Spring 1995

Generalization Metrics for Neural Modeling Applications in System Identification

Denise M. Reeves
Old Dominion University

GENERALIZATION METRICS FOR NEURAL MODELING APPLICATIONS IN SYSTEM IDENTIFICATION

Denise M. Reeves

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
May 1995

Approved by:

Oscar R. González (Director)

Joseph L. Hibey

John W. Stoughton

ABSTRACT

In this thesis a procedure to design multilayer feedforward networks for system identification with good prediction properties is presented. Central to the design procedure is a means to characterize the prediction capabilities of various trained neural networks. Such knowledge will allow for the identification of the best network design. For system identification purposes, a "good" model is one that is good at predicting. In particular, a good model is one that produces small prediction errors when applied to a set of cross-validation data. We formulate and implement a criterion function designed to measure the size of a trained neural network's prediction error. The criterion function or generalization metric is implemented in three system identification design examples. The metric is used to determine the number of delays needed for the input signal, the number of hidden nodes; and the number of training cycles necessary to train the neural network.

ACKNOWLEDGEMENTS

I wish to thank my advisor, Dr. Oscar R. González, for his help and support. I also wish to thank Dr. Joseph L. Hibey and Dr. John W. Stoughton for serving on my thesis committee. I extend special thanks to Dr. John W. Stoughton for his wisdom and guidance. I also extend special thanks to Dr. Linda L. Vahala for her support and encouragement.

My heartfelt appreciation goes to Dr. Philip R. Wohl for giving initial voice to my creative endeavor with the words "tell me". Don Soloway and Pam Haley of NASA Langley Research Center both contributed to this project and their technical support is gratefully acknowledged.

Finally, my special thanks go to Carol A. Roby and Pamela H. Schmidt for their invaluable friendship and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Estimation of Physical Processes

The physical world in which we live is a complex interconnection of physical processes characterized by nonlinear phenomena. Attempts to construct meaningful patterns from the observed physical phenomena defines the essence of many scientific activities. Many of these patterns need to have a temporal component. In particular, the behavior of dynamical systems can be modeled by such patterns. Successful construction of dynamical system models makes it possible to predict the response of the system. Good models are important in many control design procedures.

1.2 Thesis Research Objective

The objective of this research was to develop a procedure to pick the best predictor from a set of trained neural networks. Neural networks are realizations of a nonlinear model of the physical phenomenon known to be the source of the set of input/output data used in identification. For a given set of input/output data, a set of neural networks that models the data is trained. These neural networks differ on the number of delay nodes attached to the input, the number of hidden nodes, and the number of training cycles. The prediction capabilities of the neural networks are measured with

a generalization metric introduced in this research. This metric is used to pick the network that will perform with low error on unseen data. The metric also indicates whether or not neural networks with more delays on the input or more hidden nodes or more training cycles may be useful. The design procedure is illustrated via three examples in which we identify both a first order and second order system characterized by the same cutoff frequency of one radian per second, and a nonlinear system which is descriptive of a mass-spring-damper with a stiffening spring.

1.3 Thesis Organization

Chapter Two examines the background and theory necessary to design multilayer feedforward networks for system identification applications. Chapter Three considers the topic of model validation and the accompanying criterion used to evaluate the various network structures. In Chapter Four, we present the design examples. The conclusions of the thesis and topics for further research are presented in Chapter Five.

CHAPTER TWO

NEURAL MODELING AND SYSTEM IDENTIFICATION

2.1 Introduction

In this chapter we present the foundations, structures, concepts and principles of system identification and neural modeling. A synopsis of the chapter follows.

2.2 Organization of the Chapter

Section 2.3 describes the system identification problem and procedure. In Section 2.4 we present a brief overview of the modeling of natural phenomena and the limitations associated with such modeling. The general properties and attributes of neural networks are examined in Section 2.5. We present the mathematical background which justifies the use of neural networks as mathematical models in Section 2.6. The general learning process used to create multilayer neural networks is examined in detail in Section 2.7. An overview and outline of the back-propagation training algorithm is presented in Section 2.8. The statistical nature of the learning process is explored in Section 2.9, followed by the mapping performance of neural networks in Section 2.10. Neural networks are examined from the perspective of nonparametric inference in Section 2.11. We consider the matter of temporal representation by means of feedforward architecture in Section 2.12. Section 2.13 is an overview of the design of appropriate training signals, followed by section 2.14 which contains the conclusions of the chapter.

2.3 The System Identification Problem

The construction of a model based on observations is a creative process guided by a few basic principles. All models are characterized by their attempt to weave observations together into some type of meaningful pattern. In particular, the primary objective of empirical modeling is to identify and train a model that will perform with low error on unseen data. [B1, p. 87]. A model capable of accurately predicting an observed physical phenomenon can be applied to both the prediction and control of that phenomenon. Accurate understanding and modeling of the world is accompanied by the power to transform it. Specifically, system identification addresses the problem of constructing mathematical models of dynamical systems using their observed data as a basis [L3, p. 1].

The System Identification Procedure

The construction of a mathematical model from a system's observed data involves three basic entities [L3, p. 7]:

I. A data record composed of a collection of observed (x,y) pairs containing the desired response y for each input x.

II. Selection of a set of candidate models.

III. The determination of the best model. The best model will be qualified as one that best describes the data according to some chosen criterion.

Three fundamental steps comprise the system identification procedure, the last two steps defining an iterative process. The initial step requires the collection of a set of input/output data associated with the system to be identified. The collected data will be

a uniformly sampled input signal and response of the system being identified. Collection of the data is followed by the selection of a specific model structure the data are to be fit to. An entire class of models exist within this predefined structure, the model structure being characterized by a set of parameters which are not known and must be estimated.

The unknown parameters are estimated in the third step which involves fitting the data to the pre-selected model structure. An optimization criterion defines the manner in which the data are fit to the model. Typically, the parameters of the model are chosen to minimize some cost functional. A common approach is to form the sum of observed squared errors

$$\sum_{i=1}^{N} [y_i - f(x_i)]^2 \qquad (2.1)$$

where y_i is the sampled system output, x_i is the sampled input signal and N is the total number of data samples. The model f is one of the candidate models from Step 2. It is chosen to make this sum as small as possible [G1, p. 3].

Standard optimization techniques are used to find the parameters characterizing the model f. However a different model structure may lead to a reduced cost functional. Therefore, it is important to have a criterion for ranking the different model structures and selecting the best [B1, p. 87],[M1, p. 1].

Accordingly, the fundamental task of this research is the development of appropriate performance metrics to assess the various models obtained through the process of training neural networks. In particular, we seek to identify the neural network applied

to the system identification problem that will perform with the lowest error on a set of cross-validation data.

2.4 Modeling of Nature

An ever-present veil stands between nature and our own understanding and perception of it. Our best attempts to construct mathematical models of natural phenomena can be likened to describing a four dimensional world from a three dimensional perspective. Simply put, our best models will always be constrained by some type of impasse. A significant task in any modeling problem is to properly identify the model's limitations and work within these constraints towards the construction of a model based on appropriate compromises. Based on these observations, our best attempts to model physical systems will, at best, be crude approximations of the "true system".

Basically, our models will consist of pre-selected neural network architectures whose parameters have been adjusted to obtain an optimal data fit. The resulting model structures will be limited in their representation of the systems they are designed to model and simulate. With all of the above in mind, we seek to determine an optimal model structure based on a practical objective. The best model will perform with the lowest error on unseen data. The primary research task is seen to be the development of appropriate performance metrics to identify and select this best model.

2.5 The Selected Class of Models: Neural Networks

The second step of the system identification procedure requires the selection of specific model structures for fitting the data. The model structures chosen in this thesis consist of an important neural network paradigm. The specific class of neural networks to be employed as models of dynamical systems are commonly known as multilayer feedforward networks. These networks have been successfully applied to the complex tasks of nonlinear model fitting, speech processing, system identification, control, and pattern recognition. Research applications in the above areas have become both popular and fashionable. See for example [B4, p. 193],[H1, p. 221].

