter. Unfortunately this information is not readily available; the SMT module does not return the value of the input signal to a trunk amplifier. Although the previous forward pilot from the upstream trunk amplifier may be available, there are three problems with this approach. The configuration of the amplifier network occasionally changes so it may not be possible to determine the upstream trunk amplifier. The second problem is that the data between the two trunk amplifiers may not be synchronized. Depending on SMT outages and the order in which

the trunk amplifiers are sampled, a noncausal system could result. Third, the ambient temperature also effects the coax cable joining the two trunk amplifiers and thus the output of the upstream amplifier may not be equal to the input at the downstream amplifier. Some experiments using the upstream forward pilot and temperature as input to the neural network indicated that the inclusion of the upstream forward pilot did not significantly improve the modelling of the trunk amplifier.

4.1.3 Motivation for Monitoring the Trunk Amplifiers

There two motivations behind this project. The first motivation involves customer service. Currently the INMS monitoring software can determine if any of the trunk amplifier fields are outside of their normal range by using fixed bounds. This method however tends to generate a great deal of false error messages due to the cable plant's natural drift and the trunk amplifier's variation with temperature. Because of the high number of false alarms, these error messages are usually ignored by Rogers' personnel. By ignoring all the error messages it is possible that some true errors may be missed and customer service may be lost. By having a more accurate method of monitoring the trunk amplifiers that can adapt to behavior changes in the amplifiers, customer service should be improved.

The second motivation involves the future use of the cable television trunk amplifier networks. Currently, Rogers Cable uses its network to deliver cable television signals. When delivering cable television signals, fault detection is not critical as an interruption in service is unlikely to have dire consequences. In the future though, Rogers Cable wants to be able to use their cable television amplifier network to deliver Internet access and phone service. In this case, fault detection is important as a service interruption could result in lost data or loss of phone service.

4.2 Using Neural Network Simulator to Monitor the Trunk Amplifiers

Typically, detecting behavior changes in a system is done using an analytical model to model the relationship between a system's input and output. Behavior changes are

detected if the output of the system does not match the output of the analytical model as shown in Fig. 4.6.

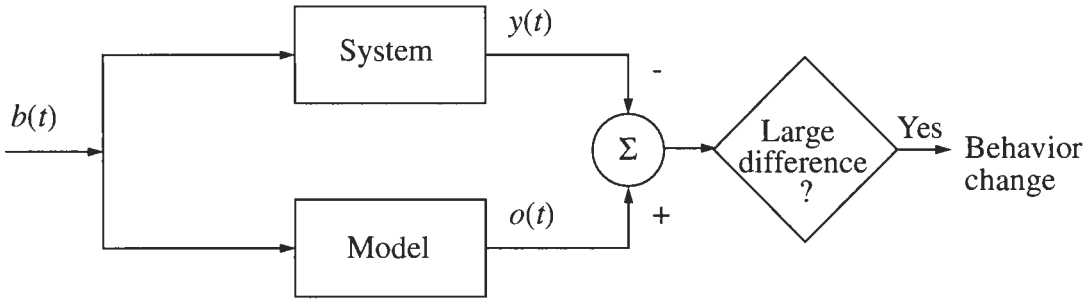

FIGURE 4.6 Conceptualization for detecting behavior changes.

In the case of the trunk amplifiers, no analytical model exists for the amplifiers when they are interconnected in a network. As little information is available on how the trunk amplifiers work, the amplifier is essentially a black box. Given these constraints, recurrent neural networks seem naturally suited to model the trunk amplifiers. As indicated in Sec. 2.4, a number of papers not only illustrate how recurrent neural networks can be used to model nonlinear systems, but that there is strong theoretical evidence that recurrent neural networks can learn a system arbitrarily well. There is a precedent for using neural networks for this type of application; [24] indicates how associative memory networks can be used for fault detection in nonlinear systems.

4.2.1 Behavior Change Detection in the Amplifier Network

The goal of the application is to be able to use the neural network simulator to monitor and detect changes in the behavior of the trunk amplifiers. From Ch. 3 we can see how this can be done. The neural network is trained on the data where the given trunk amplifier's behavior is consistent (i.e consistently correlated). The neural network is then simulated using the temperature data as an input; a large difference between the forward pilot level and the neural network output indicate the presence of a behavior change. A benchmark for the trunk amplifiers is given by amplifier number E170005 located in Victoria, B.C.. This benchmark consists of approximately one week of data. The first half of the data is correlated behavior, while the second half of the data is anticorrelated behavior. For this

benchmark, the neural network was trained on the first 1000 data points and then simulated on the entire data set; the results are shown in Fig. 4.7.

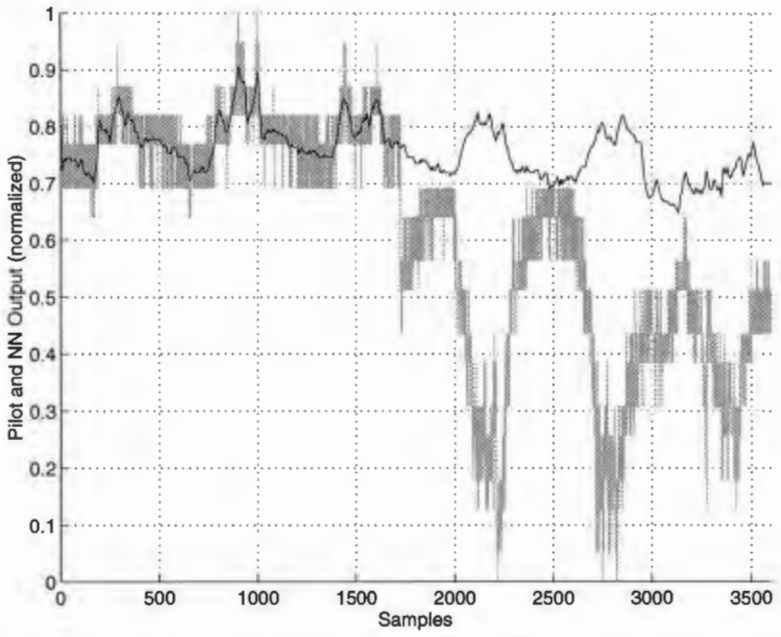

FIGURE 4.7 Behavior change detection example; light grey trace forward pilot; dark line neural network.

Fig. 4.7 clearly indicates that the neural network does track the trunk amplifier's correlated behavior, and does not track the trunk amplifier's anticorrelated behavior. Although the neural network detects the behavior change in this example, we need to determine if the results for E170005 are true in general. We also need to determine how big a difference constitutes a behavior change and when we should retrain the neural network. However before we can answer these questions, two other issues need to be analyzed first.

4.2.2 Data Preprocessing

Before the temperature and forward pilot data are given to the neural network for training or simulation the data is preprocessed. This preprocessing is currently done using a routine in matlab. The goal of this preprocessing is to remove any 'bad' data that might confuse the neural network. The routine processes the data in four ways. The routine first removes any time gaps. These gaps are caused by SMT outages. As the data just before and just after the time gaps is considered to be potentially bad, thirty data points before

and after the time gap are removed. The next three steps then analyze the forward pilot. The first step removes any ± 3200 dBmV readings from the forward pilot field. These events are caused by an 'out of range' error on the forward pilot field. Next any open lid events are removed. When the lid of the trunk amplifier is opened the amplifier's behavior becomes unstable. Finally any SMT not responding events are removed. Fig. 4.8 illustrates the before and after of the preprocessing procedure for the forward pilot from an amplifier from Newmarket Ont..

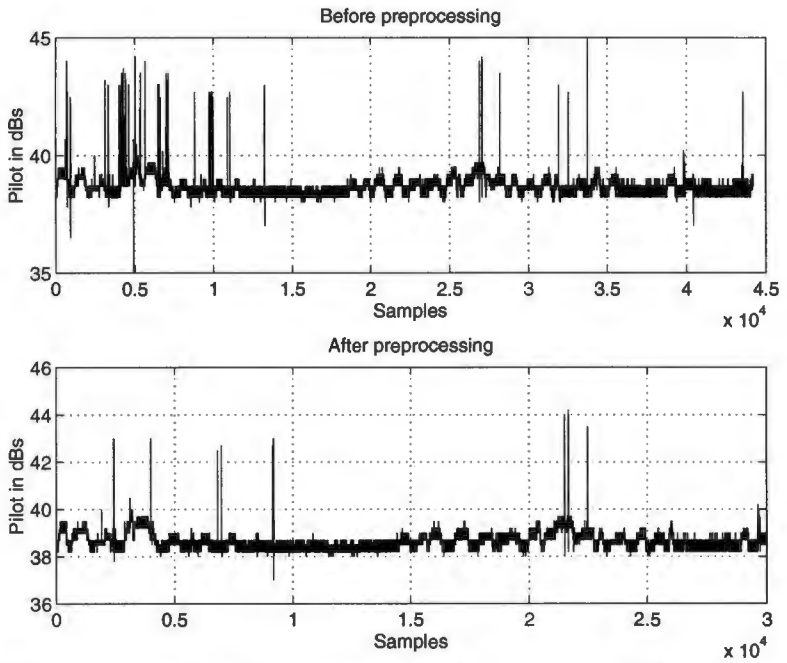

FIGURE 4.8 Before and after of preprocessing for amplifier 116 in Newmarket. Note the removal of the large spikes.

After the temperature and forward pilot data is preprocessed, the data is then normalized. Note that the filtering discussed above, can be done independently. That is, events such as an open lid, or SMT outage can be recognized independently. These, in a production system, would prevent the neural network from retraining unnecssarily and would be reported independently.

4.2.3 Normalization

As stated in Sec. 3.1, the input to the neural network must be normalized to between zero and one. For the trunk amplifiers, this step occurs after the data is preprocessed. A consis-

tent normalization method needed to be devised for the temperature, the high pilot amplifiers, and low pilot amplifiers. For the temperature, it was assumed that the temperature ranges between -20 °C and 100 °C. This range of values is mapped to between zero and one using

$$n = \frac{(t+20)}{120} \tag{4.2}$$

For the high pilot amplifiers, the forward pilot is assumed to be centered about 38 dBmV with a range of 0 to 76 dBmV. This range of values is mapped to between zero and one using

$$n = \begin{cases} 0 & (p < 0) \\ \frac{p}{360} & (0 \le p < 36) \\ \frac{p}{5} - 7.1 & (36 \le p < 40) \\ \frac{p}{360} + \frac{71}{90} & (40 \le p < 76) \\ 1 & (p \ge 76) \end{cases} \tag{4.3}$$

The reader should note that Eq. (4.3) gives a nonlinear mapping. As the high pilot amplifier should normally operate between 36 and 40 dBmV it was felt that this range should be normalized to between 0.1 and 0.9 to allow the neural network to track the trunk amplifier in this range with greater accuracy.

For the low pilot amplifiers, the forward pilot is assumed to be centered about 35 dBmV with a range of 0 to 70 dBmV. This range of values are mapped to between zero and one using

$$n = \begin{cases} 0 & (p < 0) \\ \frac{p}{335} & (0 \le p < 33.5) \\ \frac{4p}{15} - \frac{53}{6} & (33.5 \le p < 36.5) \\ \frac{p}{360} + \frac{71}{90} & (36.5 \le p < 70) \\ 1 & (p \ge 70) \end{cases} \tag{4.4}$$

Similar to the high pilot amplifiers, the mapping for the low pilot amplifiers is nonlinear.

4.3 Short Term Experiments

Having established a procedure for preprocessing and normalizing the trunk amplifier temperature and forward pilot data, the next step was to study how the neural network would model two months of amplifier data given various training regimes. The simulator configuration is given in Table 4.1.

TABLE 4.1 Simulator configuration for trunk amplifiers

Parameter	Value
samples	1000
epochs	2000
learning_rate	0.3
num_layers	4
sampling_interval	0.09

The configuration used for the neural network is given in Fig. 4.9 and Table 4.2.

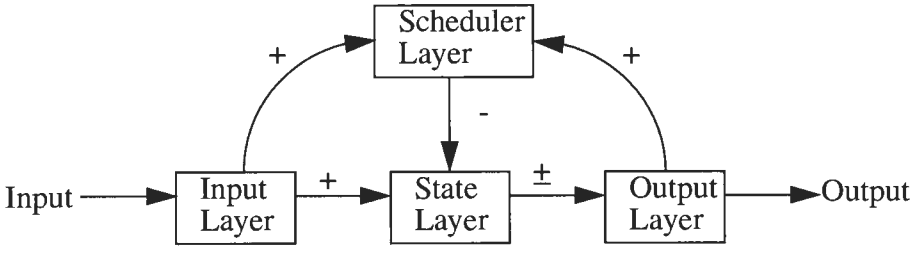

FIGURE 4.9 Neural network configuration for amplifier data.

TABLE 4.2 Neural network configuration for amplifier data

Layer	Number of neurons	Number of input neurons per destination neuron
Input Layer	1	0
Scheduler Layer	10	1 from Output layer 1 from Input layer
State Layer	5	1 from Input layer 6 from Scheduler layer
Output Layer	1	5 from State layer

Note that this configuration is slightly different than that given in Sec. 3.4.3. These experiments use initial path training heuristic with three initial paths.

4.3.1 Preliminary Experiments

In the first set of experiments the temperature and the forward pilot data from amplifier numbers 116 and 121 were used. Amplifier 116 is shown in Fig. 4.10 and is a suspected faulty amplifier as it exhibits both correlated and anticorrelated behavior.

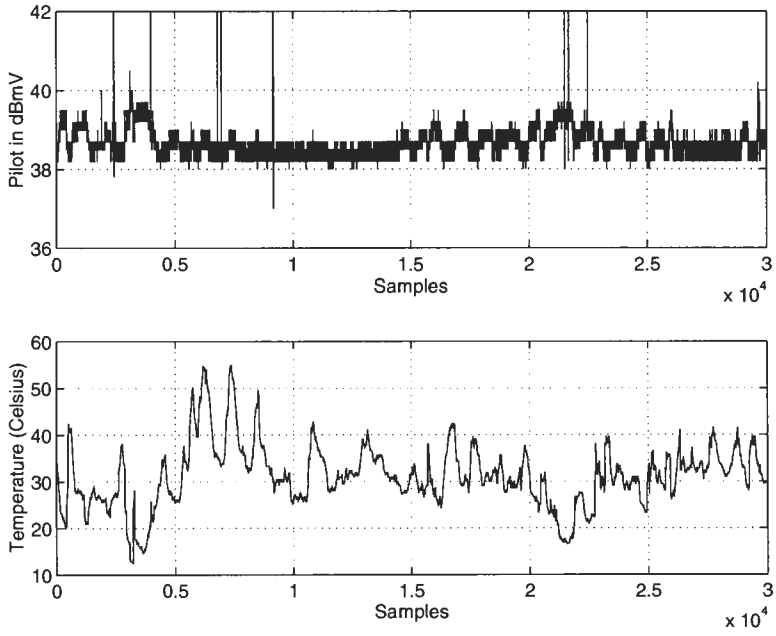

FIGURE 4.10 Temperature and forward pilot for amplifier 116. Note that the forward pilot is a mix of correlated and anticorrelated behavior.

Amplifier 121 is shown in Fig. 4.11 and is correlated for the duration of the two months.

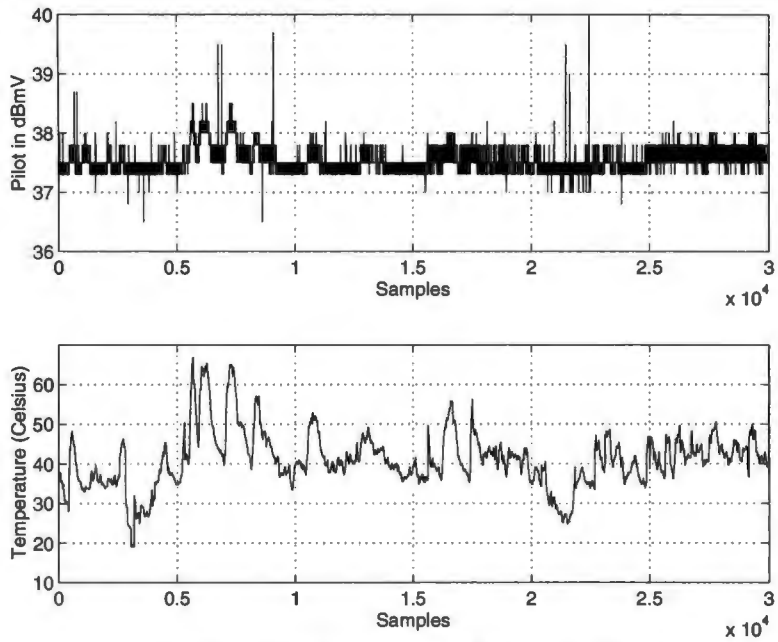

FIGURE 4.11 Temperature and forward pilot for amplifier 121. Note that the forward pilot is consistently correlated.

The data for these two high pilot amplifiers covers the months of March and April 1995; there are over 40000 sample points for each trunk amplifier. For each trunk amplifier two different training regimes were tried.

For amplifier 116, the neural network was trained on the first 400 sample points and the first 1600 sample points. This was done to distinguish between the different behaviors of the amplifier as amplifier 116 has both correlated and anticorrelated parts. The first 400 samples cover consistently anticorrelated behavior. Fig. 4.12 illustrates the neural network behavior for this training set. Fig. 4.12 indicates that the neural network learned the anticorrelated behavior and exhibits this behavior for the duration of the simulation. The first 1600 samples however covers both correlated and anticorrelated behavior. Fig. 4.13 illustrates the neural network behavior for this training set. Fig. 4.13 indicates that the neural network did not learn either the correlated or anticorrelated behavior, but instead learned the temperature.

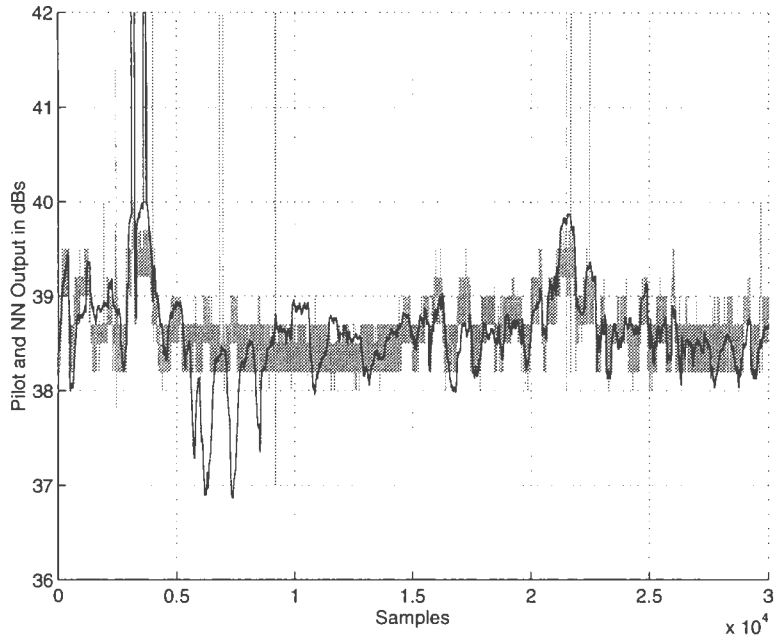

FIGURE 4.12 Amplifier 116; light grey trace forward pilot; dark line neural network. The neural network was trained on the first 400 samples.

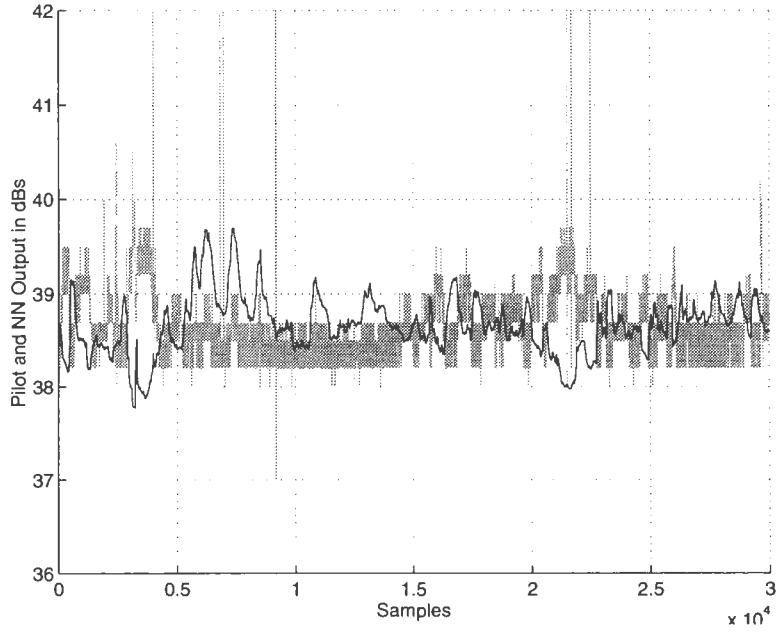

FIGURE 4.13 Amplifier 116; light grey trace forward pilot; dark line neural network. The neural network was trained on the first 1600 samples.

For amplifier 121 the neural network was trained on the first 1000 samples and first 4000 samples. Fig. 4.14 shows the results of training the neural network on the first 1000 data points.

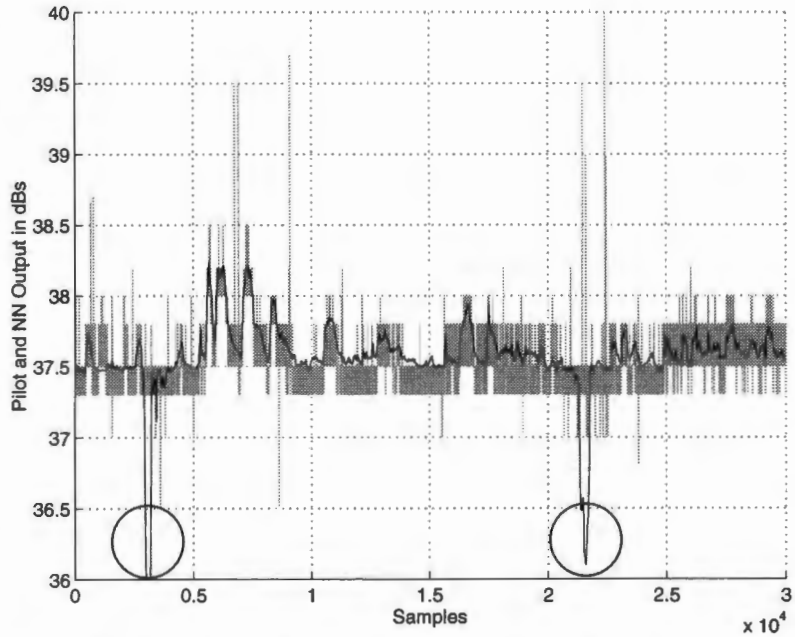

FIGURE 4.14 Amplifier 121; light grey trace forward pilot; dark line neural network. The neural network was trained on the first 1000 samples. The circles indicate where the neural network fails to track due to drops in the temperature.

As Fig. 4.14 illustrates the neural network does a good job of tracking the forward pilot except in two areas. These drops in the neural network output occur due to the temperature dropping far outside the training range. By extending the training range over the first 4000 data points, the entire temperature range is encompassed. Fig. 4.15 illustrates the results. Fig. 4.15 shows that by using the extended temperature range the neural network tracks the forward pilot for the duration of the two months.

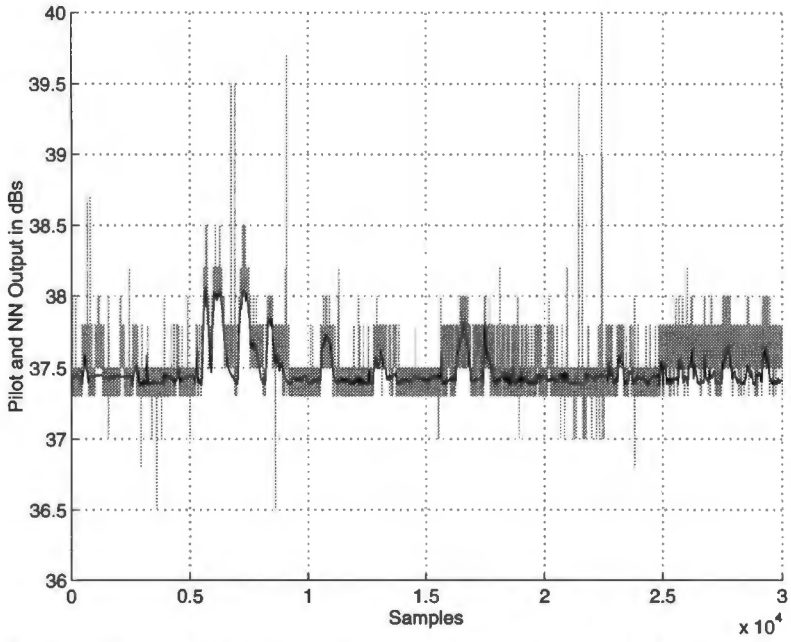

FIGURE 4.15 Amplifier 121; light grey trace forward pilot; dark line neural network. The neural network was trained on the first 4000 samples.

These experiments with amplifiers 116 and 121 yielded two important results. The first result indicates that if the neural network is trained on correlated data, then the neural network will learn the correlated behavior. Even if the amplifier exhibits anticorrelated behavior outside the training range, the neural network will continue with the correlated behavior. The same is true if the neural network is trained on anticorrelated data. However if the neural network is trained on data consisting of both correlated and anticorrelated parts, the neural network will have difficultly learning anything. This intuitively makes sense as we are trying to force the neural network to learn contradictory behavior.

Secondly these experiments indicate that after a while the neural network stops tracking the forward pilot even though there is no behavior change in the forward pilot. Analysis of the temperature indicates that the temperature at which the neural network failed to track the forward pilot was outside of the range of the temperature used to train the neural network. This and other experiments indicate that if the input to the neural network falls outside the neural network training range, the neural network will fail to track

the signal. It is important to note though that the neural network can extrapolate to some extent; how much it can extrapolate is discussed in the next chapter.

The next set of experiments are to see if these results are generally true for other amplifiers.

4.3.2 Large Scale Experiments

The next set of experiments involved using the neural network simulator to learn and simulate fifty five trunk amplifiers. Forty four of the trunk amplifiers are high pilot amplifiers and eleven are low pilot amplifiers. At the time of the data collection the trunk amplifiers were configured in the tree network shown in Figs. 4.16 and 4.17.

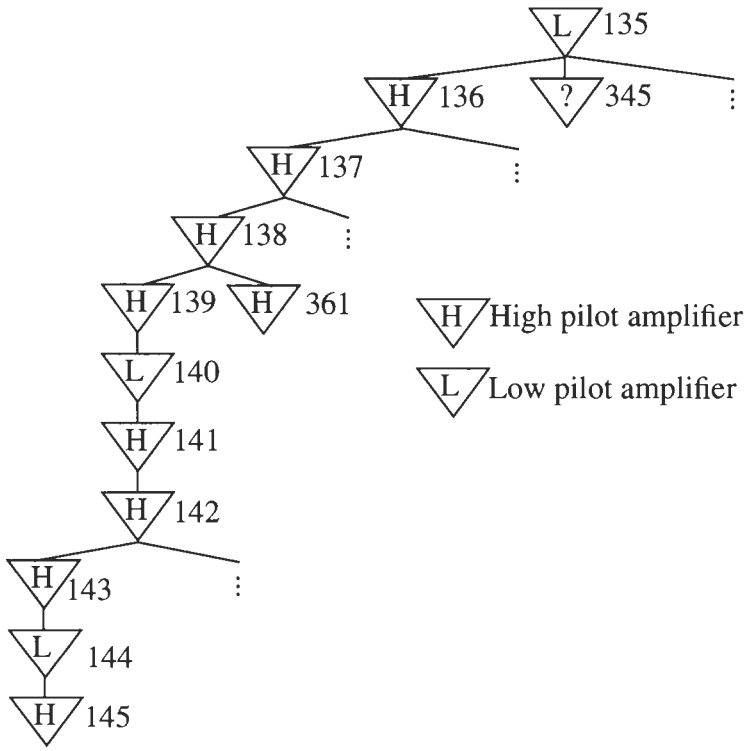

FIGURE 4.16 Partial configuration of amplifier network from Newmarket Ont. in Oct. and Nov. of 1995. Note that the parent of amplifier 135 is unknown.

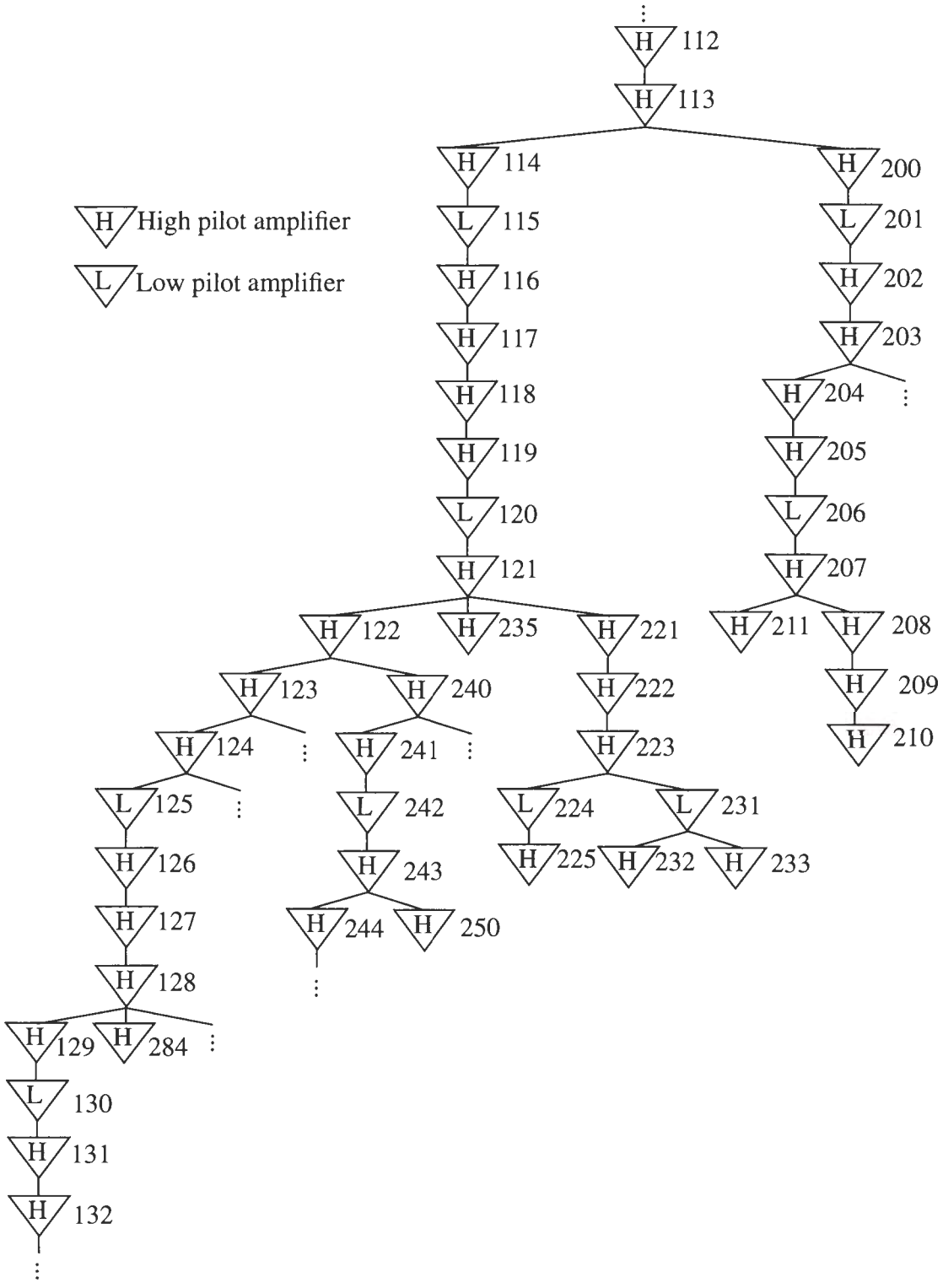

FIGURE 4.17 Partial configuration of amplifier network from Newmarket Ont. in Oct. and Nov. of 1995.

The data for the trunk amplifiers comes from October and November of 1995. The goals of this experiment are to see if the neural network simulator could learn the behavior of a variety of different trunk amplifiers, and to examine the various different types of behaviors the amplifiers exhibit. For each trunk amplifier, a neural network with the structure given in Tables 4.1 and 4.2 and Fig. 4.9 was trained on the first 1000 data points from October. These experiments use initial path training heuristic with three initial search paths. A summary of the results for high pilot amplifiers are shown in Table 4.3 and a summary of the results for low pilots are shown in Table 4.4. The full results can be found in Appendix B. The amplifier behavior is divided into three broad categories: *strictly correlated* means the amplifier is consistently correlated for the entire two month period; *correlated with level changes* means the amplifier is consistently correlated for the entire two months, but at least one sudden level change of at least 1 dB occurs; and *mix of correlated and anticorrelated* means a mix of these two behaviors over the two month period. The training and simulation results are divided into four categories: *learned correlated* means the neural network learned correlated behavior and remained consistently correlated when simulated on the two months of data; *learned anticorrelated* means the neural network learned anticorrelated behavior and remained consistently anticorrelated when simulated on the two months of data; *learned correlated extend training* means the neural network learned correlated behavior but at some point during the simulation the neural network fails to track due to the temperature falling far outside the training range; *did not learn* means the neural network did not learn either correlated or anticorrelated behavior.

TABLE 4.3 Summary of high pilot results

Amp. behavior	Learned correlated	Learned anticorrelated	Learned correlated, extend training	Did not learn	Total
Strictly correlated	6	0	14	0	20
Correlated with level changes	3	0	9	0	12
Mix of correlated and anticorrelated	2	5	4	1	12

TABLE 4.4 Summary of low pilot results

Amp. behavior	Learned correlated	Learned anticorrelated	Learned correlated, extend training	Did not learn	Total
Strictly correlated	0	0	7	0	7
Correlated with level changes	1	0	0	0	1
Mix of correlated and anticorrelated	1	0	2	0	3

From Tables 4.3 and 4.4 we can observe that for most trunk amplifiers that are correlated the training temperature range is not sufficient for the neural network to learn the trunk amplifier's entire behavior over the two months (23 / 32 for high pilot amplifiers, and 7 / 7 for low pilot amplifiers). It is interesting to note that twenty four high pilot amplifiers and four low pilot amplifiers (i.e. approximately 53% of the trunk amplifiers) are anticorrelated and / or having level changes. It is also interesting to note that amplifiers 116, 117, and 119; and 240, 241, and 243 all seem to have problems and that these amplifiers are connected in cascade with each other. The figures below illustrate some of the results. Fig. 4.18 illustrates one of the better results from amplifier number 138. Fig. 4.18 indicates that the neural network tracks extremely well for all 28 000 sample points. Fig. 4.19 illustrates one of the suspected faulty amplifiers; amplifier number 116. Fig. 4.20 illustrates one of the amplifiers that is correlated and appears to be behaving correctly, but needs an extended temperature training range. Fig. 4.20 indicates that the neural network tracks well for the first 12 000 samples, but after that there is as much as a 0.8 dBmV difference between the neural network and the forward pilot. If we analyze the temperature range over the two months we can observe that the temperature does drop to much lower than that of the training set. Fig. 4.21 illustrates the temperature for amplifier number 112 in Fig. 4.20.

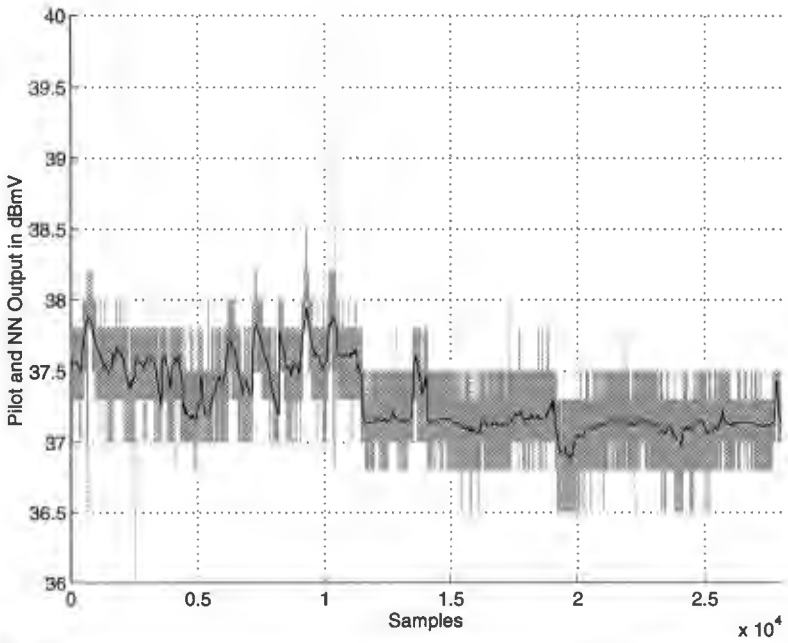

FIGURE 4.18 Training and simulation results for amplifier number 138. Light gray trace forward pilot; dark line neural network.

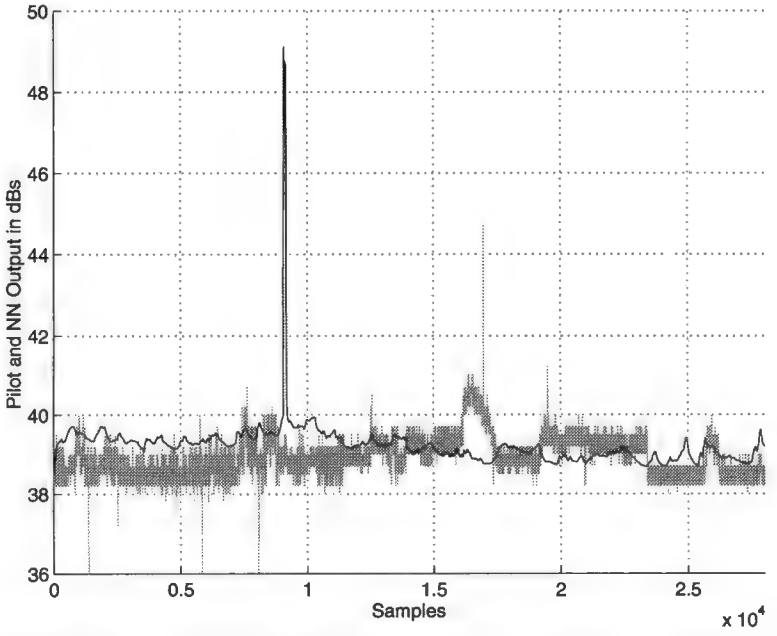

FIGURE 4.19 Training and simulation results for amplifier number 116. Light gray trace forward pilot; dark line neural network.