Neural Networks as Adaptive Mapping Machines

In its most general form, a neural network is a mapping machine that is designed to receive and embody empirical knowledge of natural phenomena. In particular, a neural network is an adaptive mapping machine. Designed networks are implemented by means of electronic components or simulated with software on a digital computer.

A neural network derives its computing power through its distributed parallel structure which supports the network's ability to learn and generalize from stored knowledge. These two information-processing features render the neural network an extremely powerful mapping machine.

A significant mapping application is found in the mathematical modeling of complex nonlinear phenomena. Such phenomena are representative of a wide class of processes whose identification is still a subject of research. This research employs the

powerful mapping capabilities associated with an important class of neural networks known as multilayer feedforward neural networks.

Feedforward Neural Network Architectures

A neural network is a parallel, interconnected network of elementary units called neurons. A neuron is a mathematical function which serves to transform an additive set of weighted inputs into a single output. The neurons used to construct feedforward networks are distinguished by two primary mathematical functions. Linear transformations characterize the input layer, while nonlinear transformations characterize the hidden layers and sometimes the output layer. A network is composed of interconnected layers of these neurons. The weighted output of each neuron in an individual layer is broadcast to each of the neurons in the next layer. Different layers may contain different neurons, however individual layers are homogeneous. The simplest architecture contains an input layer, a single hidden layer and an output layer. In general, feedforward networks are composed of layers of neurons between the input and output layers and contain at least one hidden layer. The general structure of a multilayered feedforward neural network is illustrated in Figure 2.1.

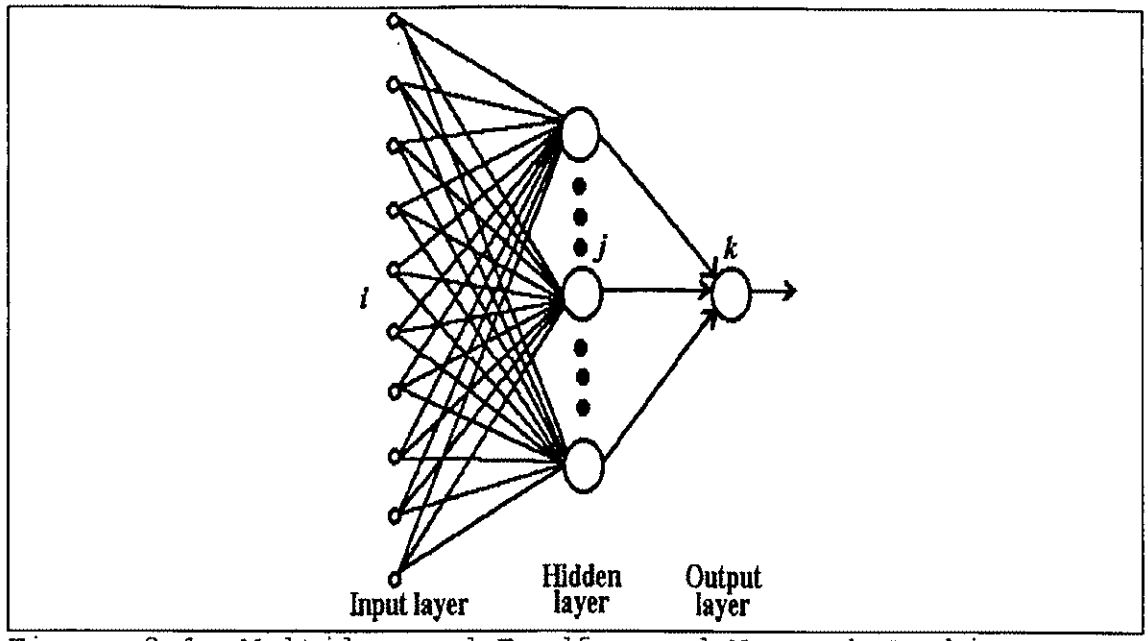


Figure 2.1 Multilayered Feedforward Network Architecture

Input Layer

A network's input layer is composed of linear nodes whose function is to broadcast the training instances presented to the network for processing. Individual training instances are weighted and summed prior to transformation by each of the hidden layer nonlinearities.

Hidden Layer

Hidden layer nodes are characterized by nonlinear functions and are the powerhouse of the network. The sigmoid function is the most common type of nonlinear function used to construct feedforward networks. Sigmoid functions are monotonically increasing functions that exhibit smoothness and asymptotic properties. The smoothness of the function allows for its differentiability, an important feature in the implementation of the gradient descent algorithm performed by the back-propagation algorithm. This

research employed the hyperbolic tangent function as the network's source of hidden layer nonlinearities. The hyperbolic tangent is mathematically defined to be:

$$\tanh\left(\frac{v}{2}\right) = \frac{1 - \exp(-v)}{1 + \exp(-v)}, \quad -\infty < v < \infty \qquad (2.2)$$

where v is the sum of weighted inputs formally defined to be the net internal activity level of the hidden neuron. This particular sigmoid maps input values into an output ranging between -1 and 1. The hyperbolic tangent is also an odd function, distinguished by the mathematical property

$$\varphi(-v) = -\varphi(v) . \qquad (2.3)$$

It has been observed that a multilayer neural network trained with the back-propagation training algorithm may learn faster (fewer training iterations are required) when the sigmoidal activation function is asymmetric [H1, p. 160]. Other research indicates that the choice of the activation function may not be as important. Figure 2.2 illustrates a hidden unit. The most significant feature of hidden units is their ability to expand the mapping capabilities of the network.

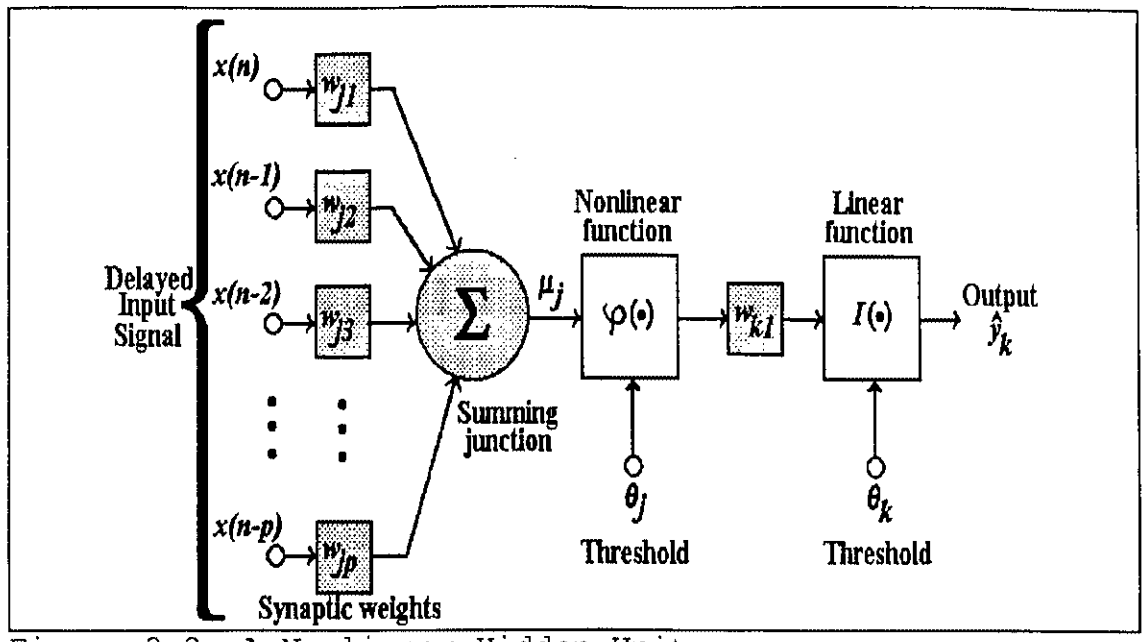


Figure 2.2 A Nonlinear Hidden Unit

Output Layer

The class of networks used to solve the system identification problem usually employs linear output nodes. This makes it possible for the dynamic range of the network output to match that of the training signal input [H1, p. 220]. If the dynamic range is big, it is better to appropriately scale the signals.