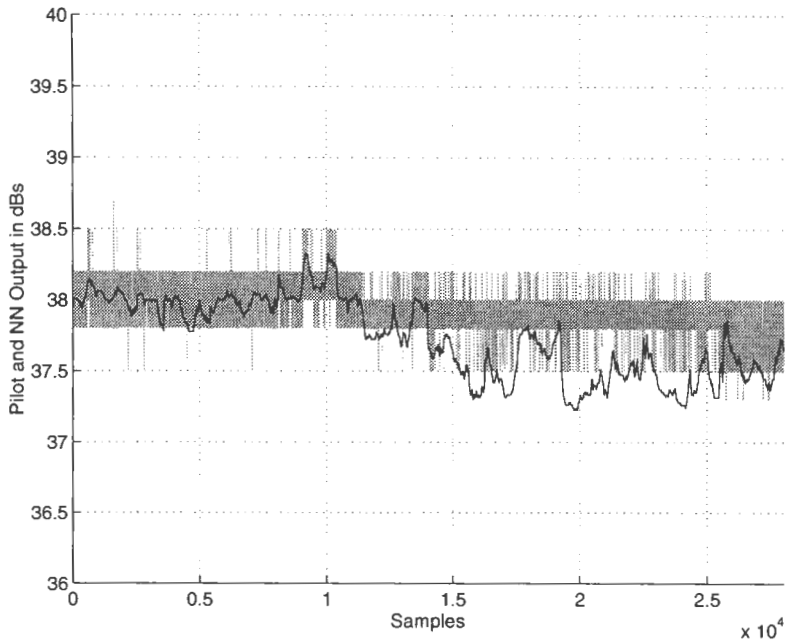

FIGURE 4.20 Training and simulation results for amplifier number 112. Light gray trace forward pilot; dark line neural network. Note the drop in the level of the neural network after 12000 samples.

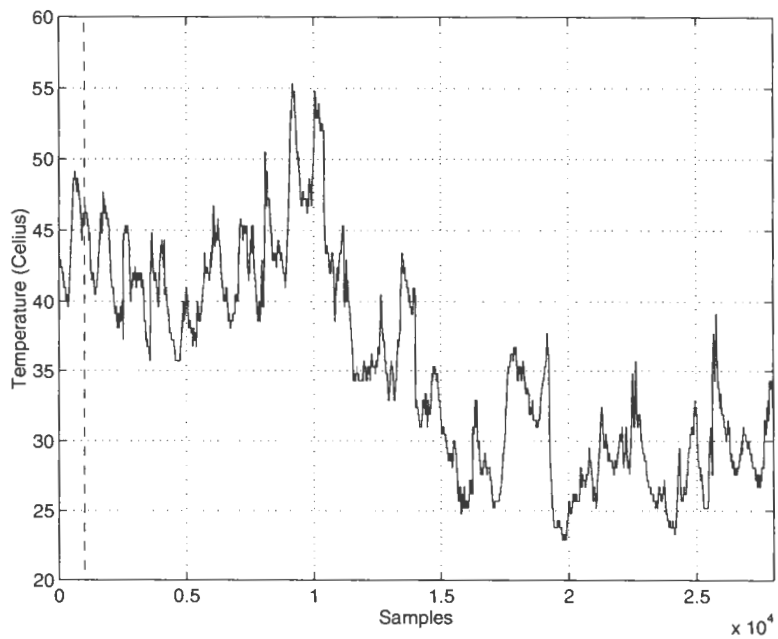

FIGURE 4.21 Amplifier 112 temperature; dashed line indicates where the training set ends.

From Fig. 4.21 we can observe that the temperature in the training set ranges from approximately 39 °C to 48 °C, while the temperature over the two month period ranges from approximately 23 °C to 55 °C. This first large scale experiment indicated that the neural network could track the forward pilot well as long as the temperature did not exceed the training range by too much.

The next experiment on this data set was to see if the neural network would be able to model the trunk amplifier for the entire two months if the neural network was trained on some lower temperature data. The data for this experiment is the first 1000 data points from October and the first 1000 data points from November as shown in Fig. 4.22.

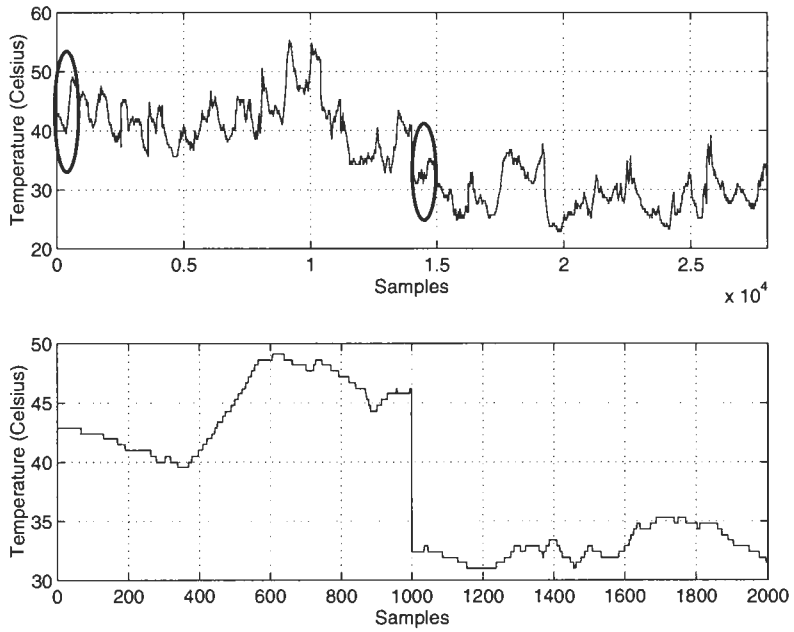

FIGURE 4.22 Extended temperature range for amplifier 112. The circles indicate the regions where the training data were taken from.

These data sets are extracted from the data and concatenated together so that the neural network is trained on 2000 data points. Despite the discontinuity in the training set, the neural network has no problem learning the behavior. The summary of results for the high and low pilot amplifiers for this experiment are given in Tables 4.5 and 4.6 respectively. (The full results are given in Appendix B).

TABLE 4.5 High pilot results

Amp. behavior	Learned correlated	Learned anticorrel ated	Learned correlated, extend training	Did not learn	Total
Strictly correlated	13	0	7	0	20
Correlated with level changes	11	0	1	0	12
Mix of correlated and anticorrelated	5	0	4	3	12

TABLE 4.6 Low pilot results

Amp. behavior	Learned correlated	Learned anticorrel ated	Learned correlated, extend training	Did not learn	Total
Strictly correlated	7	0	0	0	7
Correlated with level changes	0	0	0	1	1
Mix of correlated and anticorrelated	1	1	1	0	3

In this case, we can observe from the Tables 4.5 and 4.6 that the neural network was able to model most of the correlated trunk amplifiers for both months. For the amplifiers that were previously in the learned correlated extend training category, and are now in the learned correlated category, the average and RMS error is lower in all cases. Fig. 4.23 illustrates the results for amplifier 112. Fig 4.23 illustrates how the neural network now tracks amplifier 112 for the entire two months.

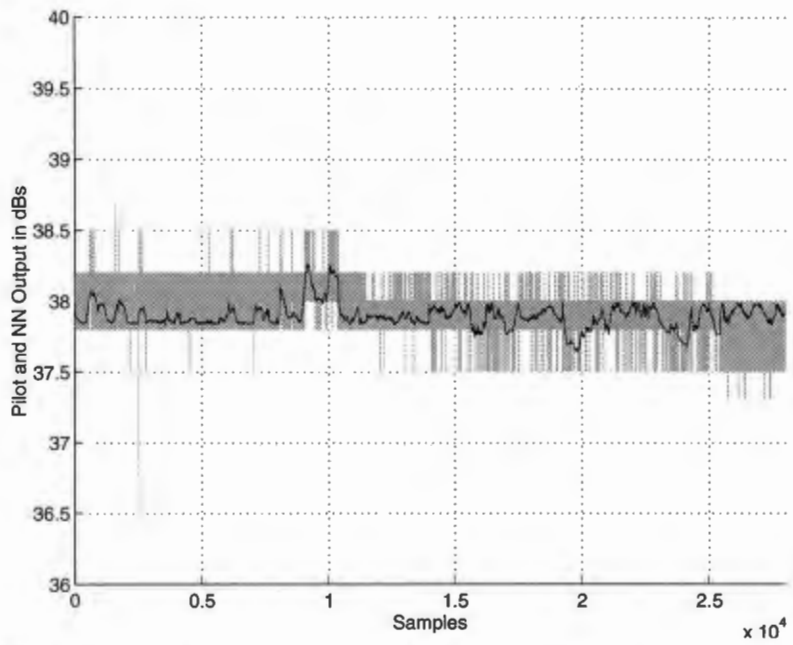

FIGURE 4.23 Training and simulation results for amplifier number 112. Light gray trace forward pilot; dark line neural network.

Given these results we can conclude that the neural network can learn the amplifiers behavior if the trunk amplifier's behavior in the training set is consistently correlated or consistently anticorrelated. The neural network will model the amplifier behavior with a reasonable degree of accuracy as long as the amplifier's behavior remains consistent and the temperature does not exceed the training temperature range by too much. The next step is to consider the case of more than two months of data.

4.4 Summary

This chapter described an application for the neural network simulator. This chapter gave some details on the operation of C-COR cable television trunk amplifiers and how this amplifier can be modelled using the neural network simulator. The recurrent neural network can not only accurately model the trunk amplifiers' behavior, but the neural network can also be used to detect behavior changes as well. Using the results of these experiments, the next chapter studies how the neural network simulator can be used for the long term monitoring of the trunk amplifiers.

Chapter 5

Recurrent Neural Network Application

Ch. 4 gave an introduction to the application for the asymptotically stable recurrent neural network simulator in modelling. The long-term goal of this application is to use the neural network simulator to monitor a trunk amplifier for the duration of the trunk amplifier's lifetime. Having shown empirically that the neural network could track the forward pilot as long as the temperature was within the training range, the next set of experiments analyze the amplifier's behavior for a much longer period of time; these experiments use over seven months of data. The experiments in Ch. 4 indicated that there are three main problems to solve in order to use the neural network simulator for long term monitoring. When the temperature falls out of the range of the training temperature, the neural network stops tracking the forward pilot. This indicates that some method of retraining the neural network is required when the temperature drops out of range. Ch. 4 illustrated that a 'large difference' between the neural network output and the forward pilot indicated a behavior change and possibly a fault in the amplifier. To use the neural network for detecting these behavior changes we need to quantify what sort of difference and for how long such a difference must exist to constitute a behavior change. The final consideration is the training set. The experiments in Ch. 4 indicated that the neural network can not learn inconsistent behavior. This means that some way of ascertaining how 'good' the training set is needs to be developed. Thus the goal of Ch. 5 is to determine the rules for retraining the neural network and detecting behavior changes. These rules will eventually comprise part of an expert system for monitoring the trunk amplifiers.

In this experiment five amplifiers 109, 129, 202, 221, and 224 with data from October 1995 to April 1996 are used. Amplifiers 109 and 129 behave 'well' for the duration of the seven months while amplifiers 202, 221, and 224 have behavior changes of varying degrees of severity. Figs. 5.1 to 5.5 illustrate the forward pilot and temperature for the five amplifiers.

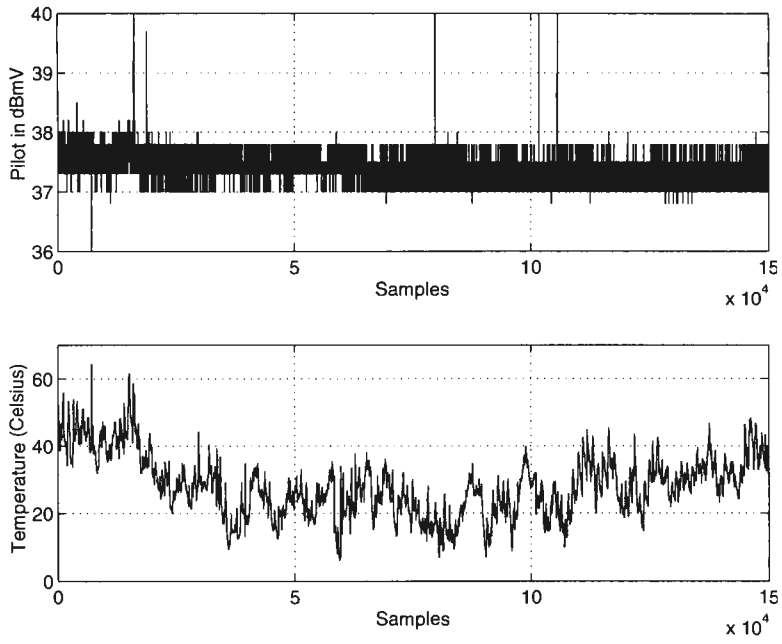

FIGURE 5.1 Forward pilot and temperature from trunk amplifier 129.

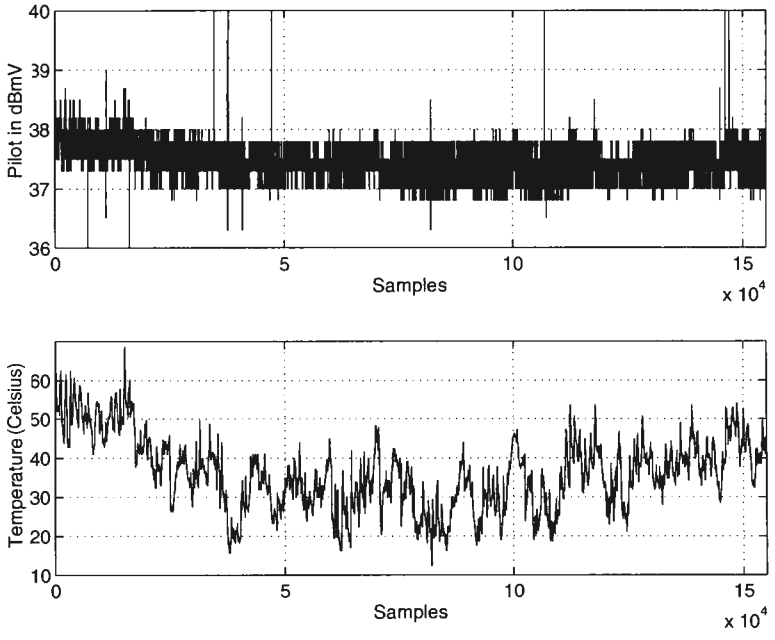

FIGURE 5.2 Forward pilot and temperature from trunk amplifier 144.

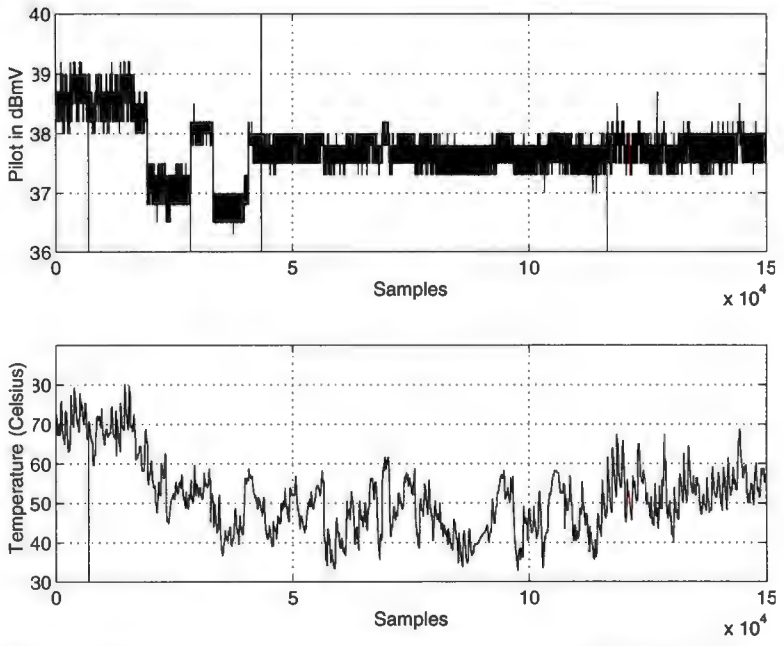

FIGURE 5.3 Forward pilot and temperature from trunk amplifier 202.

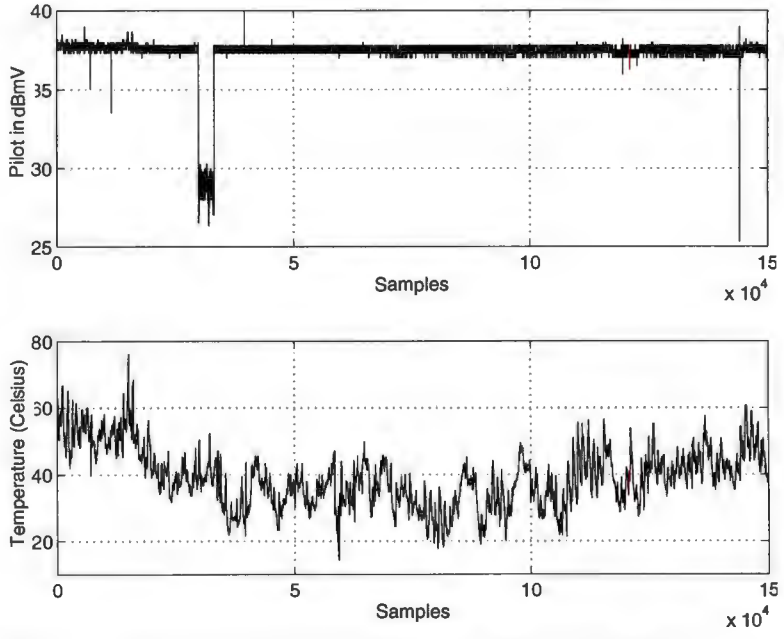

FIGURE 5.4 Forward pilot and temperature from trunk amplifier 221.

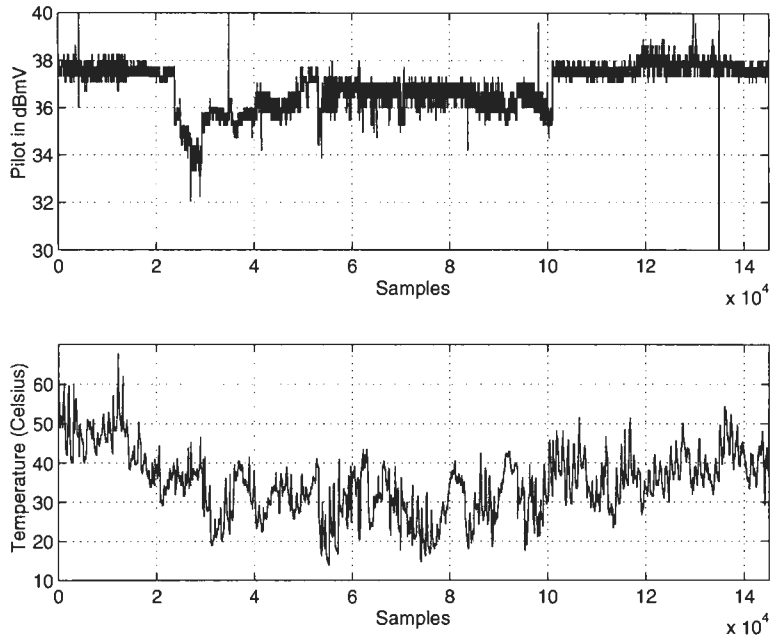

FIGURE 5.5 Forward pilot and temperature from trunk amplifier 224.

For these five amplifiers, the temperature ranges from approximately 10 °C to 80 °C.

The first set of experiments involved deriving the retraining rules for the temperature.