The use of a single linear output node is also consistent with the assumption that the residue or prediction error is drawn from a white Gaussian noise process. Behind this assumption is the implication that all of the available information content in the training vectors has been encoded into the weights of the network, resulting in the statistical independence of the samples of the residue (approximation error), a condition satisfied by samples drawn from a Gaussian noise process [H1, p. 67].

NN Representation and the Universal Approximation Theorem

A multilayer neural network trained with the back propagation algorithm is structurally distinguished by a nested sigmoid scheme and can be represented in the following compact manner for the case of a single linear output unit and two hidden layers [H1, p. 189]:

$$F(x, w) = \left(\sum_k w^3_{1k}\varphi\left(\sum_j w^2_{kj}\varphi\left(\sum_i w^1_{ji}x_i + w^1_{j0}\right) + w^2_{k0}\right) + w^3_{10}\right) \quad (2.4)$$

where $\varphi(\bullet)$ defines the sigmoidal activation function, w_{lk} defines the synaptic weight from neuron k in the last layer to the single output neuron l and x_i defines element i of the input vector x. In particular, this structure of nested nonlinear functions serves as a universal estimator in the sense that trained feedforward networks can approximate any continuous multivariable function to any desired degree of accuracy, provided that enough hidden neurons are employed [H1, p. 190].

The following expression is a simple example of the above structure and represents the general equation for a neural network composed of one input neuron, one output neuron and two neurons in the network's hidden layer:

$$F(x,w) = w^2_{11}\tanh(xw^1_{11} + w^1_{10}) + w^2_{12}\tanh(xw^1_{21} + w^1_{20}) + w^2_{10} \quad (2.5)$$

where x is the input data, F(x,w) is the network output, w^1_{ji} and w^2_{kj} correspond to the weights or synapses composing the network, and w^1_{j0} and w^2_{k0} denote the bias associated with each neuron.

2.6 Neural Networks Applied as Mathematical Models

A neural network is able to model a physical system on the basis of a set of training data, a collection of observed (x,y) pairs containing the desired response y for each input x. The training set specifies the input/output mapping of the system. This input/output mapping is learned by the neural network through a self-adaptation process wherein the weights of the network are estimated under the mathematical guidance of the back-propagation training algorithm. The essential mathematical task performed by multilayer neural networks is that of function approximation. The theoretical basis for employing neural networks in the task of function approximation is commonly known as mapping. A modified version of Kolmogorov's existence theorem known as the universal approximation theorem provides the mathematical justification for using neural networks to approximate arbitrary continuous functions.

Mapping Networks

Mapping networks serve to approximate mathematical functions. In general, a mapping network implements a bounded mapping or function from a bounded set of p dimensions to another bounded set of q dimensions [F2, p. 175]. In particular, the input/output relationship of a neural network defines a mapping from a p-dimensional Euclidian input space to a q-dimensional Euclidian output space.

Neural Networks and Function Approximation

A function or mapping machine f(•) is constructed based on a set of collected data

$$(x_1, y_1), \ldots\ldots, (x_N, y_N) \qquad (2.6)$$

known to be representative of the system's input/output mapping. The random

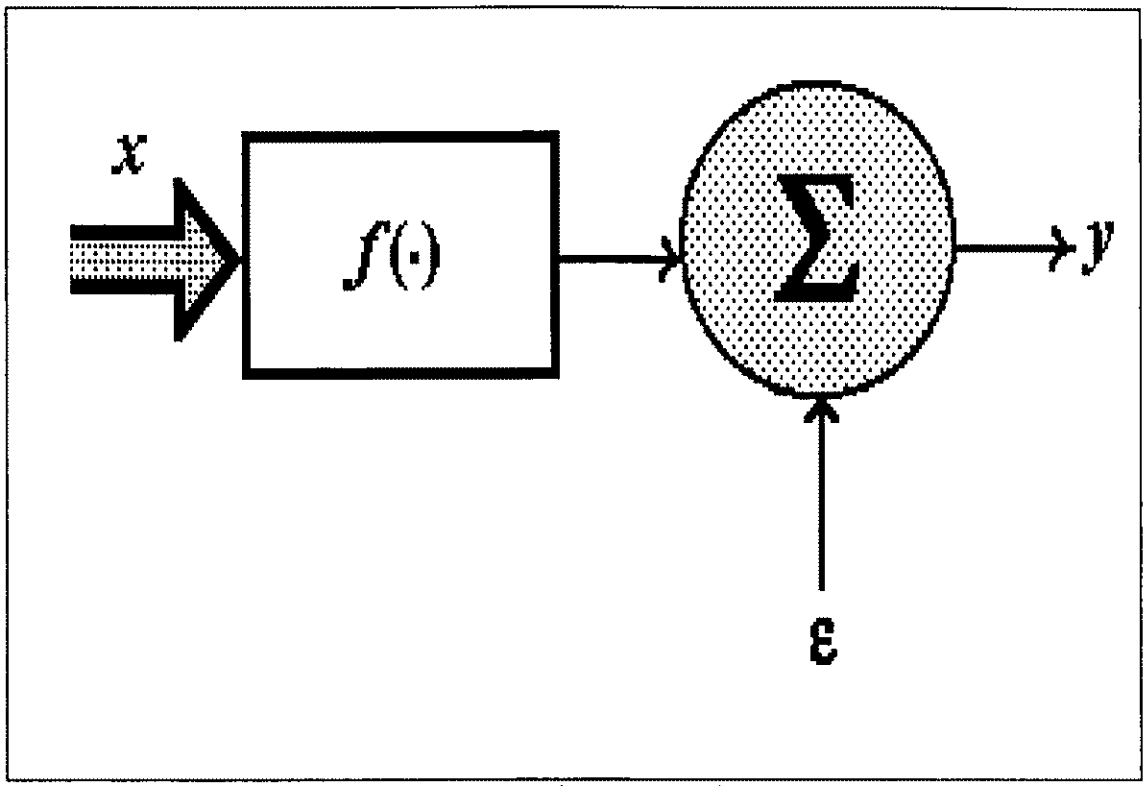


Figure 2.3 An Arbitrary Mapping Machine

expectation error that is representative of the lack of knowledge about the dependence of y on x is represented by **ε**. Figure 2.3 illustrates this idea.

Consider a function **f** and a neural network used to approximate this function. Let F(•,**w**) denote the function which defines the neural network's mapping. The neural network is designed and "trained" so that F(•,**w**) approximates the function **f**. Figure 2.4

illustrates this process. The collected data define a training set, composed of training examples. Each training example is represented by an input/output pair (x,y), where y = f(x). [F2, p. 175].

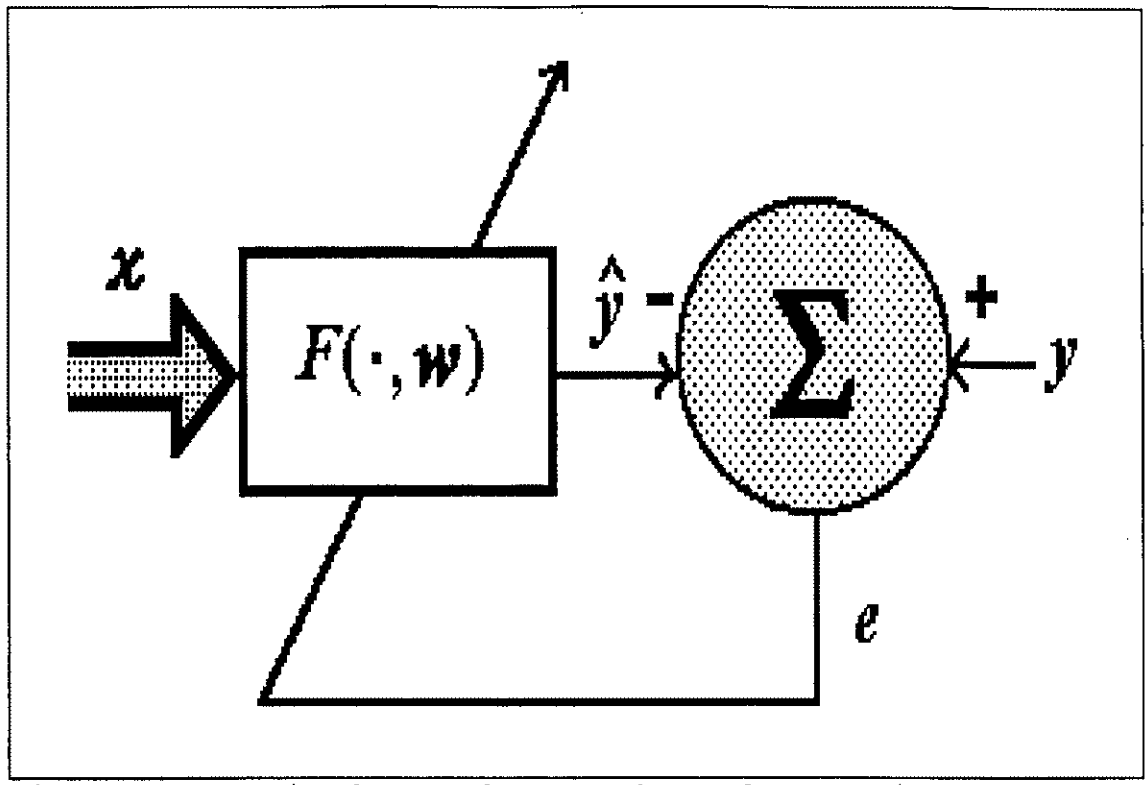


Figure 2.4 Trained Neural Network Used to Estimate the Unknown Function or Mapping Machine

The design goal is that after the training process,

$$f(\boldsymbol{x}) \approx F(\boldsymbol{x}, \boldsymbol{w}) \qquad (2.7)$$

for every x in the domain, not limited to the training set.

The major design objective is for the trained neural network F(•,w) to generalize from the training samples to the entire domain of interest. This objective is commonly referred to as generalization in the neural network community. Generalization is a

ubiquitous issue in all neural network mapping applications. It is the generalization performance or ability of trained neural networks that we seek to quantify in this research.

Theoretical History of Network Function Approximation

Hecht_Nielsen (1987) was the first to examine the capabilities of multilayer neural networks as function estimators. He applied an improved version of Kolomogorov's superposition theorem due to Sprecher (1965). Gallant and White followed (1988) by showing that a single-hidden-layer network composed of monotone "cosine" hidden layer activation functions and a linear output node yields a Fourier series approximation to a given function as its output [H1, p. 181].

However, it was Cybenko (1988, 1989) who demonstrated rigorously that a single hidden layer is sufficient to uniformly approximate any hidden function with support in a unit hypercube [H1, p. 182]. The universal approximation theorem is directly applicable to multilayer neural networks and can be interpreted to be representative of feedforward network architectures in the following general manner:

1. A network consists of p input nodes and a single hidden layer containing M neurons; Inputs to the network are denoted by $x_1,...,x_p$.
2. Hidden layer neuron j has associated synaptic weights $w^1_{j1},...,w^1_{jp}$ and threshold w^1_{j0}.
3. The network's output k is defined to be a linear combination of hidden neuron outputs, the coefficients of which are $w^2_{k1},...,w^2_{kM}$.

Effectively, the universal approximation theorem justifies the use of a single hidden layer in a multilayer neural network for the purpose of function approximation.

A single hidden layer is sufficient for a feedforward neural network to compute a uniform function approximation to a given training set represented by a set of inputs and a desired output [F2, p. 89]. While this theorem is extremely important from a theoretical perspective in that it provides mathematical justification for employing single hidden layer networks as function estimators, it does not give us any guidance towards the construction of these networks. In particular, it offers no guidance on the number of hidden nodes or delay nodes necessary to construct the approximating function. We will find that the number of hidden nodes and delay nodes composing a network play a significant role in the generalization capabilities of that network.

2.7 The Learning Process

The ability of a multilayer network to learn from experience through training is one of the primary characteristics that endows the network with its computing power. The objective of the training process for system identification is to encode within the structure of the neural network the underlying dynamics of how the system being learned transforms signal energy. The knowledge of this input/output mapping is developed or encoded within the network weights by means of a supervised learning process.

2.7.1 Supervised Learning

Supervised learning is a process performed under the guidance of an external "teacher". The "teacher" is represented by a set of input/output examples embodying the knowledge of the input/output mapping the neural network needs to learn. As such, the

"teacher" provides the network with the desired response or goal for each training instance [H1, p. 57]. Figure 2.5 illustrates this process.

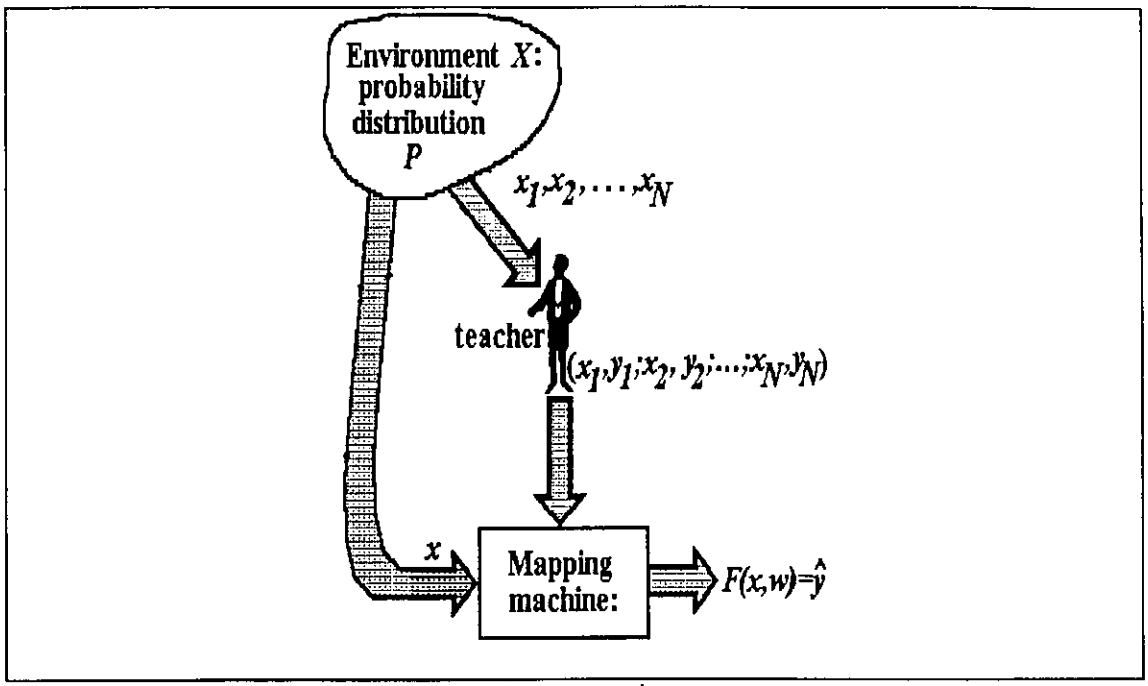


Figure 2.5 The Supervised Learning Process

Network parameter or weight adjustments are based upon information generated from a product of associated neuron inputs and error signals. The error signal is defined to be the difference between the actual response of the neural network and the desired response. The weights of the network are adjusted in an iterative fashion towards the learning goal that the network emulate the teacher. This type of learning is known as error correction learning.

Error Correction Learning

Error correction learning is the basis of the back propagation algorithm used to train multilayer neural networks. It will be instructive to review the principles behind it. Designate **y(n)** to be the desired response for an output layer neuron at time **n**.