5.1 Temperature Experiments

The experiments of Ch. 4 indicated that when the temperature falls significantly outside of the training temperature range, the neural network can no longer correctly track the forward pilot. In such cases the difference between the neural network output and the forward pilot level can be large. As amplifier behavior changes are detected when the difference between the neural network output and the forward pilot level is 'large' these drops in the neural network may be mistaken as amplifier behavior changes.

The initial idea for retraining the neural network was to let the large differences due to out of range temperatures occur and then decide later whether this was a behavior change in the trunk amplifier or a temperature change. It was realized early on that there

was an easier approach to solving the neural network's failure to track due to temperature changes. The approach was to simply prevent this failure from happening. In doing this then any large difference between the neural network output and the forward pilot level would be due to a change in forward pilot behavior or a poor training set. To prevent the neural network from failing to track due to temperature changes merely requires the simulator to monitor the input to the neural network during the simulation phase. If the temperature exceeds the training temperature range by a certain amount, the simulation routine stops and flags the main routine to retrain the neural network. Once the neural network is retrained, the simulation continues.

The question now is by how much must the temperature exceed the training range before we say we need to retrain. In other words, if the training temperature range was $[T_{min}, T_{max}]$ then the tolerated range can be considered as $[T_{min} - tol, T_{max} + tol]$. As stated previously, the neural network can extrapolate to some extend so the tolerance should certainly be nonzero. Another reason to have a nonzero tolerance is if the tolerance was zero, the neural network would probably need to retrain just after it finished training. The tolerance must be chosen so that not only no drops due to temperature will occur, but also so that the neural network does not excessively retrain. As the tolerance had to be determined empirically, amplifier 144 was initially used as 144 is a well behaved amplifier with no behavior changes to confuse the issue. Initially the tolerance was set to 10 °C. The neural network would train on the first 1000 points in the data set starting with initial random weights. When the temperature exceeds the training range, the simulator would take 1000 points starting from where the temperature was out of range, throw out the old training data and retrain the neural network using initial random weights. The training would use the initial path search heuristic with three initial paths. A problem immediately evident was that starting from initial random weights each time the neural network had to be retrained caused the retraining to be very slow. Also the previous weights 'stored' information so by starting from random weights this information was being lost. It was thus decided to use the previous set of weights as the initial weights when retraining. Even so, experiments indicated that a tolerance of 10 °C is too high as indicated in Fig. 5.6.

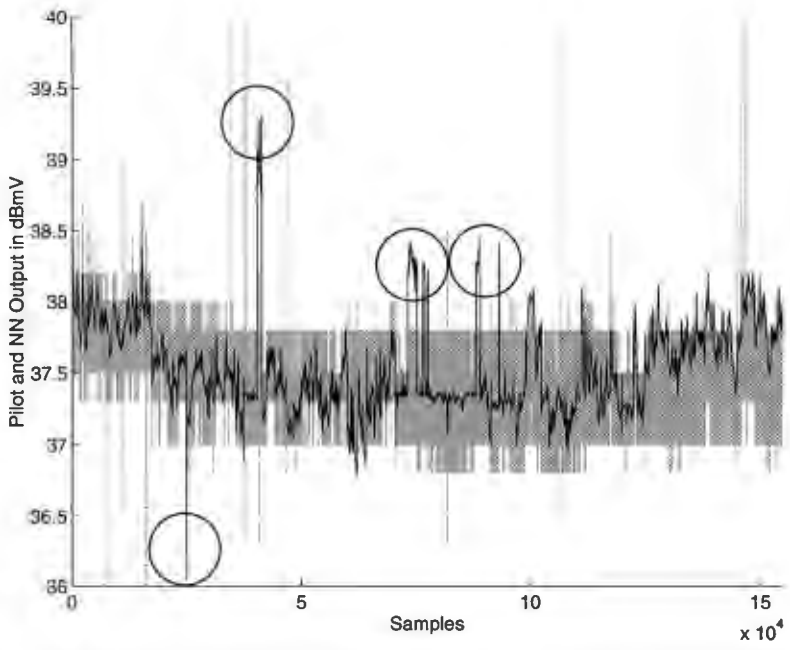

FIGURE 5.6 Amplifier 144; dark grey trace forward pilot; black line neural network. Circles indicate where the neural network failed to track due to temperature changes.

Fig. 5.6 indicates several large differences between the neural network and the forward pilot that would be mistaken as amplifier behavior changes. Because of this the tolerance was reduced to 6 °C. Although there were no problems for amplifier 144 as shown in Fig. 5.7, one serious failure to track did occur in amplifier 129 and shown in Fig. 5.8.

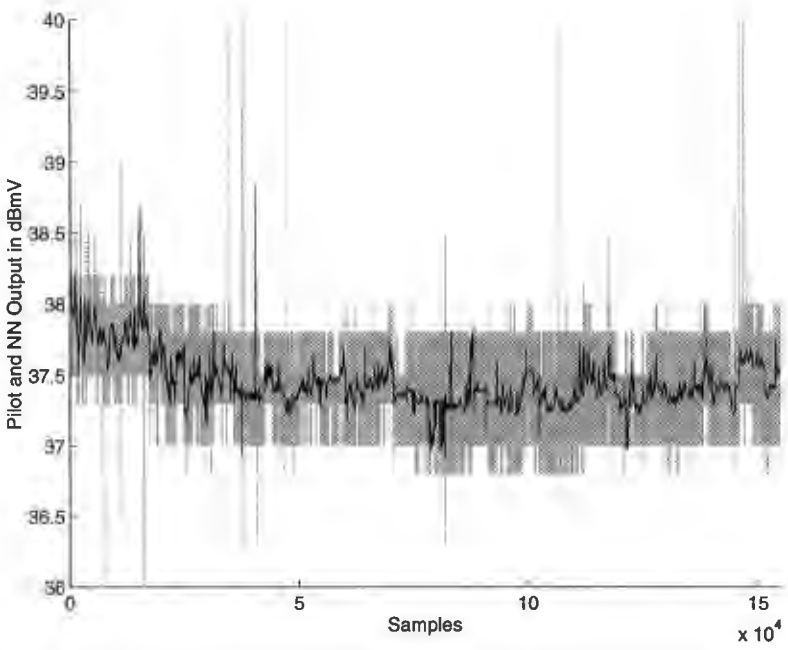

FIGURE 5.7 Amplifier 144; dark grey trace forward pilot; black line neural network.

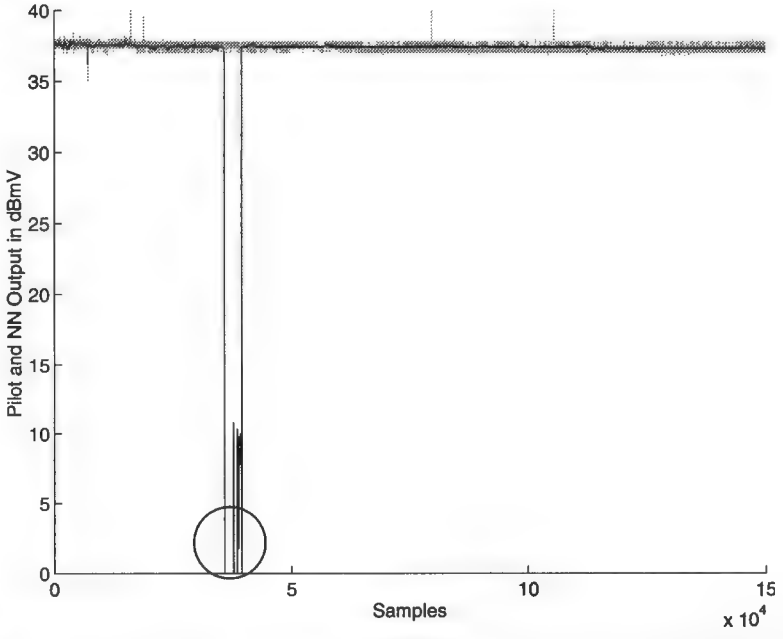

FIGURE 5.8 Amplifier 129; dark grey trace forward pilot; black line neural network. Circle indicates where the neural network failed to track due to temperature change.

Because of the failure to track due to temperature in amplifier 129, the tolerance was further reduced to 4 °C. At this point, as shown in Fig 5.9, there are no longer any failures to track for amplifier 129, however the neural network was now retraining far too often.

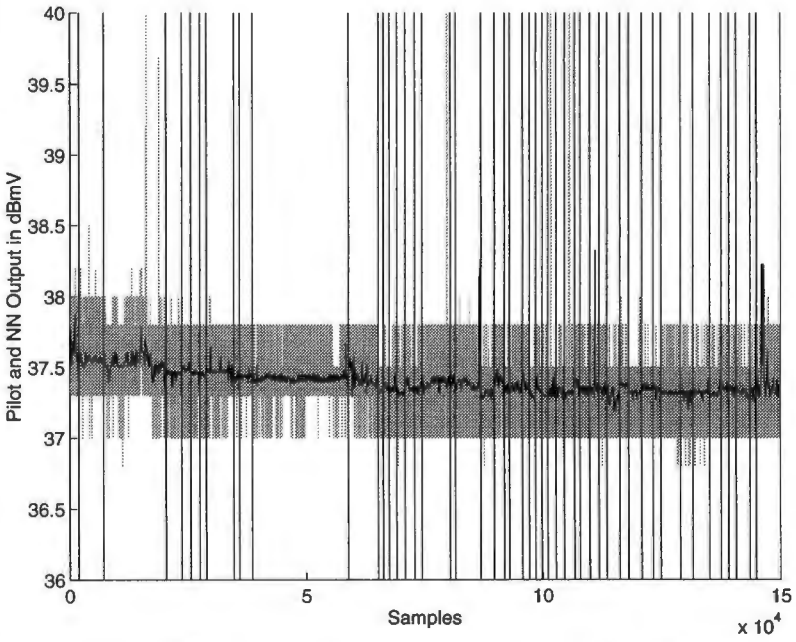

FIGURE 5.9 Amplifier 129; dark grey trace forward pilot; black line neural network. Black vertical lines indicate where retraining occurred.

Examining this situation indicated that much of this retraining was unnecessary. The temperature tends to exhibit a 'sawtooth' profile as illustrated in Fig. 5.2. This profile is due to the heating and cooling of the trunk amplifier due to the weather. With the tolerance set so low, the neural network retrains at the top of the peak and then falls out of range at the bottom of the peak and thus needs to retrain. However the temperature then peaks again at an out of range temperature, but the behavior at this peak is basically the same as the previous peak. With such a low tolerance useful information is being thrown out unnecessarily. However increasing the tolerance causes failures to track due to temperature to occur.

The solution to this problem was to split the training set in half. Similar to the second experiment in Sec. 4.3.2., half the training set would consist of high temperature data and the other half would consist of low temperature data. If the temperature exceeds the high temperature limit $T_{max} + tol$, the 500 high temperature data points are 'thrown out'

and replaced with the 500 data points after the point where temperature was exceeded. The neural network would then be retrained using both the high and low set. A similar procedure is followed if the low temperature limit is violated. The experiment was tried with a tolerance of 6 °C. As Fig. 4.20 indicates the neural network does a reasonable job of tracking the data with few retrains.

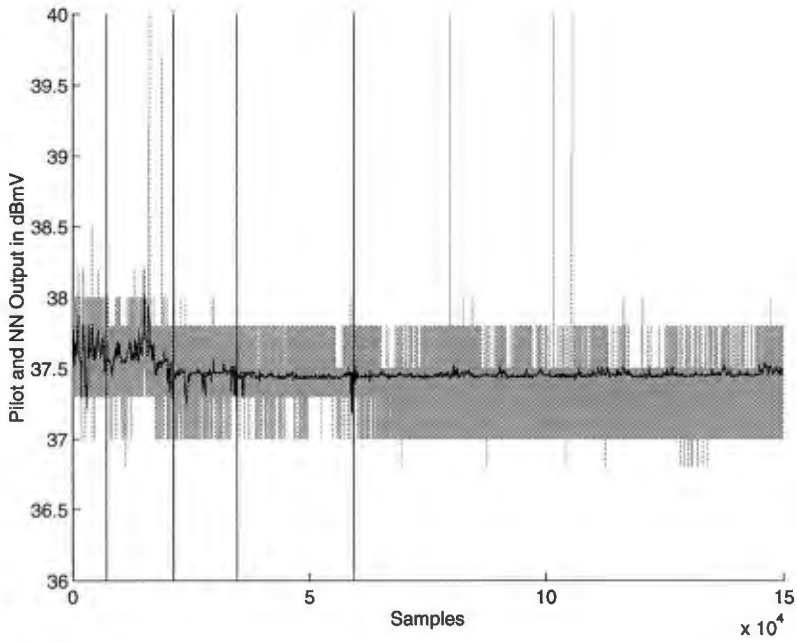

FIGURE 5.10 Amplifier 129; dark grey trace forward pilot; black line neural network. Black vertical lines indicate where retraining occurred.

However, Fig. 5.10 indicates that after the last retrain the neural network appears to be 'over generalizing'. This occurs as a result of the high and low training sets becoming too far apart. As these sets become far apart, the neural network becomes more general in its extrapolation between the two extremes. Another problem with this method is that eventually the neural network will not retrain as the high and low data sets will cover the entire temperature range. This is a problem as the trunk amplifier's behavior 'drifts' over time; i.e the trunk amplifier's behavior will slowly change over time even if there is no fault. Because of this, the neural network will need to be retrained occasionally. The method chosen to solve both these problems is to restrict the maximum difference between the high and low training set. Essentially a 'temperature window' is used. If the difference between the maximum of the high set and the minimum of the low set becomes greater

than 30 °C, then one of the sets is thrown out. The results of applying the 30 °C temperature window and a *tol* of 6 °C to amplifier 129 is shown in Fig. 5.11.

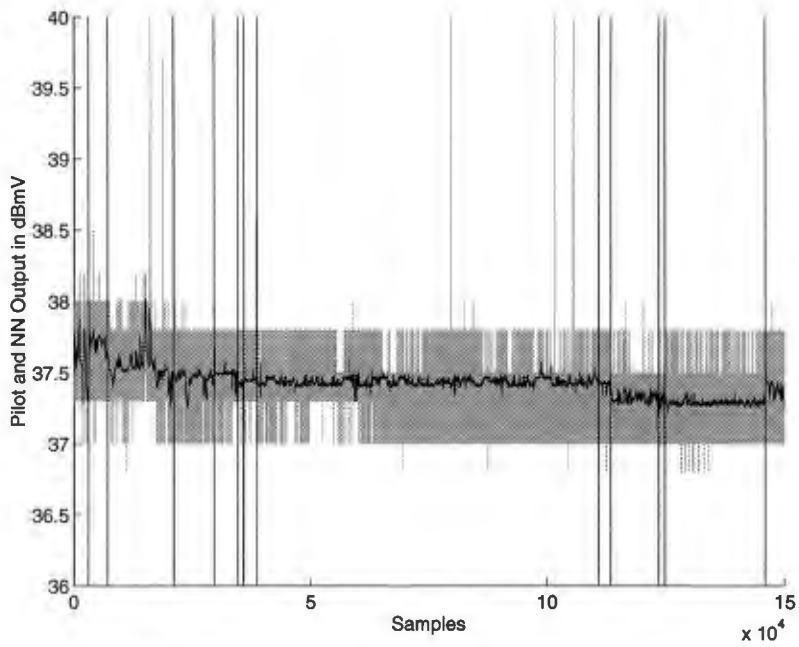

FIGURE 5.11 Amplifier 129; dark grey trace forward pilot; black line neural network. Black vertical lines indicate where retraining occurred.

Fig. 5.11 indicates that this method works. Thus the method for retraining the neural network for temperature changes is based on having two temperature training sets; a high set and a low set. The range of these sets are given as $[T_{min}^{low} - tol^{low}, T_{max}^{low} + tol^{low}]$ and $[T_{min}^{high} - tol^{high}, T_{max}^{high} + tol^{high}]$. The neural network retrains when the temperature goes outside of either the high or low range. If after collecting the new training data set, the difference between T_{max}^{high} and T_{min}^{low} exceeds 30 °C, then both old training sets are thrown out. Note that this is a simple model in that *tol* may vary with temperature and tol^{high} does not necessarily equal to tol^{low}.

Having established a reasonable method for retraining due to temperature, the next step was to determine what constitutes an amplifier behavior change and how to handle these occurrences.

5.2 Behavior Change Experiments

Having established a method for retraining due to temperature changes, the next step is to consider the amplifier behavior changes in the training. At this point we can not really say that we are doing fault detection, but rather the neural network is detecting changes in the amplifiers' behavior. When the amplifier's behavior changes, we want the neural network first to detect and then to try to track the new behavior. To detect these behavior changes we need to decide what exactly constitutes a behavior change. So far we have defined a behavior change as a 'large' difference between the forward pilot level and the neural network output, however what exactly a 'large' difference is needs to be quantified.

Originally it was thought to calculate the difference between the neural network and the forward pilot. However as stated in Ch. 4, there is a great deal of noise on the forward pilot signal due to the coarse quantization steps and the analog noise inherent in the signal. As a result, the noise bounds on the forward pilot are 40 to 50 percent of the average step size [23]. Thus any difference between the neural network and the forward pilot would need to be outside this noise bound in order to be considered a behavior change. With such a large noise bound it is possible that some behavior changes may be missed or delayed in their detection. Research indicates that the noise consists of high frequency components [23]. To reduce the noise and thus the noise bounds, the pilot was 'dequantized' by lowpass filtering it. Using techniques from [2], a fourth order digital elliptic lowpass filter with a cutoff frequency of 1% was derived assuming a sampling frequency of 1/77 Hz. The loss function for this filter is shown in Fig. 5.12. Fig 5.13 illustrates the result of lowpass filtering amplifier 202. Fig 5.13 illustrates the similarity between the neural network and the lowpassed forward pilot. It is important to note that in order to prevent the slow rise time of the filter from causing problems the filter's initial values are set to nonzero values in a similar manner as is done for the neural network in Sec. 3.2.4. Also as the filter can be slow in tracking large drops of the forward pilot, the filter values are reset for drops of more than 2 dBmV over ten samples.

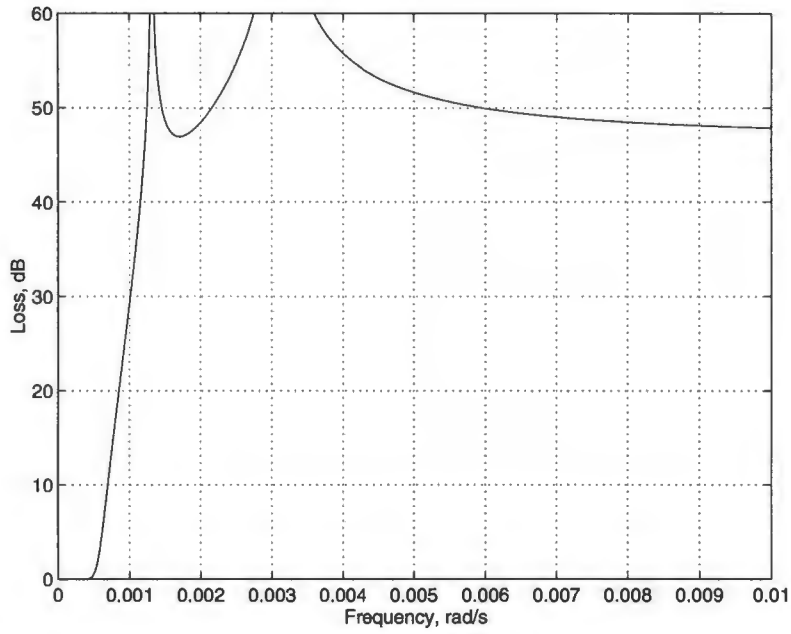

FIGURE 5.12 Loss function for a fourth order digital elliptic lowpass filter.