Designate $\hat{\mathbf{y}}(\mathbf{n})$ to be the actual response of this neuron at time **n**. The actual response $\hat{\mathbf{y}}(\mathbf{n})$ is the output due to the transformed signal $\mathbf{x}(\mathbf{n})$ which is applied to the input layer of the network and processed throughout each of its layers. The input signal $\mathbf{x}(\mathbf{n})$ and the desired response $\mathbf{y}(\mathbf{n})$ comprise an example presented to the network at time **n**. It is assumed that all of the examples presented to the network have as their source an environment that is stochastic in nature with an unknown underlying probability distribution function [H1, p. 47].

In general, the actual response $\hat{\mathbf{y}}(\mathbf{n})$ of the output neuron does not match that of the desired response $\mathbf{y}(\mathbf{n})$.

Designate the error signal to be the difference between the desired response $\mathbf{y}(\mathbf{n})$ and the actual response $\hat{\mathbf{y}}(\mathbf{n})$

$$\mathbf{e}(\mathbf{n}) = \mathbf{y}(\mathbf{n}) - \hat{\mathbf{y}}(\mathbf{n}) \tag{2.8}$$

The error signal is applied towards the minimization of a cost function, such that the actual response of the output neuron approaches the target or desired response in some statistical sense [H1, p. 47]. Essentially, error correction learning attempts to solve an optimization problem with a specified cost function.

The criterion most commonly used for the cost function is the mean-square-error criterion, defined to be the mean-square value of the sum of squared errors:

$$J = E\left[\frac{1}{2}\sum_{k} e_k^2(n)\right] \tag{2.9}$$

where E is the statistical expectation operator. The summation extends over all of the neurons in the output layer of the neural network. The above criterion assumes that the

underlying processes are wide-sense-stationary. The method of gradient descent involves minimizing the cost function J with respect to the network weights or parameters (Haykin, 1991; Widrow and Stearns, 1985) [H1, p. 47]. However, this optimization procedure requires knowledge of the statistical characteristics of the underlying process. This problem can be overcome by taking an approximate solution to the optimization process. The modified criterion uses the instantaneous values of the sum of squared errors:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{k} e_k^2(n) \qquad (2.10)$$

This modified cost criterion is now minimized with respect to the synaptic weights of the network.

The Delta Rule

The error correction learning rule or delta rule is used to adjust the weights of the network. The adjustment made to the synaptic weight in layer j connected to a neuron in layer k at time n is given by (Widrow and Hoff, 1960) [H1, p. 48]

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \qquad (2.11)$$

where e(n) is the error signal and x(n) is the input signal, also defined to be the output signal of the presynaptic neuron. The adjustment made to a synaptic weight is proportional to the product of the error signal computed at the output layer(k) and the incoming signal broadcasted by the presynaptic neuron in the previous layer(j).

The Nature of the Error Surface Being Searched

A multidimensional surface known as the error surface may be constructed by plotting the cost function J against the synaptic weights of the neural network.

If a neural network is solely constructed from linear processing units, the resulting error surface is a quadratic function of the weights in the network. Such an error surface is bowl-shaped and contains a unique minimum point [H1, p. 49].

A neural network composed of both linear and nonlinear processing units is characterized by a complex and convoluted error surface. Such an error surface may exhibit multiple global minima in addition to multiple local minima [H3, p. 17].

The objective of the error-correction learning algorithm is to begin from an arbitrary point in weight space (the initial values assigned to the synaptic weights determine an exact position on the error surface) and to move toward a global minimum in an iterative fashion. A trained homogeneous network composed of linear processing units always realizes this objective. However, the search along the convoluted error surface associated with nonlinear processing units may be terminated at a local minimum instead of the desired global minimum. Consequently, the global minimum may not be realized in the search. The weight spaces being searched in the system identification problem are of a complex and convoluted nature.

2.8 The Back-Propagation Training Algorithm

The back-propagation training algorithm is the most popular algorithm used to train multilayer networks in a supervised manner. This algorithm is based on the error-

correction learning rule and is actually a generalization of the widely used adaptive filtering algorithm which is commonly referred to as the least-mean-square (LMS) algorithm [H1, p. 138].

Through an iterative process of weight adjustments defined by the mathematical mechanics of the back-propagation algorithm, a feedforward neural network transforms itself into a unique function distinguished by its weighted processing units. It is this iterative process which is referred to as "learning". During this learning process, the neural network is storing arbitrary discrete spatial pattern pairs (x_k,y_k), $k = 1,2,...,N$ under the direction of the back-propagation learning algorithm, where the kth pattern pair is represented by the vectors $x_k = (x_1,...,x_N)$, $y_k = (y_1,...,y_N)$.

The pattern pairs (x_k,y_k) represent the mapping relationship between a function and the transformed output of the function. The back-propagation training algorithm performs this input to output mapping by minimizing a cost function. The minimization of the cost function is realized by a series of iterative weight changes which are directly proportional to an error signal formed between the computed output signal of the neural network and the desired output signal. The cost function typically minimized is the least-squares error. The algorithm directs a series of weight changes which correspond to performing the steepest descent at each step on a surface in weight space whose height at any point in this weight space is equivalent to the error measure. The power of the algorithm lies in its ability to employ weight changes at the hidden units level. Each hidden unit contributes its own error to the overall estimation process. The back-

propagation algorithm measures this error and prescribes weight changes designed to minimize it. The algorithm seeks to find the best set of weights

which will succeed in accomplishing a complex curve fitting process. In particular, the algorithm seeks to generate synapses or weights which contribute to an additive set of curves that model the function the network is "learning".

The algorithm consists of two passes through the network, a forward pass and a backward pass. The forward pass begins at the input layer of the network with the application of the input signal. The synaptic weights of the network remain fixed during this pass and the output of each neuron is computed separately. The computation for the forward pass begins at the first hidden layer which transforms the input vector and terminates with the computation of the error signal at the network output.

The backward pass begins at the output layer with the propagation of the error signal backward through the network, layer by layer, whereby the local gradient signal is computed for each of the network's neurons. This backward pass delivers an error signal to each of the neurons which conveys the amount of "accountability" each of the neurons has for the overall function approximation error. The synaptic weight and bias associated with each network neuron is then changed in accordance with the previously defined delta rule.

Outline of the Back-Propagation Algorithm

The back-propagation algorithm consists of a series of well defined iterative steps, readily implemented by means of computer code. The algorithm is outlined in what

follows. The following algorithm assumes N inputs, one hidden layer with p neurons and a single output neuron.

The Back-Propagation Training Algorithm

1. Assign small random numbers that are uniformly distributed to each of the synaptic weights in the network: i.e. interconnections between neurons and their respective biases.

2. Perform the following set of mathematical transformations for each pattern pair (x_k,y_k):

   Forward Computation

   a. Process the x_k values at the input neurons. The mapping of the input layer is identity. The x_k values and a unit constant bias term are scaled by the first hidden layer. The scaled values are summed and acted upon by the activation function:

$$y^1_j = f(\sum_{i=1}^{N} x_i w^1_{ji} + w^1_{j0}) \qquad (2.12)$$

   for all j = 1,2,...,p, where y_j is the activation value of the jth processing unit or neuron, w_{j0} is the bias of the jth neuron and f(•) is a sigmoid function:

$$f(x) = \frac{1-e^{-x}}{1+e^{-x}} \qquad (2.13)$$

   b. The outputs of the hidden layer neurons then serve as inputs to output layer. These signals are scaled by the output layer weights and summed

up to form the output through the identity mapping of the output node, resulting in

$$\hat{y} = y^2_k = \sum_{i=0}^{p} w^2_{1i} y^1_i \qquad (2.14)$$

where $y1_0 = 1$ corresponds to the neuron's bias.

c. Compute the error signal between the output layer neurons and the desired output values defined by the equation:

$$e_k = y_k - \hat{y}_k \qquad (2.15)$$

for all k = 1,2,...q, where e_k is the kth output neuron's computed error.

Backward Computation

d. Compute the local gradient for each neuron in the hidden layer:

$$\delta_j = y^1_j (1-y^1_j) \sum_k \delta_k w^1_{kj} \qquad (2.16)$$

for all j = 1,2,...,p, where δ_k is the error gradient at neuron k connected to hidden unit j.

e. Adjust the synaptic weights of the network according to the generalized delta rule:

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji} \qquad (2.17)$$

where the weight adjustment is computed for synaptic weights by:

$$\Delta w_{ji} = \eta \delta_j y_i \qquad (2.18)$$

and for neuron biases by:

$$\Delta w_{ji} = \eta \delta_j \qquad (2.19)$$

for all i = 1,2,...,p, and all j = 1,2,...,q, where Δw_{ji} is the weight change made to the "synapse" connecting the ith neuron to the jth neuron or the corresponding bias of the ith neuron. The term η is a positive constant which controls the learning rate. In this thesis η was set to 0.01.

3. Repeat step(2) until the algorithm converges according to the selected error criterion [F2, p. 83].

2.9 The Statistical Nature of the Learning Process

The learning process can be better understood from a statistical point of view. A neural network may be viewed as a mathematical structure that encodes a set of measurements characterizing a specific physical phenomenon [H1, p. 71]. This encoding of a set of measurements is characteristic of regressive modeling. In regressive modeling, we are interested in learning about the relationship between two variables X and Y, a numerical pair embodying some phenomenon of interest to us. Within the context of the system identification problem, X is a vector of measurements on a sequence of past inputs to a dynamical system, and Y is the corresponding vector of outputs of that system based on current and past measurements of X. By obtaining repeated measurements of X and Y, we can build up empirical knowledge about the phenomenon of interest. A neural

network provides a flexible and viable structure to encode this empirical knowledge in. The training set used in the formation of a neural network is actually a less familiar example of a statistic [W2, p. 427]. This statistic or sample provides a complete representation of our empirical knowledge about the system. Averages and correlations can be used to summarize the entire sample. However, we can compress our empirical knowledge in an extremely potent manner by converting it into the synaptic weights of an appropriate neural network [W2, p. 427].

2.10 Mapping Performance of Neural Networks

An important performance attribute of a trained neural network is the generalization capability of that network. Generalization refers to how well the network performs on input/output patterns which have not been used to train or create the network. At the very least, the new input/expected output patterns must be within the amplitude range of the data used for training. Within the context of the system identification problem, generalization refers to the predictive capability of the network. In the paradigm considered in this thesis, the neural net inputs for system identification consists of uniform samples of the plant input and delay versions of it. In this context, generalization refers to the ability of the network to correctly predict the system output, given a new set of samples of the plant input and the delayed versions of it. Generalization is the fundamental problem area for neural modeling in general. Much research has been done on the subject of neural networks and their generalization performance. Factors which are known to affect a network's generalization performance have been identified.

Numerous research efforts have revealed that the generalization performance of neural networks is significantly influenced by the following factors: **the quality and quantity of the training data, the architecture of the network, the complexity of the underlying problem and the learning algorithm used to train the network** [B2],[F2, p. 90],[H2],[H3,17],[S1]. While these factors have been identified, they have not been rigorously quantified, in particular towards the realization of a network's generalization capabilities or associated metrics. A specific architecture is chosen, the weights are fitted to the data and performance on the trained network is evaluated in some manner. Network performance is compared with respect to various architectures and the best performing network is selected. As such, the construction of an approximating function is an iterative process, as previously described in the section on system identification. Central to this iterative process is a means to rate the constructed mapping machines or estimators. More specifically, we need to be able to measure and rate a neural network's generalization performance or predictive capabilities. But how exactly shall the network's generalization performance be evaluated? What type of measure should be employed to reveal this capability?

It is not enough to simply correlate a network's generalization ability with the previously mentioned factors. Training various networks on a trial and error basis while hoping to come up with a good estimator consumes too much time and involves too much chance in the process. The factors which are known to influence generalization performance can be understood in a more quantitative way, as well as applied in a more reliable and effective manner, if we examine the issue of generalization from the

perspective of nonparametric statistical inference. Multilayer neural networks trained by the back-propagation algorithm are examples of nonparametric regression estimators. Viewing them from this perspective will transform what has been a formidable task into a manageable one.

2.11 Neural Networks and Nonparametric Inference

Multilayer neural networks trained with the back -propagation training algorithm are examples of nonparametric, nonlinear regression estimators [G1, p. 1],[W1, p. 1],[W2, p. 451]. Within this context, the learning accomplished under the training algorithm can be formulated as a nonlinear regression problem. This viewpoint is of great significance. Much of the work done with multilayer neural networks has been based on the hope that these networks will show better generalization abilities by being able to develop clever "internal representations" by means of the network's hidden nodes. Indeed, many investigators believe that the hidden layer's ability to implement any nonlinear transformation of the input space can be exploited to "abstract the regularities" from the environment, thereby constructing estimators that promise to solve extremely difficult or impossible problems [G1, p. 45]. This is an unrealistic hope.

In reality, the hidden layers in a neural network are a nonlinear device, subject to the limitations of nonparametric inference methods. These limitations are well known and well understood in terms of what is called the **bias/variance dilemma** [G1, p. 2]. At the core of this dilemma is the estimation error. The impasse arises from two distinctive components of the estimation error known as the bias and the variance. Incorrect model

structure leads to high bias, whereas truly model-free inference (no limit on the number of parameters) results in high variance. If accurate estimators are to be constructed in the form of feedforward networks, both the source and contributions of these two terms must be understood from an applied mathematical perspective. This is so because the estimator we are hoping to construct will be the result of an appropriate balance between the bias and variance components of the estimation error.

2.11.1 Regression and Least-Squares Learning

A typical learning problem involves a feature or input vector **x**, a response vector **y**, where the pair (**x**,**y**) obeys some unknown probability distribution P and the goal of learning is to predict **y** from **x**. A collection of observed (**x**,**y**) pairs containing the desired response **y** for each input **x** compose the training set. Usually these samples are independently drawn from P and many variations are possible [G1, p. 3]. The desired response **y** = y is designated as one-dimensional in what follows.

Learning and Regression Estimation

The learning problem is to construct a function or "mapping machine" f(**x**) based on the data

$$(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots\ldots (\boldsymbol{x}_N, y_N) \qquad (2.20)$$

so that f(**x**) approximates the desired response y.

Generally, f(**x**) is chosen to minimize some cost function. In feedforward networks the sum of observed squared errors is formed

$$\sum_{i=1}^{N} [y_i - F(\mathbf{x}_i)]^2 \qquad (2.21)$$

and F(**x**) is chosen to make this sum as small as possible. The synaptic weights parameterize F(**x**) and the minimization over the sum of observed squared errors is not over all possible functions f(**x**), but over the class which is generated by all allowed values of the parameters [G1, p. 4]. This minimization is one way to estimate a regression. Figure 2.6 illustrates this estimation process.