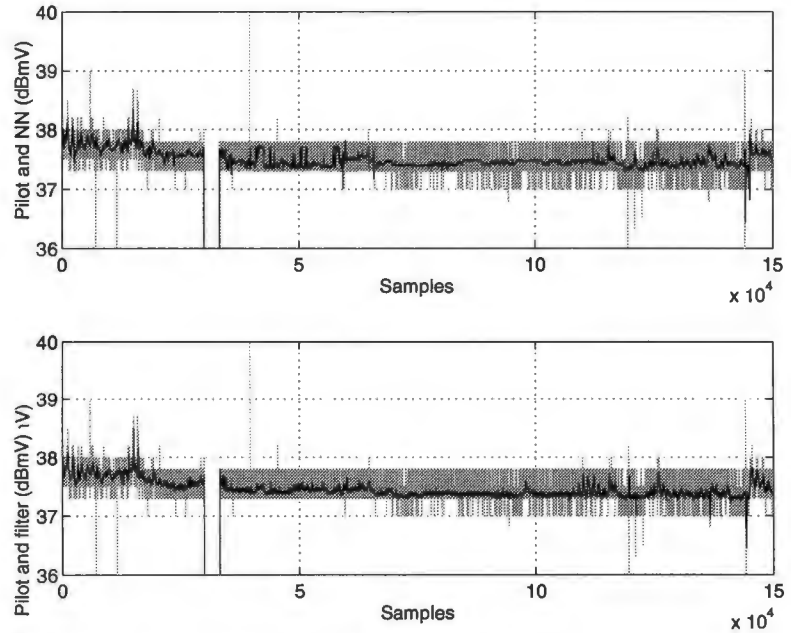

FIGURE 5.13 Results of lowpass filtering amplifier 202.

Experimentation with several data sets indicates that a 1 dBmV difference between the neural network and the lowpass forward pilot for five consecutive samples should constitute a behavior change. The large difference has to remain for a number of consecutive samples in order to avoid the possibility of spikes being misclassified as behavior changes. Experiments with this method and the temperature retraining method on amplifiers 202, 221, and 224 give the results shown in Figs. 5.14 to 5.16. Figs. 5.14 and 5.15 indicate that the neural network was able to detect the behavior changes in amplifiers 202 and 221 respectively.

Fig. 5.16 however indicates that the neural network had problems trying to learn the unstable behavior of amplifier 224. These large spikes in the neural network output appear to be caused by poor training or poor initial weights. The next section addresses this problem.

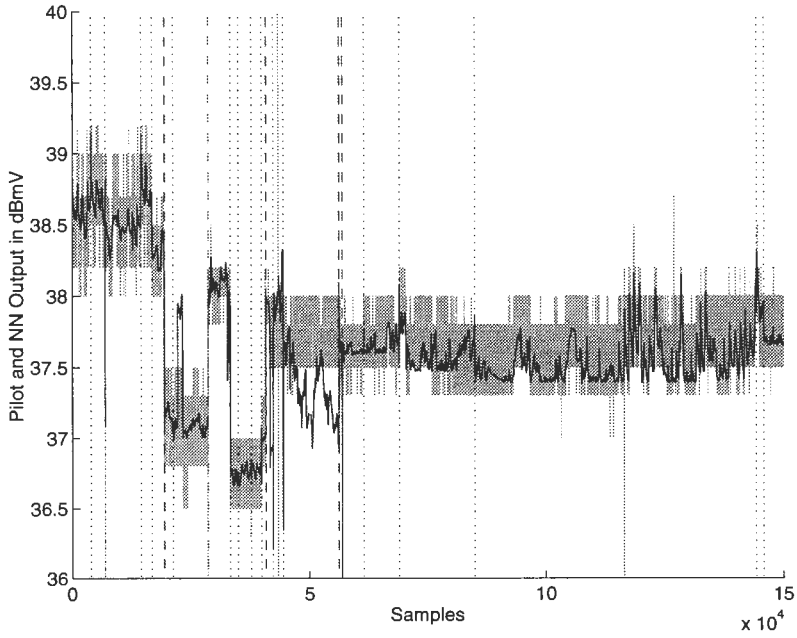

FIGURE 5.14 Amplifier 202; dark grey trace forward pilot; black line neural network. Dashed vertical lines indicate where retraining due behavior change occurred. Dotted vertical lines indicate where retraining due temperature change occurred.

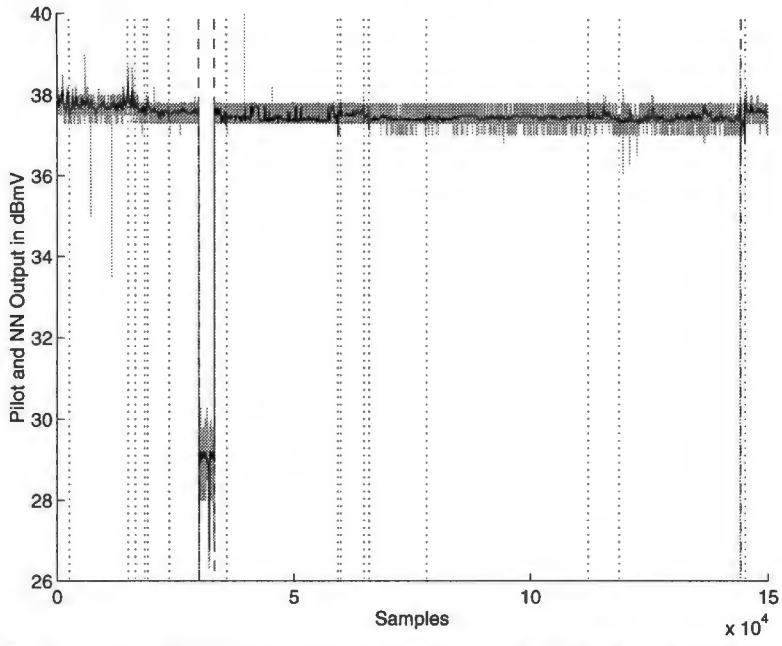

FIGURE 5.15 Amplifier 221; dark grey trace forward pilot; black line neural network. Dashed vertical lines indicate where retraining due behavior change occurred. Dotted vertical lines indicate where retraining due temperature change occurred.

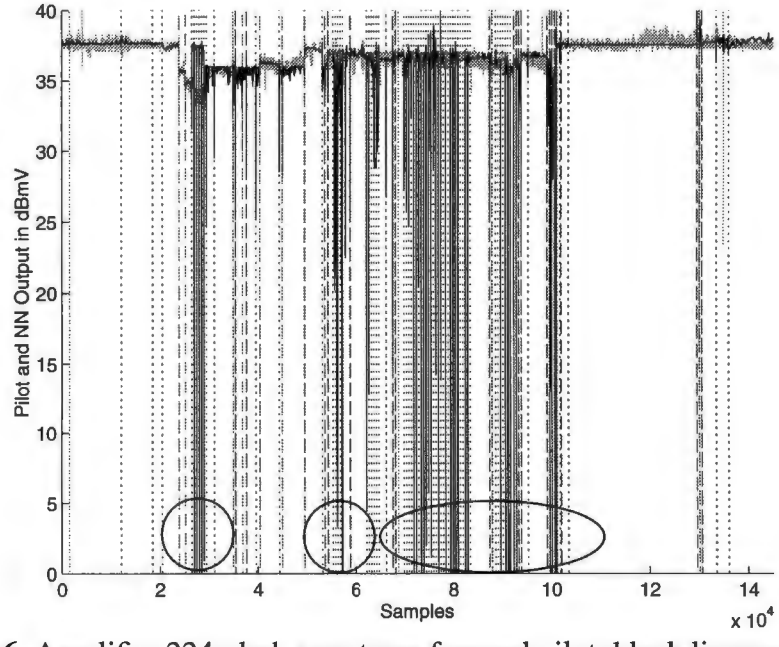

FIGURE 5.16 Amplifier 224; dark grey trace forward pilot; black line neural network. Dashed vertical lines indicate where retraining due behavior change occurred. Dotted vertical lines indicate where retraining due temperature change occurred. Circles indicate where the neural network failed to track due to temperature changes.

5.3 Poor Training Experiments

Results from amplifier 224 indicate that poor training sets or poor initial weights can cause serious problems for the neural network and possibly false alarms. To counter this problem the mean square error per sample is calculated. If this value is 'large' then this indicates either a poor training set or poor initial weights. In order to differentiate between these two, after a high training error occurs the first time, the initial weights are randomized and the training is attempted again. If the training error is still unacceptably high, the training set is assumed to be poor. In this case this data is skipped over and the training is tried again.

Using this method as well as the retraining rules from Secs. 5.1 and 5.2, amplifier 224 was simulated again. The results are given in Fig. 5.17. Fig. 5.17 indicates that the neural network tracks better than in Fig. 5.16; no large spikes are present.

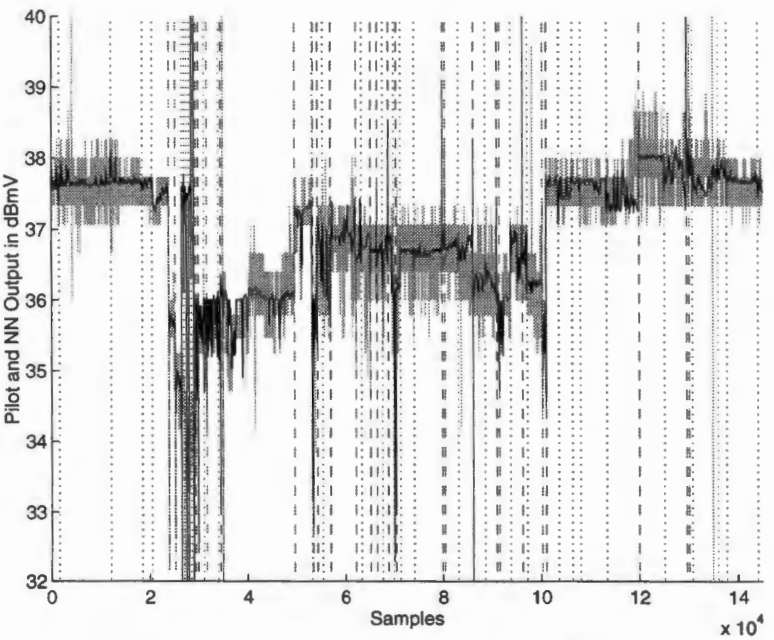

FIGURE 5.17 Amplifier 224; dark grey trace forward pilot; black line neural network. Dashed vertical lines indicate where retraining due behavior change occurred. Dotted vertical lines indicate where retraining due temperature change occurred.

5.4 Final Training Heuristics

The final heuristics used are illustrated in pseudo code form in Table 5.1

TABLE 5.1 Final training heuristics

```
Initialize
Do forever
  Loop until training acceptable
    Train neural network (as in Sec. 5.3)
    Check 'goodness' of training
  Find range of training temperature
  Loop until need to retrain
    Simulate neural network
    Check if a difference of greater than 1 dBmV exists between the forward pilot
    and the neural network for greater than five samples
      Flag retrain due to behavior change
    If no large difference check input temperature during simulation
      If input temperature is less than T_min^low - tol^low
        Flag retrain for low temperature
      If input temperature is greater than T_max^high - tol^high
        Flag retrain for high temperature
  Check if we need to retrain for low temperatures
  Else check if we need to retrain for high temperatures
  Else check if we need to retrain due to difference
```

5.5 Summary

This chapter described the solution to three problems in using the neural network simulator for behavior change detection of the trunk amplifiers. The problem of temperature falling out of the training range was solved by devising a retraining algorithm based on temperature changes. By reducing the noise bounds on the forward pilot using a lowpass filter a method for detecting behavior changes was developed. Finally a method for handling poor training sets and poor initial weights was devised. The next chapter details what further work is necessary for this application to meet its goals.

Chapter 6

Conclusions

6.1 Summary of Work

This thesis described the biological basis for some types artificial neural networks as well as the historical development of artificial neural networks. The thesis went on to focus on recurrent neural networks. In particular, a special type of recurrent neural network was examined and proven to be asymptotically stable. A back propagation training technique for the asymptotically stable recurrent neural network was derived. By citing various papers, theoretical evidence was presented on the ability of recurrent neural networks to model nonlinear systems. Ch. 3 described the simulator based on the asymptotically stable recurrent neural networks and the theory in Ch. 2. Ch. 3 also described the heuristics used to improve the training of the neural network.

Ch. 4 described the main area of the research; the use of the asymptotically stable recurrent neural networks for monitoring in a cable television trunk amplifier network. Experiments in Ch. 4 with a large number of amplifiers exhibiting a variety of different behaviors indicated that the neural network could accurately model a given trunk amplifier as long as the behavior of the amplifier remained consistent and the temperature did not deviate much from the training range.

Ch. 5 tried to solve three of the major problems for using the neural network simulator for the long-term monitoring of the trunk amplifiers. This consisted of developing a retraining algorithm based on the temperature changes, a retraining algorithm for behavior changes, and a method for handling a poor training sets and poor initial weights.

6.2 Future Work

This thesis covered both the theoretical and application aspects of the asymptotically stable recurrent neural network. There is a variety of work to be done in both areas.

6.2.1 Future Theoretical Work

As stated in Sec. 2.4, there is currently no proof that the asymptotically stable recurrent neural network is a universal approximator. Although anecdotal evidence seems to indicate that the asymptotically stable recurrent neural network is a universal approximator, deriving the actual proof would be a valuable contribution to research.

Another unanswered theoretical question regards the internal behavior of the recurrent neural network. Currently the neural network used by the simulator is little more than a black box. Little is known how the neurons interact with each other inside the network. Knowledge of this behavior could yield valuable information on the minimum number of neurons and the type of interconnections required.

6.2.2 Future Application Work

Much work needs to be done before the simulator can be used for the long term, real-time monitoring of the trunk amplifiers. There are three main areas of work.

6.2.2.1 Data Collection

Currently the simulator reads all the trunk amplifier data into memory all at once from a data file. For the simulator to be used for monitoring a trunk amplifier in real-time, the data must arrive to the simulator in a data stream. This means that the simulator must not only collect the data, but preprocess it as well. As the data points come in only once every one to five minutes, collecting a suitable number of samples to train the neural network could take on the order of hours to days. Such lengthy data collection could significantly delay the detection of faults. Experiments need to be performed that train the simulator neural network with very small data sets (less than 100 data points) to determine if the neural network can still learn the trunk amplifier's behavior.

Another possible solution to the data collection problem is that within the next year the INMS software will be replaced with new software called Demon. Unlike the INMS software, Demon can poll an amplifier continuously (i.e every few seconds) [25]. Doing this would allow a suitable amount of data for retraining to be collected in only a

matter of minutes. Although this seems like a reasonable solution to the problem of data collection, there is a potentially serious drawback. Changing the sampling rate may alter the dynamics of the system. Thus a neural network trained on data sampled only a few seconds apart may not model the data sampled a minute apart. Experiments would need to be performed to determine this.

Another solution is to developed a multithreaded simulator. In such a simulator there would be a 'master process' and a variety of 'slave processes'. Under ordinary circumstances, the master process would sleep while a slave process would monitor the trunk amplifier. If it was necessary to retrain, the slave process would wake up the master process and the master process would create a slave process to collect the data. After collecting an hours worth of data, the slave process would give the data to the master process and the master process would create a slave process to retrain the neural network. At this point three slave process would exist; one continuing to collect the data, one to retrain the neural network, and another to monitor the amplifier. Fig. 6.1 illustrates the multithreaded simulator at this point.

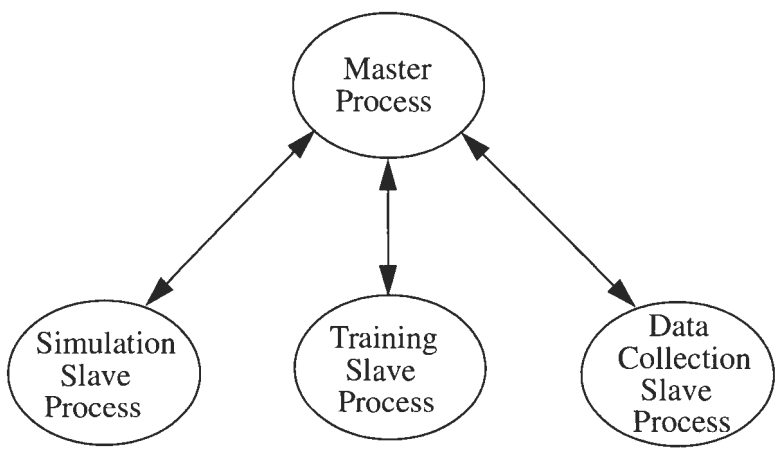

FIGURE 6.1 Multithreaded simulator.

Once the training slave process finished training the neural network, the master process would give the simulation slave process the new set of weights. The data collection slave process would continue collecting data until a suitably large training set had been collected.

6.2.2.2 Behavior Changes

There are three issues regarding the forward pilot data returned by INMS. First, as stated in Ch. 3, INMS does not return the true value of the output of the trunk amplifier. It needs to be ascertained what effect the SMT module and the INMS software has on the forward pilot data. It is entirely possible that the anticorrelation effects and other behavior changes observed in the lab may be artifacts of the SMT module or INMS software.

Assuming that the behaviors observed in the lab are not artifacts, then the two other issues need to be resolved. If the neural network simulator is to be eventually used for fault diagnosis, then the cause of the abnormal behavior must be found. Presently there is no correlation between what is being observed in the lab and to what work is being done on the trunk amplifiers. The cause of the anticorrelated behavior is also presently unknown.

Another key issue that needs to be addressed is how the faulty behavior relates to the subscriber. For example, it is currently unknown whether drops in the forward pilot level cause the subscriber to lose their television signal completely, add some 'snow' to the signal, or have no effect at all. Furthermore as the long term goal of Rogers Cable is to use the amplifier network for Internet access, it needs to be determined whether drops in the forward pilot destroy the data, make it noisy, or not effect it at all.

6.2.2.3 User Interface

The interface to the simulator is rather user unfriendly. The user having to specify the neural network configuration through a text file is clumsy and prone to errors. This is particularly true as there are so many restrictions on how the neural network can be configured. A menu based system for the configuring the neural network should be developed.

A graphical user interface displaying results is also required; currently matlab is used to display the results. For the real-time monitoring of the trunk amplifiers, a graphical display directly connected to the simulator is a necessity.

Bibliography

[1] Anderson, J.A., and E. Rosenfeld, *Neurocomputing: Foundations of Research*, The MIT Press, 1989.

[2] Antoniou, A., *Digital Filters: Analysis, Design, and Applications*, Second Edition, McGraw-Hill, Inc., 1993.

[3] Antoniou, A., *Introduction to Optimization Theory and Practice*, University of Victoria, 1994.

[4] Burden, R.L., and J.D. Faires, *Numerical Analysis*, Fourth Edition, PWS-KENT Publishing Company, 1989.

[5] Chen, T., and H. Chen, "Universal Approximation to Nonlinear Operators Using Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems", *IEEE Transactions of Neural Networks*, Vol. 6, No. 4, July 1995.

[6] Demuth, H. and Beals, M., *Neural Network ToolBox*, The Mathworks, 1994.

[7] Dimopoulos, N.J., "A Study of the Asymptotic Behavior of Neural Networks", *IEEE Transactions on Circuits and Systems*, Vol. 36, No.5, pp. 687-694, May 1989.

[8] Dimopoulos, N.J., *Elec. 564 Course Notes* University of Victoria, 1989.

[9] Dorocicz, J., E. Kosmatopoulos, S. Neville, and N.J. Dimopoulos, "Recurrent Neural Networks in System Modeling and Error Detection," *Second World Automation Congress*, Montpellier, France, May 27-30, 1996.