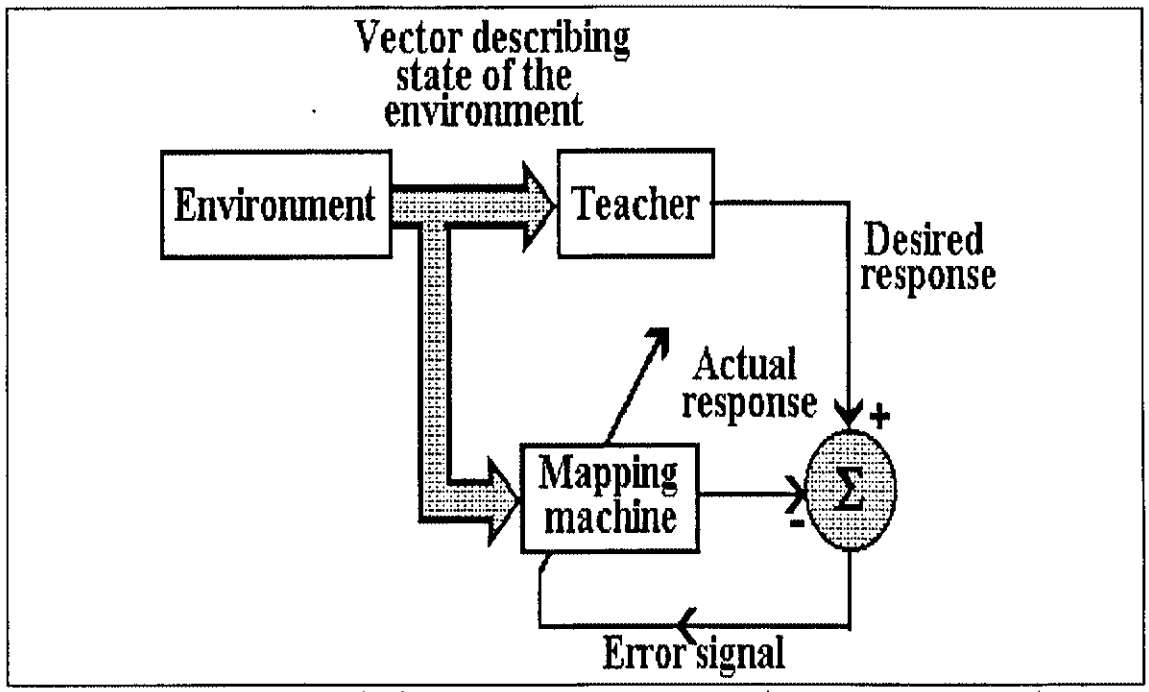


Figure 2.6 Training a NN to Estimate a Nonlinear, Nonparametric Regression

In particular, a trained multilayer neural network estimates the regression of y on **x**, that is the network realizes a deterministic function of **x** that gives the mean value of y conditioned on **x**, E[y|**x**].

The general objective in estimating the regression is to "fit the data" or, more precisely, fit the ensemble from which the data were collected [G1, p. 4]. Theoretically, the regression is an excellent solution, based on the following reasoning. For **any** function f(**x**), and any fixed **x**,

$$E[(y-f(\boldsymbol{x}))^2|\boldsymbol{x}] = E[((y-E[y|\boldsymbol{x}]) + (E[y|\boldsymbol{x}] - f(\boldsymbol{x})))^2|\boldsymbol{x}] \quad (2.22)$$

$$\geq E[(y-E[y|\boldsymbol{x}])^2|\boldsymbol{x}] \quad (2.23)$$

Simply put in words, the regression is the **best** predictor of y given **x** in the mean-squared-error-sense, among all functions of **x** [G1, p. 4].

2.11.2 Bias and Variance Components of Mean-Squared-Error

The regression problem is to construct a function f(**x**) based on a "training set" $(x_1,y_1),(x_2,y_2),....,(x_N,y_N)$ so that future observations of **x** result in the approximation of y. This is sometimes called "**generalization**", a term originating from psychology. Construction of f is dependent on the data. To be explicit about this dependence on the data

$$D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), ... (\boldsymbol{x}_N, \boldsymbol{y}_N)\} \quad (2.24)$$

f(**x**,D) will be written instead of simply f(**x**).

Given a training set D and a particular **x**, a natural measure of the effectiveness of f as a predictor of y is the mean-squared error $E[(y - f(\mathbf{x},D))^2|\mathbf{x},D]$ (2.25) where the expectation is with respect to the probability distribution P [G1, p. 9]. Eq. (2.25) now becomes

$$E[(y-f(\mathbf{x};D))^2|\mathbf{x},D] = E[(y-E[y|\mathbf{x}])^2|\mathbf{x},D] + (f(\mathbf{x},D) - E[y|\mathbf{x}])^2 \quad (2.26)$$

The term

$$E[(y-E[y|\mathbf{x}])^2|\mathbf{x},D] \quad (2.27)$$

does not depend on the data D or the estimator f, it is simply the variance of y given x. It follows then that the squared distance to the regression function

$$(f(\mathbf{x};D) - E[y|\mathbf{x}])^2 \quad (2.28)$$

measures the effectiveness of f as a predictor of y in a natural way [G1, p. 9]. Furthermore, the mean-squared error of f as an estimator of the regression E[y|x] is defined to be

$$E_D[(f(\mathbf{x};D) - E[y|\mathbf{x}])^2] \quad (2.29)$$

where the expectation is with respect to the training set D, the average over the ensemble of possible D.

It is extremely instructive to examine the statistical nature of this measure. The resulting error between the constructed function and the regression being estimated is of a twofold nature. Dual sources of error known as the bias and the variance contribute to the overall mean-squared estimation error.

These two sources of estimation error manifest themselves in the constructed model in distinctive ways. Consider the variance component of the error. It might be that for a particular training set D, f(x;D) is an excellent approximation of E[y|x], and

consequently a reliable predictor of y. However, it might also be the case that f(x;D) is very different for other training set realizations of D and in general varies substantially with D. If so, the estimator is said to exhibit high variance. How does the bias component affect the accurateness of the estimator? It might be the case that the average (over all possible D) of f(x;D) is somewhat far from the regression E[y|x]. When this occurs, the estimator is characterized by a high bias. These conditions of either high bias or high variance can contribute large values to the mean-squared error rendering f(x;D) an unreliable predictor of y. The bias/variance decomposition of the mean-squared error provides a useful way to assess these sources of estimation error.

This decomposition is illustrated in what follows [G1, p. 10].

$$E_D[\,(f(\boldsymbol{x};D) - E[y|\boldsymbol{x}]\,)^2] \tag{2.30}$$

$$= (E_D[f(\boldsymbol{x};D)] - E[y|\boldsymbol{x}])^2 \quad \textbf{\textit{bias}} \tag{2.31}$$

$$+ E_D[\,(f(\boldsymbol{x};D) - E_D[f(\boldsymbol{x};D)]\,)^2] \quad \textbf{\textit{variance}} \tag{2.32}$$

If, on the average, f(x;D) is different from the regression E[y|x], then f(x,D) is said to be biased as an estimator of E[y|x]. In general, the bias depends on the joint probability distribution P. The same estimator (f) may be biased in some cases and unbiased in others.

An unbiased estimator may still allow a large mean -squared error if the variance is large. Even if the following holds

$$E_D[f(\boldsymbol{x};D)] = E[y|\boldsymbol{x}] \tag{2.33}$$

$f(\mathbf{x};D)$ may be highly sensitive to the data and usually far from the regression $E[y|\mathbf{x}]$ [G1, p. 101].

From the above analysis, it is clear that either bias or variance can contribute to an estimator's poor performance. Successful design will involve reducing both terms to an acceptable level. The above analysis allows for a quantitative understanding of an estimator's bias and variance. In order to design an appropriate tradeoff between the terms, it will be instructive to examine them from a qualitative perspective.

The Nature of Bias and Variance

The determination of the appropriate number of weights or parameters in a neural model is a task of an empirical nature. We know that the estimation capabilities of multilayer neural networks are constrained by a bias term and a variance term. It will be useful to relate these constraints to network architecture in general. One of the significant questions to be answered during the design process is how large a network should be employed for a specific estimation task.

Bias, Variance and Neural Network Architecture

A small network with single or few hidden units has a limited repertoire of available functions spanned by $F(\mathbf{x},\mathbf{w})$ over allowable weights and is likely to be biased. Furthermore, poor approximation of the true regression within the class of available functions will result in a substantial bias [G1, p. 12].

This bias may be significantly reduced if we choose to overparameterize the estimator by means of a large number of hidden units and associated weights (in fact, with enough hidden units and associated weights, the network will interpolate the data).

However a large number of parameters results in a high variance, resulting in the need for prohibitively large training sets to reduce the variance contribution to the training error. An overparameterized network pays too much attention to the training data, in essence it memorizes the data. When this occurs, the estimator is overly sensitive to the training data, resulting in a high variance and subsequent poor predictive capabilities. Essentially, the variance of an estimator will be high anytime the estimator becomes too dependent on particular samples observed, that is particular realizations of (**x**,y) [G1, p. 12].