[10] Fletcher, R., *Practical Methods of Optimization*, John Wiley & Sons Ltd., 1987.

[11] Graf, H.P., E. Sackinger, and L.D. Jackal, "Recent Developments of Electronic Neural Nets in North America", *Journal of VLSI Signal Processing*, Vol. 5, pp. 19-31, 1993.

[12] Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Sciences U.S.A.*, Vol. 79, pp. 2554-2558, April, 1982.

[13] Hornik, K., M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol. 2, pp. 359-366, 1989.

[14] Hutchinson, J., C. Kock, J. Luo, and C. Mead, "Computing Motion Using Analog and Binary Resistive Networks", *IEEE Computer*, pp. 52-63, March, 1988.

[15] Jubien, C.M., *Asymptotically Stable Neural Networks for Identification of Nonlinear Dynamic Systems*, M.A.Sc. Thesis, University of Victoria, 1993.

[16] Kohonen, T., "The "Neural" Phonetic Typewriter", *IEEE Computer*, pp. 11-22, March, 1988.

[17] Kosmatopoulos, E.V., M.M. Polycarpou, M.A. Christodoulou, P.A. Ioannou, "High-Order Neural Network Structures for Identification of Dynamic Systems", *IEEE Transactions of Neural Networks*, Vol. 6, No. 2, March 1995.

[18] Lippmann, R.P., "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April, 1987.

[19] Lu, W.-S., personal communication, 1995.

[20] Luger, G.F., and W.A. Stubblefield, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Second Edition, The Benjamin / Cummings Publishing Company, Inc., 1993.

[21] Narendra, K.S., and K. Parthasarathy," Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Transactions of Neural Networks*, Vol. 1, No. 1, March 1990.

[22] Neville, S.W., *A Prototype Expert System Based Diagnostic Tool for Cable trunk Amplifier Networks*, M.A.Sc Thesis, University of Victoria, 1992.

[23] Neville, S.W., unpublished data, 1996.

[24] Wang. H., M. Brown, and C.J. Harris, "Fault Detection for a Class of Unknown Nonlinear Systems via Associative Memory Networks", accepted for *I. Mech. E.*, 21 pages.

[25] Watkins, A., personal communication, 1996.

[26] Zurada, J.M., *Introduction to Artificial Neural System*, West Publishing Company, 1992.

Appendix A

Neural Network Simulator

This appendix will give background information on the neural network simulator used in Chs. 3 and 4. This appendix will describe the text files used by the simulator the data structures used by the simulator, and the program logic.; the simulator returns results also using text files. If the debugging flag is set during compile time, the simulator will generate two more text files. The neural network simulator also generates one file for each layer in the neural network. These files contain the weights, shape constants, and decay constants for the layer, and are used internally by the simulator.

A.1 Configuration Files

The simulator gets almost all user input from four text files.

A.1.1 Simulator.cfg

The simulator.cfg file contains important parameters used for configuring the simulator. A sample simulator.cfg file is illustrated in Table A.1, and a description of these parameters are given in Table A.2.

TABLE A.1 Sample simulator.cfg File

total_samples:	1000
epochs:	2000
learning_rate:	0.2
data_file_name:	training_data.dat
sampling_interval:	0.09
num_layers:	4

TABLE A.2 Configuration parameters

Parameter	Description
total_samples	Total number of data points that the neural network will train and / or simulate on
epochs	Number of epochs to train for
learning_rate	η in Eq. (2.19)
data_file_name	File that contains the data
sampling_interval	Used in solving the differential equations
num_layer	Number of layers in the neural network

Note that the strings used to describe the parameters should not be changed, as the simulator uses these strings to load in the parameters. However, the order these parameters are listed in simulator.cfg is not important.

A.1.2 Neural_net.cfg

This file is used to configure the neural network. A sample neural_net.cfg file is shown in Table A.3.

TABLE A.3 Sample neural_net.cfg File

```
Input_Layer
num_neurons: 1
num_input_layers: 0
-------------------------
Scheduler_Layer
num_neurons: 10
num_input_layers: 2
input_class_num: 0
connection_polarity: 1
num_input_neurons: 1
input_class_num: 3
connection_polarity: 1
num_input_neurons: 1
-------------------------
State_Layer
num_neurons: 5
num_input_layers: 2
input_class_num: 0
connection_polarity: 1
num_input_neurons: 1
input_class_num: 1
connection_polarity: -1
num_input_neurons: 6
-------------------------
Output_Layer
num_neurons: 1
num_input_layers: 1
input_class_num: 2
connection_polarity: 0
num_input_neurons: 5
-------------------------
```

The dashed lines are used to separate the configuration of each layer. The position of the parameters in the file is important as this implies which layer the parameters belong to. For all layers, the first parameter is the title of the layer; this parameter is primarily for user clarity. The parameter, num_neurons, specifies how many neurons there are in the layer. Note that for layer 0, the input layer, num_neurons must be either one or two; and for the output layer, num_neurons must be one. The next parameter (num_input_layers) specifies the number of other layers that input to a layer. (Note that for clarity the term

destination layer will be used for layer receiving the inputs). Note that for layer 0, the input layer, num_input_layers must be zero. The next three parameters (input_layer_num, connection_polarity, and num_input_neurons) are used for defining the connection(s) to the destination layer. This group of three parameters must be defined for each connection to the destination layer. The parameter input_layer_num specifies the number of the layer (i.e zero to three for a four layer network) that is an input to the destination layer. The connection_polarity parameter specifies the polarity of the connection between the input layer and the destination layer; i.e. -1 means negative weights, +1 mean positive weights, and 0 means the weights can be positive or negative. Finally the num_input_neurons specified how neurons from the input layer connect to each neuron in the destination layer.

The order of the parameters in this file must be as shown, as the simulator reads the parameters in expecting them in the order shown. Also the dashed lines must not be changed, as the simulator uses these strings to load in the parameters. Although the strings used to describe the parameters can be changed, some type of string is needed for the parameters to be read in correctly. Note that although the simulator performs some error checking on this file as it is read in, the user should not rely on this to ensure that the neural network configuration is valid. The user should determine the validity of the neural network configuration before giving the file to the simulator. The neural network interconnection must also be carefully configured, as violating the conditions set out by [7], may cause the neural network to become unstable.

A.1.3 Training_dir.cfg

This file specifies the directory the configuration and data files are in as shown in Table A.4.

TABLE A.4 Sample training_dir.cfg

```
/package/steve/temp1/
/package/steve/temp2/
```

This file allows for batch processing.

A.1.4 Training data

The training data file must contain either two or three columns of data. The input data is in the first column(s); the corresponding output data is in the last column. The first few lines of such a file is illustrated in Table A.5.

TABLE A.5 Sample training data file

5.3693694e-01	4.9377593e-01
5.4414414e-01	4.8132780e-01
5.3153153e-01	4.8962656e-01
5.4054054e-01	4.8547718e-01
5.5135135e-01	4.9377593e-01
:	:

This data is used for both in training and simulating the neural network. It is important to note that the neural network used by the simulator can only handle input / output values that are in the range of zero to one. Thus any training data must be normalized to be in this range. Ideally the data should be normalized so that the average is 0.5.

A.2 Debugging Files

If the debugging flag is set during compile time, two text files to help the user debug the simulator are written out: connections.dat and state.dat. Connections.dat shows how the neural network is interconnected; a portion of a sample file is illustrated in Table A.6.

TABLE A.6 Sample connections.dat file

```
Layer 0: Input_Layer
  Neuron 0
Layer 1: Scheduler_Layer
  Neuron 0
    Input is Output_Layer, neuron 0 (Excitatory)
  Neuron 1
    Input is Output_Layer, neuron 0 (Excitatory)
      :
  Neuron 9
    Input is Output_Layer, neuron 0 (Excitatory)
Layer 2: State_Layer
  Neuron 0
    Input is Input_Layer, neuron 0 (Excitatory)
    Input is Scheduler_Layer, neuron 0 (Inhibitory)
    Input is Scheduler_Layer, neuron 1 (Inhibitory)
    Input is Scheduler_Layer, neuron 2 (Inhibitory)
    Input is Scheduler_Layer, neuron 3 (Inhibitory)
    Input is Scheduler_Layer, neuron 4 (Inhibitory)
    Input is Scheduler_Layer, neuron 5 (Inhibitory)
  Neuron 1
    Input is Input_Layer, neuron 0 (Excitatory)
    Input is Scheduler_Layer, neuron 1 (Inhibitory)
    Input is Scheduler_Layer, neuron 2 (Inhibitory)
    Input is Scheduler_Layer, neuron 3 (Inhibitory)
    Input is Scheduler_Layer, neuron 4 (Inhibitory)
    Input is Scheduler_Layer, neuron 5 (Inhibitory)
    Input is Scheduler_Layer, neuron 6 (Inhibitory)
  Neuron 4
    Input is Input_Layer, neuron 0 (Excitatory)
    Input is Scheduler_Layer, neuron 4 (Inhibitory)
    Input is Scheduler_Layer, neuron 5 (Inhibitory)
    Input is Scheduler_Layer, neuron 6 (Inhibitory)
    Input is Scheduler_Layer, neuron 7 (Inhibitory)
    Input is Scheduler_Layer, neuron 8 (Inhibitory)
    Input is Scheduler_Layer, neuron 9 (Inhibitory)
Layer 3: Output_Layer
  Neuron 0
    Input is State_Layer, neuron 0 (Bipolar)
    Input is State_Layer, neuron 1 (Bipolar)
    Input is State_Layer, neuron 2 (Bipolar)
    Input is State_Layer, neuron 3 (Bipolar)
    Input is State_Layer, neuron 4 (Bipolar)
```

The state.dat file shows what the weights, sigmoid shape constants, and time membrane constants are for each layer at the time the function was called. A sample file state.dat is shown in Table A.7.

TABLE A.7 Sample state.dat file

```
layer: 0 neuron: 0
 sigmoid_shape_const: 1.000000 time_membrane_const: 3.778802

layer: 1 neuron: 0
 sigmoid_shape_const: 0.068182 time_membrane_const: 0.075000
 weight: 1.000000

layer: 1 neuron: 1
 sigmoid_shape_const: 0.068182 time_membrane_const: 0.169444
 weight: 1.000000
  :
layer: 1 neuron: 9
 sigmoid_shape_const: 0.068182 time_membrane_const: 0.925000
 weight: 1.000000

layer: 2 neuron: 0
 sigmoid_shape_const: 1.000000 time_membrane_const: 0.837733
 weight: 0.406903
 weight: -0.534806
 weight: -0.345653
 weight: -0.208686
 weight: -0.749504
 weight: -0.596179
 weight: -0.162358

     :
```

A.3 Output Files

There are three types of files written by the neural network simulator: training and simulation result files, debugging files, and internal use files.

A.3.1 Results Files

For each epoch the following value is calculated using

$$s = \sum_{i=0}^{n} (o_i - o_{di})^2 \qquad \text{(A.1)}$$

where s is the sum of the square of the error of each sample, n is the number of samples, o_i is the neural network output for sample i, and o_{di} is the target output for sample i.This value is written to the file error_out.dat for each epoch (i.e. 500 epochs mean 500 file entries). By looking at this file in matlab, the user can see how well the neural network trained. An example is shown in Fig. A.1.

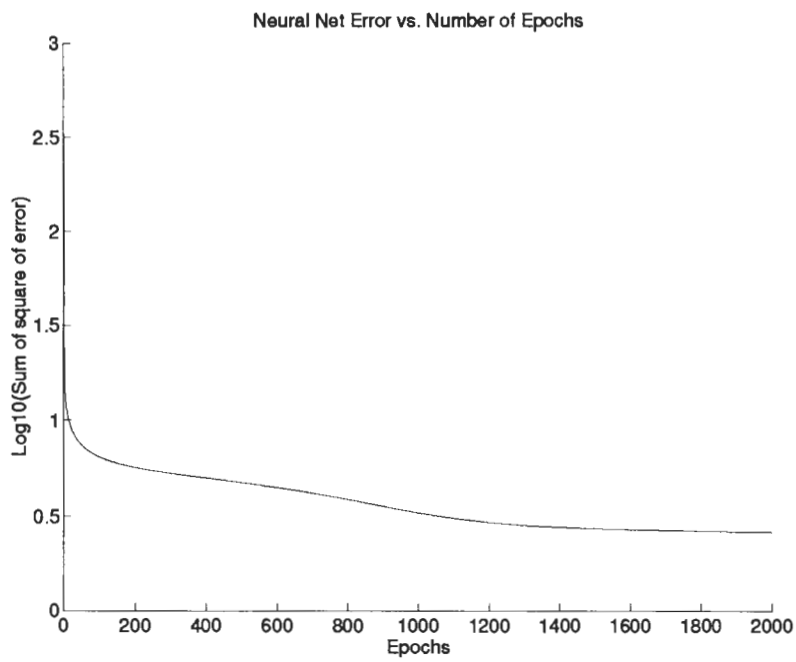

FIGURE A.1 Example training error.

After the neural network is trained, simulations can be run. In the simulation the input data (from the training data file) is applied to the input of the neural network, and the output of the neural network is calculated. Every sample the input, target output, and actual output are written to file. Again using matlab, the user can see how well the neural network tracked the system.

A.3.2 Internal use Files

Several files are generated by the simulator for internal use. A file for each layer is generated; this file contains the weights, sigmoid shape constants, and time membrane constants for that layer. These files are called l<layer_number>_state.dat. An example of this file is shown in Table A.8.

TABLE A.8 Sample l3_state.dat file

s1.00000000000000000
t2.98700000000000010
w-2.93694708491426270
w0.65330973235267187
w10.76908547348520300
w-0.82927945799127167
w3.59203948901987150

These files are used to restore the state of the neural network.

A.4 Program Logic

The neural network simulator consists of six.c files and one.h file as shown in Table A.9.

TABLE A.9 Program Files

File	Description
initialize.c	functions for initialization, and configuration the simulator and the recurrent neural network
train.c	functions for training the recurrent neural network
simulate.c	functions for simulating the recurrent neural network
main.c	main program
common.c	functions common to more than one .c file
debug.c	function used for debugging
common.h	data structure declarations and defines

The neural network simulator has three phases. In the first phase the neural network simulator reads required information from data files (described in Section 4.1) and configures the internal data structures. In phase two, the neural network simulator trains the weights (sigmoid shape constants, and time membrane constants if the correct flags are set at com-

pile time) of the neural network, so that the neural network will track the training data. Finally in phase three, the neural network simulator simulates the neural network by applying the input data and recording the actual output and target output and a time index.

A.4.1 Phase One: Setup and Initialization

The calling structure and functions called for this phase are shown in Fig. A.2. (All these functions are found in initialize.c).

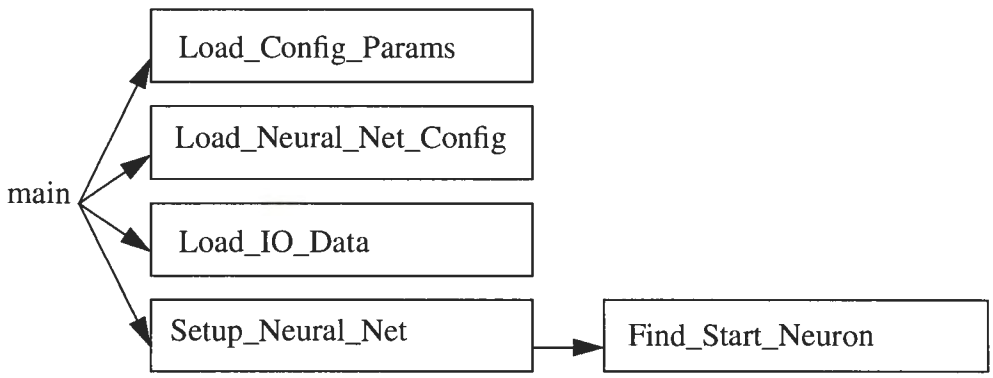

FIGURE A.2 Phase one calling structure.

Load_Config_Params is called first. This function reads the file simulator.cfg and initializes the data structure config_params. This function performs some simple error checking on the inputs in the file simulator.cfg. This function also makes sure that all the parameters are read in. Load_Config_params returns OK if it could read simulator.cfg correctly and all the parameters are valid, otherwise the function returns ERROR.

Load_Neural_Net_Config is called next to read the file neural_net.cfg. Information from this file is used to configure the data structure layer_array and the intermediate data structure connection_struct. Connection_struct is used to store how the neural network is to be interconnected. This function performs some simple error checking on the inputs in the file neural_net.cfg. This function returns OK if the data was read in correctly, otherwise the function returns ERROR.

Load_IO_Data is called next to read the training data file. The input data is stored in an array of type double for single input systems; for two input systems, the data is stored in two by n matrix of type double. The output data is stored in a separate array of type double. Both the input and output data structures are dynamically allocated (using calloc) based on the num_samples parameter. This function returns OK if the data was read in correctly, otherwise the function returns ERROR.

Setup_Neural_Net is called last. Setup_Neural_Net uses connection_struct to configure layer_array. The weights and poles are initially set to random values (except in the scheduler layer, where weights are one). For every layer except the scheduler layer, the sigmas are set to one. In the scheduler layer the sigmas are set width of the notch divided by 2.2. For every layer except the schedule layer and the output layer, the decay constants are set to random values. For the output layer the decay constant is set to three. In the scheduler layer the decay constant is the midpoint of the hump. The neural network interconnections are initialized using connection_struct. The function Find_Start_Point is used to determine how to connect the neurons from the input layer to the neurons in the destination layer.

A.4.2 Phase Two: Training

Once the neural network is setup, the training takes place. Although the neural network simulator is capable of training the weights, sigmas, and decay constants, by default the neural network simulator usually only trains the weights. The calling structure for this phase in shown in Fig. A.3. (Note that the functions marked with * are found in common.c; all other functions are found in train.c).

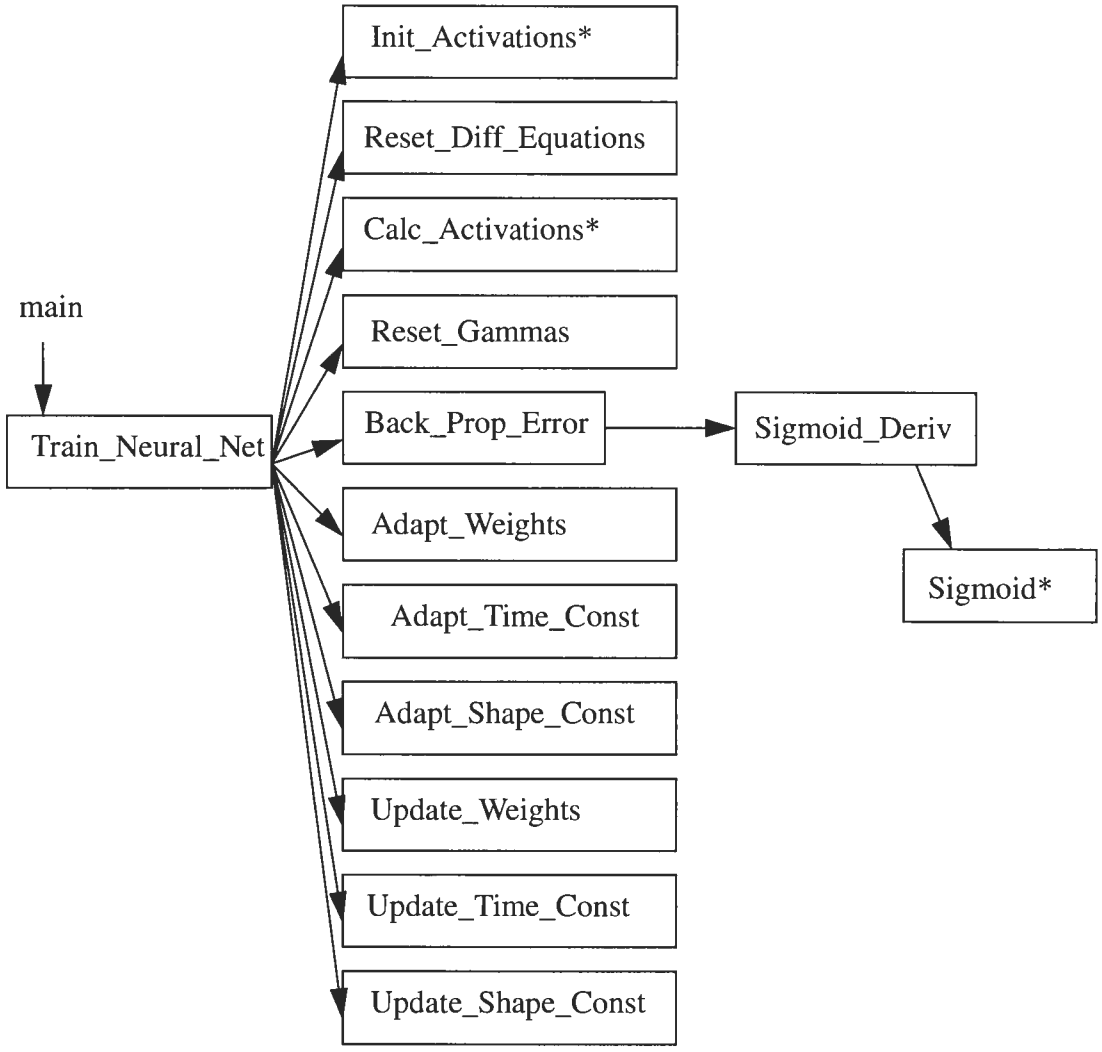

FIGURE A.3 Phase two calling structure.

The function Train_Neural_Net loops for a user specified number of epochs. In each epoch the dino program goes through all the sample data. At the beginning of each epoch all the neuron activations, except for the output neuron activation, are reset to zero by Reset_Activations. Reset_Activations sets the activation of the output neuron to the value given in simulator.cfg. Reset_Diff_Equations resets all the variables temp_delta, and xi in the structure synapse_struct to zero. (Note that these two variables are used for the numerical integration). Train_Neural_Net then goes through each sample. The input data for each sample is applied to the input(s) of the neural network. The activations are then cal-

culated for each layer (i.e. the input is propagated through the network) using Calc_Activations. Calc_Activations uses Euler's method to solve the Eq. (2.11) The output layer's activation (i.e. the actual neural network output) is compared to the target output using Eq. (2.24). This error is then back propagated through to the state layer(s) by calling the function Back_Prop_Error. Back_Prop_Error and Adapt_Weights use Euler's method to solve the differential equations required to find the weight change. The sigmas and decay_const are adjusted by calling Adapt_Shape_Const, and Adapt_Decay_Const respectively. Adapt_Shape_Const uses Euler's method to solve the differential equation, while Adapt_Decay_Const uses a closed form solution. The weights, shape_const, and decay_const are adjusted using the formulas defined in Section 2.2. The weights, sigmas, and poles are then updated (using the functions Update_Weights, Update_Sigmas, and Update_Decay_Consst respectively), and Train_Neural_Net goes to the next sample. At the end of each epoch, the sum of the square of the error (see equation 4.1) is written to file (see Section 4.2.1). Train_Neural_Net returns OK if it could write the error to file correctly, otherwise it returns ERROR.

A.4.3 Phase Three: Simulation

The calling structure for the final phase is shown in Fig. A.4. (Note that the functions marked with * are found in common.c; Simulate_Neural_Net is found in simulate.c).

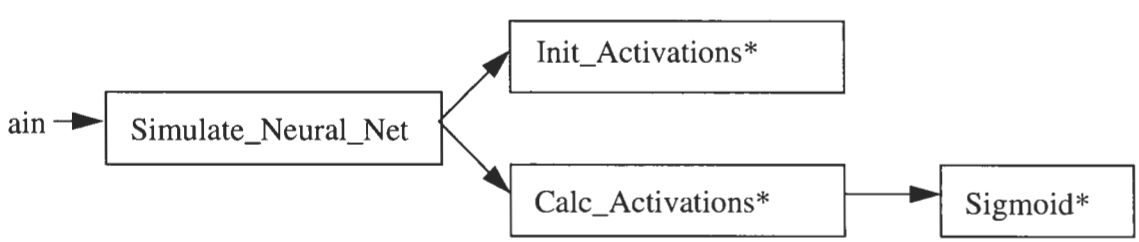

FIGURE A.4 Phase three calling structure.

Simulation is simple. The neuron activations are first reset by Reset_Activations as described in Section 5.2. The neural network is assumed trained, so the input data is applied to the input(s) of the neural network. The function Calc_Activation calculates the output of the neural network and the time, the target outputs, and the actual outputs are

recorded. The results are written to data file that can be viewed in matlab. This function returns OK if the simulation results could be written to file, otherwise the function returns ERROR.

The data structures used by the simulator (declared in common.h) are shown in the next section.

A.4.4 Data Structures

The data structure struct synapse_struct, illustrated in Table A.10, defines and stores the connections between the neurons.

TABLE A.10 Data Structure struct synapse_struct

Data type	Name	Description
double	prev_weight_change	Used for momentum term to store previous weight change
double	temp_delta	Used by function Back_Prop_Error for numerical integration
double	xi	Used by function Back_Prop_Error for numerical integration
double	weight	Interconnection weight
double	weight_change	Weight change calculated by function Back_Prop_Error
int	polarity	Interconnection polarity (-1, 0, 1)
struct neuron	*presynaptic	Pointer to input neuron

The data structure struct neuron, illustrated in Table A.11, defines the neuron. Many of the parameters are used as temporary variables in calculations.

TABLE A.11 Data Structure struct neuron

Data type	Name	Description
double	activation	Neuron activation or output (i.e. membrane potential)
double	delta	Used by function Back_Prop_Error for numerical integration
double	neuron_decay_const	Pole in activation equation

TABLE A.11 Data Structure struct neuron

Data type	Name	Description
double	new_activation	Temporary variable used for calculating new activation
double	beta	Used for adjusting the decay constant
double	sigma	Controls the shape of the sigmoid
double	decay_const_change	Used for adjusting the pole
double	shape_const_change	Used for adjusting the sigma
double	total_input	The total input to the neuron (i.e. potential at the soma)
struct synapse_struct	synapse	Connections from other neurons

The data structure struct layer_struct, illustrated in Table A.12, defines and stores a layer in the neural network. An array of layer_struct is used to define and store the neural network.

TABLE A.12 Data Structure struct layer_struct

Data type	Name	Description
char	layer_name[40]	Layer name
unsigned int	neuron_size	Size in bytes of the neurons in this layer
unsigned int	num_neurons	Number of neurons in this layer
unsigned int	num_input_layers	Number of layers that input to this layer
void	*data	The neurons and their connections are stored here (i.e pointers to neuron

The data structure struct inputs_to_layer_struct, illustrated in Table A.13, defines and stores a connection to a layer.

TABLE A.13 Data Structure struct inputs_to_layer_struct

Data type	Name	Description
unsigned int	input_layer_num	Number of the input layer
unsigned int	num_input_neurons	Number of input neurons
int	connection polarity	Polarity of the connection (-1, 0, 1)

The data structure struct connections_struct, illustrated in Table A.14, is an array of input_to_layer_struct. This array is used as temporary storage of layer interconnections. Note that the maximum number of layers that can input to another layer is ten.

TABLE A.14 Data Structure struct connections_struct

Data type	Name	Description
struct inputs_to_layer_struct	inputs[10]	Intermediate data structure used to store layer interconnections

The data structure struct config_params, illustrated in Table A.15, stores the configuration parameters described in Section A.1.

TABLE A.15 Data Structure struct config_params

Data type	Name	Description
char	training_data_file_name[80]	File that contains the training data
double	neta	Learning rate
double	output_decay_const	Initial value for the output decay constant
double	output_layer_init_cond	Output layer initial conditions
double	sampling_interval	Used for numerical integration
unsigned int	num_layers	Total number of layers
unsigned int	num_samples	Total number of samples
unsigned int	num_epochs	Total number of training epochs

Common.h also contains important defines; these are shown in Table 5.8.

Appendix B

Simulation Results

The full results of the experiments in Sec. 4.3.2 are given in Tables B.1 and B.2. Training 1 results correspond to the first experiment that used the first 1000 data points from October to train the neural network, while Training 2 results correspond to the second experiment that used the first 1000 data points from October and first 1000 data points from November. The temperature range given in the column 'Amplifier behavior' gives the temperature range of the amplifier for the entire two months. The temperature range given in the column 'Training 1 results' gives the temperature range for the first 1000 data points in October, while the temperature range given in the column 'Training 2 results' gives the temperature range for the first 1000 data points in November.

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
112	Correlated Full temp. 22.9- 55.3	Learned correlated behavior, but extend training temp. range. Training temp. 39.6- 49.1 RMS error 0.3119 Max. abs. diff. 1.4424 Min. abs. diff. 0 Ave. abs. diff. 0.2392 Std. abs. diff. 0.2001	Learned correlated behavior Training temp. 31- 35.3 RMS error 0.1576 Max. abs. diff. 1.4577 Min. abs. diff. 1.000e-5 Ave. abs. diff. 0.1199 Std. abs. diff. 0.1022
113	Correlated, with level change at the end of data set Full temp. 17.6- 65.3	Learned correlated behavior, but extend training temp. range. Training temp. 33.4- 44.8 RMS error 0.4461 Max. abs. diff. 1.5622 Min. abs. diff. 5.000e-6 Ave. abs. diff. 0.3321 Std. abs. diff. 0.2979	Learned correlated behavior Training temp. 24.3- 30.5 RMS error 0.3353 Max. abs. diff. 1.6256 Min. abs. diff. 3.5000e-5 Ave. abs. diff. 0.2074 Std. abs. diff. 0.2634

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
114	Correlated, with level change at the end of data set Full temp. 30- 69.2	Learned correlated behavior, but extend training temp. range. Training temp. 46.2- 57.2 RMS error 0.4326 Max. abs. diff. 1.6688 Min. abs. diff. 0 Ave. abs. diff. 0.3168 Std. abs. diff. 0.2947	Learned correlated behavior Training temp. 37.7- 44.3 RMS error 0.2709 Max. abs. diff. 1.7265 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.1766 Std. abs. diff. 0.2054
116	Anticorrelated, with several level changes Full temp. 24.8- 62.5	Learned anticorrelated behavior. Training temp. 37.7- 49.1 RMS error 0.9191 Max. abs. diff. 12.9308 Min. abs. diff. 3.0000e-5 Ave. abs. diff. 0.5922 Std. abs. diff. 0.7029	Learned correlated behavior Training temp. 30- 37.7 RMS error 0.6648 Max. abs. diff. 12.7608 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.4954 Std. abs. diff. 0.4434
117	Mix of correlated and anticorrelated, with large level change Full temp. 30.5- 70.6	Learned anticorrelated behavior. Training temp. 46.7- 58.7 RMS error 23.1417 Max. abs. diff. 34.8212 Min. abs. diff. 0 Ave. abs. diff. 16.5605 Std. abs. diff. 16.1646	Did not learn Training temp. 35.3- 45.8 RMS error 38.2166 Max. abs. diff. 38.2166 Min. abs. diff. 3.6000e-4 Ave. abs. diff. 17.6766 Std. abs. diff. 17.4941
118	Correlated, with large level change at the end of data set Full temp. 25.7- 66.8	Learned correlated behavior, but extend training temp. range. Training temp. 43.9- 55.8 RMS error 0.3013 Max. abs. diff. 7.3499 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.1878 Std. abs. diff. 0.2356	Learned correlated behavior, but extend training temp. range Training temp. 34.3- 38.6 RMS error 0.4368 Max. abs. diff. 7.3239 Min. abs. diff. 5.0000e-5 Ave. abs. diff. 0.2700 Std. abs. diff. 0.3434

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
119	Mix of correlated and anticorrelated behavior Full temp. 21- 60.1	Learned anticorrelated behavior. Training temp. 38.1- 50.1 RMS error 0.3637 Max. abs. diff. 9.0259 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.2147 Std. abs. diff. 0.2936	Learned correlated behavior, but extend training temp. range Training temp. 28.6- 35.7 RMS error 5.1989 Max. abs. diff. 18.5912 Min. abs. diff. 0 Ave. abs. diff. 1.7372 Std. abs. diff. 4.9002
121	Correlated, with level change at the end of data set Full temp. 29.5- 69.2	Learned correlated behavior, but extend training temp. range. Training temp. 48.2- 57.7 RMS error 3.4969 Max. abs. diff. 13.7408 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 1.3649 Std. abs. diff. 3.2196	Learned correlated behavior Training temp. 36.2- 46.2 RMS error 3.4557 Max. abs. diff. 13.7134 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 1.1899 Std. abs. diff. 3.2445
123	Mix of correlated and anticorrelated with two level changes at end of data set Full temp. 14.7- 60.6	Learned correlated behavior, but extend training temp. range. Training temp. 35.7- 49.1 RMS error 1.5904 Max. abs. diff. 9.5894 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.8678 Std. abs. diff. 1.3328	Learned correlated behavior, but extend training temp. range Training temp. 23.8- 29.1 RMS error 2.0662 Max. abs. diff. 9.6206 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.9589 Std. abs. diff. 1.8303
124	Correlated, with level change at the end of data set Full temp. 29.1- 74.4	Learned correlated behavior, but extend training temp. range. Training temp. 49.1- 62.5 RMS error 0.6790 Max. abs. diff. 5.6733 Min. abs. diff. 3.0000e-5 Ave. abs. diff. 0.3688 Std. abs. diff. 0.5701	Learned correlated behavior Training temp. 37.7- 43.4 RMS error 0.6825 Max. abs. diff. 5.7165 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.3283 Std. abs. diff. 0.5984

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
126	Correlated Full temp. 19- 60.1	Learned correlated behavior, but extend training temp. range. Training temp. 37.7- 50.1 RMS error 0.4252 Max. abs. diff. 1.9806 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.3242 Std. abs. diff. 0.2751	Learned correlated behavior Training temp. 26.2- 34.3 RMS error 0.2272 Max. abs. diff. 2.0043 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1456 Std. abs. diff. 0.1744
127	Correlated Full temp. 21- 61	Learned correlated behavior, but extend training temp. range. Training temp. 39.6- 51.5 RMS error 0.3529 Max. abs. diff. 2.7877 Min. abs. diff. 0 Ave. abs. diff. 0.2740 Std. abs. diff.0.2224	Learned correlated behavior Training temp. 29.5- 34.3 RMS error 0.1863 Max. abs. diff. 2.8221 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1545 Std. abs. diff. 0.1041
128	Correlated Full temp. 29.1- 74.4	Learned correlated behavior, but extend training temp. range. Training temp. 39.1- 49.6 RMS error 0.3223 Max. abs. diff. 2.8337 Min. abs. diff. 5.0000e Ave. abs. diff. 0.2494 Std. abs. diff. 0.2042	Learned correlated behavior Training temp. 28.1- 34.8 RMS error 0.1659 Max. abs. diff. 2.8799 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1228 Std. abs. diff. 0.1116
129	Correlated Full temp. 20- 64.4	Learned correlated behavior, but extend training temp. range. Training temp. 40.5- 51 RMS error 0.4442 Max. abs. diff. 7.9455 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.3401 Std. abs. diff. 0.2858	Learned correlated behavior Training temp. 28.6- 35.3 RMS error 0.1885 Max. abs. diff. 7.7785 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.1398 Std. abs. diff. 0.1264

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
131	Correlated Full temp. 27.6- 63.3	Learned correlated behavior, but extend training temp. range. Training temp. 46.7- 59.6 RMS error 0.8628 Max. abs. diff. 5.8864 Min. abs. diff. 0 Ave. abs. diff. 0.6541 Std. abs. diff. 0.5627	Learned correlated behavior, but extend training temp. range Training temp. 36.2- 42.4 RMS error 0.2527 Max. abs. diff. 5.9084 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.1503 Std. abs. diff. 0.2032
132	Correlated Full temp. 32.9- 72	Learned correlated behavior, but extend training temp. range. Training temp. 53.9- 62 RMS error 0.1719 Max. abs. diff. 3.0928 Min. abs. diff. 0 Ave. abs. diff. 0.1355 Std. abs. diff. 0.1058	Learned correlated behavior, but extend training temp. range Training temp. 40.5- 46.2 RMS error 0.3222 Max. abs. diff. 3.1228 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.1764 Std. abs. diff. 0.2696
136	Correlated Full temp. 23.3- 61	Learned correlated behavior, but extend training temp. range. Training temp. 47.2- 57.7 RMS error 0.5924 Max. abs. diff. 1.6505 Min. abs. diff. 3.0000e-5 Ave. abs. diff. 0.4830 Std. abs. diff. 0.3430	Learned correlated behavior Training temp. 32.9- 40.5 RMS error 0.2517 Max. abs. diff. 1.9378 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1994 Std. abs. diff. 0.1535
137	Correlated Full temp. 24.3- 61	Learned correlated behavior, but extend training temp. range. Training temp. 49.6- 61 RMS error 0.7601 Max. abs. diff. 6.8402 Min. abs. diff. 2.0000e-5 Ave. abs. diff. 0.5343 Std. abs. diff. 0.5406	Learned correlated behavior, but extend training temp. range Training temp. 33.4- 42.9 RMS error 0.2242 Max. abs. diff. 6.8562 Min. abs. diff. 0 Ave. abs. diff. 0.1591 Std. abs. diff. 0.1581

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
138	Correlated Full temp. 34.8- 75.4	Learned correlated behavior. Training temp. 57.2- 73 RMS error 0.2093 Max. abs. diff. 2.7230 Min. abs. diff. 6.5000e-5 Ave. abs. diff. 0.1693 Std. abs. diff. 0.1232	Learned correlated behavior Training temp. 47.7- 51.5 RMS error 0.3049 Max. abs. diff. 2.8299 Min. abs. diff. 5.5000e-5 Ave. abs. diff. 0.2522 Std. abs. diff. 0.1714
139	Correlated Full temp. 22.9- 68.7	Learned correlated behavior. Training temp. 44.8- 68.7 RMS error 0.4753 Max. abs. diff. 2.8625 Min. abs. diff. 9.5000e-5 Ave. abs. diff. 0.3909 Std. abs. diff. 0.2703	Learned correlated behavior Training temp. 31- 39.6 RMS error 0.4010 Max. abs. diff. 2.8311 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.2497 Std. abs. diff. 0.3138
141	Correlated Full temp. 36.2- 76.3	Learned correlated behavior. Training temp. 55.8- 64.9 RMS error 0.1924 Max. abs. diff. 2.4577 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1405 Std. abs. diff. 0.1315	Learned correlated behavior Training temp. 44.3- 51 RMS error 0.2892 Max. abs. diff. 2.6375 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.2284 Std. abs. diff. 0.1773
142	Correlated Full temp. 20.5- 59.1	Learned correlated behavior, but extend training temp. range. Training temp. 41.5- 50.1 RMS error 0.3282 Max. abs. diff. 2.5962 Min. abs. diff. 0 Ave. abs. diff. 0.2679 Std. abs. diff. 0.1895	Learned correlated behavior, but extend training temp. range Training temp. 29.1- 34.3 RMS error 0.2853 Max. abs. diff. 2.6619 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.1959 Std. abs. diff. 0.2074
143	Correlated Full temp. 21.9- 64.4	Learned correlated behavior, but extend training temp. range. Training temp. 44.3- 52.5 RMS error 0.4470 Max. abs. diff. 1.4952 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.3613 Std. abs. diff. 0.2632	Learned correlated behavior Training temp. 31.5- 38.6 RMS error 0.2268 Max. abs. diff. 1.5641 Min. abs. diff. 2.0000e-5 Ave. abs. diff. 0.1716 Std. abs. diff. 0.1483

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
144	Correlated Full temp. 26.2- 68.7	Learned correlated behavior. Training temp. 47.7- 58.7 RMS error 0.2056 Max. abs. diff. 10.0754 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.1520 Std. abs. diff. 0.1386	Learned correlated behavior, but extend training temp. range Training temp. 39.1- 45.3 RMS error 0.4319 Max. abs. diff. 10.0702 Min. abs. diff. 3.0000e-5 Ave. abs. diff. 0.2694 Std. abs. diff. 0.3376
145	Correlated Full temp. 40- 74.9	Learned correlated behavior. Training temp. 53.4- 68.7 RMS error 0.2274 Max. abs. diff. 4.5675 Min. abs. diff. 4.5000e-5 Ave. abs. diff. 0.1817 Std. abs. diff. 0.1368	Learned correlated behavior Training temp. 47.2- 52.9 RMS error 0.1831 Max. abs. diff. 4.6762 Min. abs. diff. 0 Ave. abs. diff. 0.1441 Std. abs. diff. 0.1130
200	Correlated, with level change at the end of the data set, anticorrelated after level change Full temp. 17.6- 57.7	Learned correlated behavior, but extend training temp. range. Training temp. 34.8- 45.3 RMS error 0.8349 Max. abs. diff. 11.9850 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.5647 Std. abs. diff. 0.6149	Learned correlated behavior, but extend training temp. range Training temp. 24.3- 33.4 RMS error 3.4763 Max. abs. diff. 16.3212 Min. abs. diff. 0 Ave. abs. diff. 1.1800 Std. abs. diff. 3.2699
202	Correlated, with two level changes in amplifier behavior Full temp. 27.6- 80.1	Learned correlated behavior. Training temp. 68.2- 77.8 RMS error 0.7331 Max. abs. diff. 8.9532 Min. abs. diff. 0 Ave. abs. diff. 0.5472 Std. abs. diff.0.4879	Learned correlated behavior Training temp. 54.8- 60.6 RMS error 0.7974 Max. abs. diff. 7.9097 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.6707 Std. abs. diff. 0.4312
203	Correlated, with two level changes in amplifier behavior Full temp. 29.5- 69.2	Learned correlated behavior. Training temp. 52- 62 RMS error 0.8499 Max. abs. diff. 3.5130 Min. abs. diff. 5.5000e-5 Ave. abs. diff. 0.5490 Std. abs. diff. 0.6488	Learned correlated behavior Training temp. 40.5- 45.8 RMS error 0.8007 Max. abs. diff. 3.6379 Min. abs. diff. 4.5000e-5 Ave. abs. diff. 0.5095 Std. abs. diff. 0.6177

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
204	mix of correlated and anticorrelated, two behavior changes Full temp. 41- 80.6	Learned correlated behavior. Training temp. 63.9- 75.4 RMS error 0.7883 Max. abs. diff. 11.2506 Min. abs. diff. 3.8000e-4 Ave. abs. diff. 0.6814 Std. abs. diff. 0.3965	Learned correlated behavior Training temp. 52.9- 58.2 RMS error 0.3473 Max. abs. diff. 11.6774 Min. abs. diff. 8.3000e-4 Ave. abs. diff. 0.2248 Std. abs. diff. 0.2647
207	Correlated Full temp. 33.4- 78.2	Learned correlated behavior, but extend training temp. range. Training temp. 33.4- 68.2 RMS error 0.4030 Max. abs. diff. 6.2469 Min. abs. diff. 9.5000e-5 Ave. abs. diff. 0.3347 Std. abs. diff. 0.2244	Learned correlated behavior, but extend training temp. range Training temp. 43.4- 50.5 RMS error 0.4566 Max. abs. diff. 6.1770 Min. abs. diff. 4.5000e-5 Ave. abs. diff. 0.3370 Std. abs. diff. 0.3081
208	Correlated, pilot level low Full temp. 35.4- 83.5	Learned correlated behavior, but extend training temp. range. Training temp. 63.9- 75.4 RMS error 0.2887 Max. abs. diff. 5.6576 Min. abs. diff. 7.5000e-5 Ave. abs. diff. 0.2243 Std. abs. diff. 0.1818	Learned correlated behavior, but extend training temp. range Training temp. 53.4- 58.2 RMS error 5.1281 Max. abs. diff. 13.8859 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 2.3824 Std. abs. diff. 4.5412
210	Correlated, with two level changes in amplifier behavior Full temp. 40.5 - 80.1	Learned correlated behavior. Training temp. 61.5- 73.5 RMS error 0.6259 Max. abs. diff. 13.6200 Min. abs. diff. 1.3000e-4 Ave. abs. diff. 0.5308 Std. abs. diff. 0.3317	Learned correlated behavior Training temp. 52.9- 57.7 RMS error 0.4017 Max. abs. diff. 13.9329 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.2895 Std. abs. diff. 0.2784
211	Correlated Full temp. 31.5- 68.7	Learned correlated behavior. Training temp. 54.4- 62.5 RMS error 0.3161 Max. abs. diff. 7.1550 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1566 Std. abs. diff. 0.2746	Learned correlated behavior Training temp. 44.3- 49.1 RMS error 0.3675 Max. abs. diff. 7.1345 Min. abs. diff. 0 Ave. abs. diff. 0.1562 Std. abs. diff. 0.3327

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
221	Correlated Full temp. 31.9- 76.3	Learned correlated behavior, but extend training temp. range. Training temp. 48.2- 59.6 RMS error 2.5236 Max. abs. diff. 11.4200 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.8808 Std. abs. diff. 2.3649	Learned correlated behavior Training temp. 39.1- 46.7 RMS error 0.1561 Max. abs. diff. 4.2950 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1250 Std. abs. diff. 0.0936
222	Correlated, with level change at end of data set Full temp. 18.6 -55.8	Learned correlated behavior, but extend training temp. range. Training temp. 37.2- 48.6 RMS error 1.3570 Max. abs. diff. 7.6599 Min. abs. diff. 5.0000e-5 Ave. abs. diff. 0.6580 Std. abs. diff. 1.1868	Learned correlated behavior Training temp. 26.2- 31.9 RMS error 1.4199 Max. abs. diff. 7.7818 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.5412 Std. abs. diff. 1.3127
225	Mix of correlated and anticorrelated, several level changes Full temp. 28.1- 72	Learned anticorrelated behavior. Training temp. 38.6- 58.7 RMS error 101.8045 Max. abs. diff. 232.2475 Min. abs. diff. 0.0053 Ave. abs. diff. 96.2732 Std. abs. diff. 33.1008	Learned correlated behavior Training temp. 38.1- 46.2 RMS error 1.2715 Max. abs. diff. 61.9485 Min. abs. diff. 0 Ave. abs. diff. 0.5281 Std. abs. diff. 1.1566
232	Correlated, with level change at end of data set Full temp. 28.1- 70.1	Learned correlated behavior, but extend training temp. range. Training temp. 38.6 -58.7 RMS error 0.4087 Max. abs. diff. 6.2457 Min. abs. diff. 0 Ave. abs. diff. 0.2272 Std. abs. diff. 0.3398	Learned correlated behavior Training temp. 40- 45.8 RMS error 0.4084 Max. abs. diff. 6.2483 Min. abs. diff. 0 Ave. abs. diff. 0.2227 Std. abs. diff. 0.3423

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
233	Anticorrelated, two level changes Full temp. 39.1- 82.5	Learned correlated behavior. Training temp. 49.1- 72.5 RMS error 0.7621 Max. abs. diff. 4.7499 Min. abs. diff. 1.9000e-4 Ave. abs. diff. 0.5941 Std. abs. diff. 0.4774	Learned correlated behavior Training temp. 49.1- 57.2 RMS error 0.8118 Max. abs. diff. 5.3068 Min. abs. diff. 7.5000e-5 Ave. abs. diff. 0.4732 Std. abs. diff. 0.6596
235	Correlated, with level change at the end of the data set Full temp. 27.6- 70.6	Learned correlated behavior, but extend training temp. range. Training temp. 47.7- 58.2 RMS error 2.1305 Max. abs. diff. 9.3021 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.7749 Std. abs. diff. 1.9846	Learned correlated behavior Training temp. 34.3- 43.9 RMS error 2.1851 Max. abs. diff. 9.3360 Min. abs. diff. 3.5000e-5 Ave. abs. diff. 0.7599 Std. abs. diff. 2.0488
236	mix of correlated and anticorrelated, one level change Full temp. 20- 61.5	Learned correlated behavior, but extend training temp. range. Training temp. 37.7- 47.2 RMS error 2.2657 Max. abs. diff. 11.6784 Min. abs. diff. 2.5000e-5 Ave. abs. diff. 0.8637 Std. abs. diff. 2.0946	Learned correlated behavior, but extend training temp. range Training temp. 25.7- 35.3 RMS error 2.2782 Max. abs. diff. 11.7668 Min. abs. diff. 2.0000e-5 Ave. abs. diff. 0.7253 Std. abs. diff. 2.1597
240	mix of correlated and anticorrelated, one level change Full temp. 28.1- 69.2	Learned correlated behavior, but extend training temp. range. Training temp. 47.2- 55.8 RMS error 2.3178 Max. abs. diff. 10.3640 Min. abs. diff. 5.0000e-5 Ave. abs. diff. 0.9779 Std. abs. diff. 2.1015	Learned correlated behavior Training temp. 34.8- 44.8 RMS error 2.4230 Max. abs. diff. 10.6232 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.8554 Std. abs. diff. 2.2670

TABLE B.1 High pilot amplifier results

Amp	Amplifier behavior	Training 1 results	Training 2 results
241	Anticorrelated, two level changes Full temp. 20- 58.7	Learned anticorrelated behavior. Training temp. 40.5- 50.1 RMS error 4.0498 Max. abs. diff. 37.5713 Min. abs. diff. 9.5500e-4 Ave. abs. diff. 2.0325 Std. abs. diff. 3.5046	Did not learn Training temp. 25.7- 37.2 RMS error 1.0092e3 Max. abs. diff. 1.5681e3 Min. abs. diff. 0.0012 Ave. abs. diff. 942.0031 Std. abs. diff. 362.1729
243	Anticorrelated, several level changes, extremely unstable behavior Full temp. 31.9- 74.4	Did not learn. Training temp. 50.1- 60.6 RMS error 2.0228 Max. abs. diff. 7.2496 Min. abs. diff. 0.0015 Ave. abs. diff. 1.5078 Std. abs. diff. 1.3484	Did not learn Training temp. 39.1- 47.7 RMS error 2.3638 Max. abs. diff. 18.3136 Min. abs. diff. 2.1000e-4 Ave. abs. diff. 1.5981 Std. abs. diff. 1.7417
244	Correlated, with level change at end of data set Full temp. 30.5- 69.2	Learned correlated behavior, but extend training temp. range. Training temp. 47.2- 57.2 RMS error 0.1577 Max. abs. diff. 0.6101 Min. abs. diff. 1.7000e-4 Ave. abs. diff. 0.1235 Std. abs. diff. 0.0982	Learned correlated behavior Training temp. 36.2- 43.9 RMS error 0.3314 Max. abs. diff. 5.6821 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1705 Std. abs. diff. 0.2842

TABLE B.2 Low pilot amplifier results

Amp	Amplifier behavior	Training 1	Training 2
102	Initially correlated, with one level change in amp. behavior, anticorrelated after level change Full temp. 33.8- 71.5	Learned correlated behavior. Training temp. 50.1- 62.5 RMS error 1.3083 Max. abs. diff. 7.2202 Min. abs. diff. 0 Ave. abs. diff. 1.1969 Std. abs. diff. 0.5282	Learned anticorrelated behavior Training temp. 41.5- 47.7 RMS error 1.1461 Max. abs. diff. 12.6594 Min. abs. diff. 3.5000e-5 Ave. abs. diff. 0.4927 Std. abs. diff. 1.0348
115	Correlated Full temp. 29.1- 70.1	Learned correlated behavior, but extend training temp. range. Training temp. 42.4- 54.8 RMS error 1.9213 Max. abs. diff. 25.5208 Min. abs. diff. 3.0000e-5 Ave. abs. diff. 0.6939 Std. abs. diff. 1.7916	Learned correlated behavior Training temp. 35.3- 43.4 RMS error 0.8581 Max. abs. diff. 6.6417 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.4558 Std. abs. diff. 0.7270
120	Correlated Full temp. 23.8- 65.3	Learned correlated behavior, but extend training temp. range. Training temp. 42.4- 52 RMS error 1.9277 Max. abs. diff. 8.4734 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.7721 Std. abs. diff. 1.7663	Learned correlated behavior Training temp. 31- 40 RMS error 2.0076 Max. abs. diff. 8.5665 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.7344 Std. abs. diff. 1.8685
125	Correlated Full temp. 26.7- 65.3	Learned correlated behavior, but extend training temp. range. Training temp. 44.8- 56.3 RMS error 0.3994 Max. abs. diff. 7.0151 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.3031 Std. abs. diff. 0.2601	Learned correlated behavior Training temp. 33.4- 39.1 RMS error 0.2751 Max. abs. diff. 6.9728 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.2211 Std. abs. diff. 0.1637

TABLE B.2 Low pilot amplifier results

Amp	Amplifier behavior	Training 1	Training 2
130	Correlated Full temp. 27.6- 68.2	Learned correlated behavior, but extend training temp. range. Training temp. 46.2- 60.1 RMS error 0.1748 Max. abs. diff. 0.6463 Min. abs. diff. 3.8000e-4 Ave. abs. diff. 0.1364 Std. abs. diff. 0.1094	Learned correlated behavior Training temp. 36.2- 41 RMS error 0.2314 Max. abs. diff. 3.8228 Min. abs. diff. 2.5000e-5 Ave. abs. diff. 0.1824 Std. abs. diff. 0.1423
140	Correlated Full temp. 29.1- 74.4	Learned correlated behavior, but extend training temp. range. Training temp. 56.8- 66.3 RMS error 0.3810 Max. abs. diff. 7.7331 Min. abs. diff. 3.0000e-5 Ave. abs. diff. 0.2444 Std. abs. diff. 0.2922	Learned correlated behavior Training temp. 41.5- 50.5 RMS error 0.1960 Max. abs. diff. 7.5034 Min. abs. diff. 0 Ave. abs. diff. 0.1479 Std. abs. diff. 0.1286
201	Correlated with two level changes in amp. behavior Full temp. 34.8- 72	Learned correlated behavior. Training temp. 50.5- 61.5 RMS error 1.6224 Max. abs. diff. 8.7494 Min. abs. diff. 0 Ave. abs. diff. 1.1025 Std. abs. diff. 1.1903	Did not learn Training temp. 41.5- 52.5 RMS error 1.6881 Max. abs. diff. 17.9993 Min. abs. diff. 0 Ave. abs. diff. 1.3821 Std. abs. diff. 0.9692
206	Correlated Full temp. 42- 82.5	Learned correlated behavior, but extend training temp. range. Training temp. 64.4- 70.6 RMS error 0.6424 Max. abs. diff. 7.4837 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.3548 Std. abs. diff. 0.5356	Learned correlated behavior Training temp. 52.9- 59.6 RMS error 0.5727 Max. abs. diff. 7.4331 Min. abs. diff. 2.0000e-5 Ave. abs. diff. 0.3263 Std. abs. diff. 0.4707

TABLE B.2 Low pilot amplifier results

Amp	Amplifier behavior	Training 1	Training 2
209	Correlated Full temp. 36.7- 80.6	Learned correlated behavior, but extend training temp. range. Training temp. 59.1- 70.1 RMS error 1.7586 Max. abs. diff. 21.5766 Min. abs. diff. 0 Ave. abs. diff. 0.3927 Std. abs. diff. 1.7142	Learned correlated behavior Training temp. 45.8- 53.4 RMS error 0.2245 Max. abs. diff. 4.4775 Min. abs. diff. 5.0000e-6 Ave. abs. diff. 0.1782 Std. abs. diff. 0.1365
224	Initially correlated, with several level changes in amp. behavior, anticorrelated after last level change Full temp. 29.1- 67.7	Learned correlated behavior, but extend training temp. range. Training temp. 41.5- 56.3 RMS error 1.3236 Max. abs. diff. 5.6307 Min. abs. diff. 3.5000e-5 Ave. abs. diff. 0.7518 Std. abs. diff. 1.0894	Learned correlated behavior Training temp. 37.2- 42.4 RMS error 1.3903 Max. abs. diff. 5.6730 Min. abs. diff. 1.0000e-5 Ave. abs. diff. 0.7035 Std. abs. diff. 1.1992
231	Initially correlated, with several level changes in amp. behavior, anticorrelated after last level change Full temp. 36.2- 73.5	Learned correlated behavior, but extend training temp. range. Training temp. 57.7- 64.4 RMS error 1.6173 Max. abs. diff. 7.2078 Min. abs. diff. 1.5000e-5 Ave. abs. diff. 0.8802 Std. abs. diff. 1.3568	Learned correlated behavior, but extend training temp. range Training temp. RMS error Max. abs. diff. Min. abs. diff. Ave. abs. diff. Std. abs. diff.

Vita

Surname: Dorocicz Given Names: John Tadeusz

Place of Birth: Port Alberni, British Columbia, Canada

Education Institutions Attended:

Unviersity of Victoria	1990 to 1996
Malaspina College	1989 to 1990

Degrees Awarded:

B.Eng.	University of Victoria	1994

Honors and Awards:

University of Victoria Fellowship	1995-96

Publications:

Campbell, A., and J. Dorocicz, "Yield and Risk Analysis for the Geoduck Fisheries in two Areas of Southern British Columbia," PSARC Working Paper 192-02, PSARC Shellfish Subcommittee Meeting, 1992.

Dimopoulos, N.J., S. Neville, J. Dorocicz, and C. Jubien, "Training Asymptotically Stable Recurrent Neural Networks," *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics*, pp. 4392 - 4397, Vancouver, Canada, Oct. 22-25, 1995.

Dorocicz, J., E. Kosmatopoulos, S. Neville, and N.J. Dimopoulos, "Recurrent Neural Networks in System Modeling and Error Detection," *Second World Automation Congress*, Montpellier, France, May 27-30, 1996.

Dorocicz, J., E. Kosmatopoulos, S. Neville, and N.J. Dimopoulos, "Recurrent Neural Networks for Fault Detection," *Proceedings of the 4th IEEE Mediterranean Symposium on New Directions in Control and Automation*, pp 17-22, Chania, Greece, June 10-14, 1996.

Kosmatopoulos, E., J. Dorocicz, N.J. Dimopoulos, S. Neville, A. Watkins, and K.F. Li. "Recurrent Neural Networks in Modelling and Fault Detection of the Forward Pilot in Main Trunk Amplifiers," *Proceedings of the CCTA*, pp 21-28, Edmonton, Canada, June 2-5, 1996.

Dimopoulos, N.J., J. Dorocicz, C. Jubien, and S. Neville, "Training Asymptotically Stable Recurrent Neural Networks," accepted (Aug. 27, 1996) for *Intelligent Automation and Soft Computing*, (21 pages).

Partial Copyright License

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis:

Asymptotically Stable Recurrent Neural Networks: Theory and Application

Autho[redacted]

John Dorocicz

Mar 10, 1997