The number of hidden units is an important variable in controlling the bias and variance contributions to the mean-squared error. Additional hidden units result in the increased complexity and versatility of the repertoire of available functions spanned by F(**x**;**w**) [G1, p. 18].

Clearly, we have a dilemma. Neural networks with few parameters produce small-variance, high-bias estimators, where the data are essentially ignored in favor of the constrained architecture. Networks with many parameters span a much more complex and versatile set of functions but are subject to the danger of overfitting the data and subsequent high variance.

If we do not impose structure and introduce bias, the estimator will exhibit substantial dependence on the training data. Furthermore, if the problem being resolved is a complex one, training samples of reasonable size will never adequately cover the space, and the parts of the space which are covered will be highly dependent on the specific training samples.

The only accepted way to avoid having to densely cover the input space with training examples is to deliberately introduce bias into the estimator [G1, pp. 45-46]. This is the only manner in which variance can be eliminated or significantly reduced. But if we are going to deliberately introduce bias, then we must carefully design it so that the bias does not contribute significant error to the estimation error.

2.11.3 The Design of Appropriate Bias

The best solution for the design of our estimator will be a compromise between two extremes. In general, the most effective number of parameters will result in a significant reduction of the variance without introducing too much bias. Reduction of an estimator's variance can only come about through the introduction of bias [G1, p. 46],[H1, p. 75]. More specifically, the only way to control the variance in complex inference problems is to use model-based estimation. However, any model-based scheme is likely to be incorrect or highly biased. The design task now becomes the determination of the most effective number of parameters, the resulting estimator being characterized by the most significant reduction of the combined bias and variance of the estimation error. An important area of consideration in the design of an estimator's bias and variance is the sampling rate of the training signals used to create the function. The choice of sampling frequency for system identification purposes must be empirically determined. In general, the sampling frequency must be at least 20 times the open-loop bandwidth of the system. The choice of sampling frequency establishes the number of delay nodes on the input signal and subsequent number of associated weights.

The above discussion again motivates the need for the development of an appropriate measure, but now we have an idea of just what it is we need to measure.

2.11.4 Predictive Capabilities and Training Data Distribution

Based on the knowledge that a trained neural network is estimating a regression, it is reasonable to expect that the joint probability distribution of the observations (x,y) should assume a significant role in the development of the estimator's predictive capabilities. Additional insight into this observation can be obtained by further examining neural networks as estimators of regressions.

Examination of the cost function used in the estimation process and a mathematically equivalent multiple integral reveals that the synaptic weight vector that will minimize the cost function must also minimize the multiple integral

$$\mathbf{E}[(f(\boldsymbol{x}) - F(\boldsymbol{x},\boldsymbol{w}))^2] = \int_{\mathbb{R}^p} g(\boldsymbol{x})\,(f(\boldsymbol{x}) - F(\boldsymbol{x},\boldsymbol{w}))^2\, d\boldsymbol{x} \quad (2.34)$$

where $x \in \mathbb{R}^p$ and g(x) is the probability density function of x [H1, p. 73].

In general, a trained neural network F(x,w) is characterized by a synaptic weight vector satisfying the minimization of the cost function J(w), and is a mean-squared error minimizing approximation to the conditional expectation function

$$\mathbf{E}[d|\boldsymbol{x}] \equiv \mathbf{E}[y|\boldsymbol{x}] \quad (2.35)$$

or, more specifically, the regression [H1, p. 73].

Based on the above observations, it is clear that the probability density function g(x) plays a critical role in the development of the optimum weight vector w of the neural

network. It will be instructive to further examine the distribution of the training observations and their effect on the generalization performance of the network.

Distribution of the Observations and Generalization

A neural network trained with a specific set of examples representative of the phenomenon of interest will produce small errors on average for values of **x** most likely to occur at the expense of larger errors on average for values of **x** that are unlikely to occur [W2, p. 433]. This leads to the following **constraint** on a neural networks **generalization performance**:

A neural network characterized by the synaptic weight vector **w** that minimizes the previously defined multiple integral for a specific g(**x**) will **not give optimum performance** in an operating environment characterized by a different probability density function. This leads to the constraint that training signals for system identification purposes be characterized by an appropriate probability density function.

2.12 Neural Modeling and Temporal Representation

Neural networks employed for the task of system identification must store and represent time varying patterns. The storage and representation of time varying patterns is a complex task and will not be possible without carefully designing bias into the network's architecture. The generalization measure should serve to identify the "right" bias for the system being identified. Sometimes the right biases can be achieved through proper data representations [G1, pp. 50-51]. With this in mind, we will examine how time varying patterns can be represented with feedforward network architectures.

2.12.1 Temporal Representation and Feedforward Architectures

Memory

The essence of a memory is that it alters its environment in some manner. Anything that remains after the action causing it has ceased is a memory of that action. This research seeks to model systems with memory, in particular dynamical systems. The current output of a dynamic system depends on the effects of past inputs in addition to the current input signal. The system output represents a memory of its previous inputs. We will need to construct a network architecture which supports these memory requirements in its input/output mapping. Specifically, we need to be able to represent and store temporal sequences.

Temporal Sequences of Dynamic Systems

A dynamic system is one whose output (response) depends on both the past and present values of the inputs. The past and present input values are events (in time) which are related by precedence. A general representation of these dynamic relations is expressed by

$$Y_j(t) = F(X_1[t_0, t], X_2[t_0, t], \ldots) \qquad (2.36)$$

where t is a time parameter such that $Y_j(t)$ is the value of Y at time t, the starting time of the system is t_0, and

$$X_i[t_0, t] = \{X_i(s) \mid t_0 \le s \le t\} \qquad (2.37)$$

Such a system is said to have memory since the current output can depend on both the current and/or the past inputs. An example of a dynamic system with memory is an electric circuit which contains elements that store energy (e.g. inductor and capacitors).

The weights and biases of the network encode the knowledge of the network. We need a neural network architecture which encodes knowledge of dynamic memories in the form of temporal sequences. In general, networks learn a mapping from a set of input patterns to a set of output patterns. The learning of a dynamical system requires the network to learn an input/output mapping of temporal patterns. If temporal patterns are to be learned by a neural network, they must be effectively identified, represented and stored within the network's structure, in particular within the network weights.

Temporal Data

Temporal data are one-dimensional and can be treated as a sequence. Spatial data are defined in three dimensions. Both types of data are related by precedence in one dimension. Temporal data consists of precedence relationships between events whereas spatial data consists of precedence relationships between objects [F2, p. 253]. If a neural network is to learn the input/output relationship of a dynamic system, means must be developed to identify, represent and store these precedent relationships between events within the network weights. What types of network architectures will best support this input/output mapping?

Spatiotemporal Neural Networks

Static Neural Networks

In this approach, an entire sequence of temporal data is presented simultaneously on the input layer of the network. Precedence relationships between events are represented by a set of spatial units [F2, pp. 254-255]. The network employed for this task is a multilayer neural network. This approach is a static one as it does not acknowledge the dynamic or temporal nature of the data being processed.

Time-Delay Neural Networks

In this approach, the temporal sequence being processed is delayed by a specific number of uniform time units. The simplest case involves delaying the input signal by one time unit and processing both the present and delayed signals. A time-delay neural network is illustrated in Figure 2.7. The structure of this network admits the dynamics of the system being learned. The output of a time-delay neural network depends on both its past and present inputs. The knowledge encoded within the weights is determined by the output at time t, y(t) and the previous inputs x(t), x(t - d), .., x(t - nd), nd being representative of a specific number of uniform time units the input signal has been delayed by. This architecture directly addresses the dependency of dynamical systems on both past and present inputs and is characterized by a dynamic mapping.

Implicit Temporal Precedence Relationships

Although the encoded knowledge within the weights supports the storing of temporal sequences, the precedence relationships in the temporal sequence being learned are implicit ones. This can be understood by examining the manner in which the

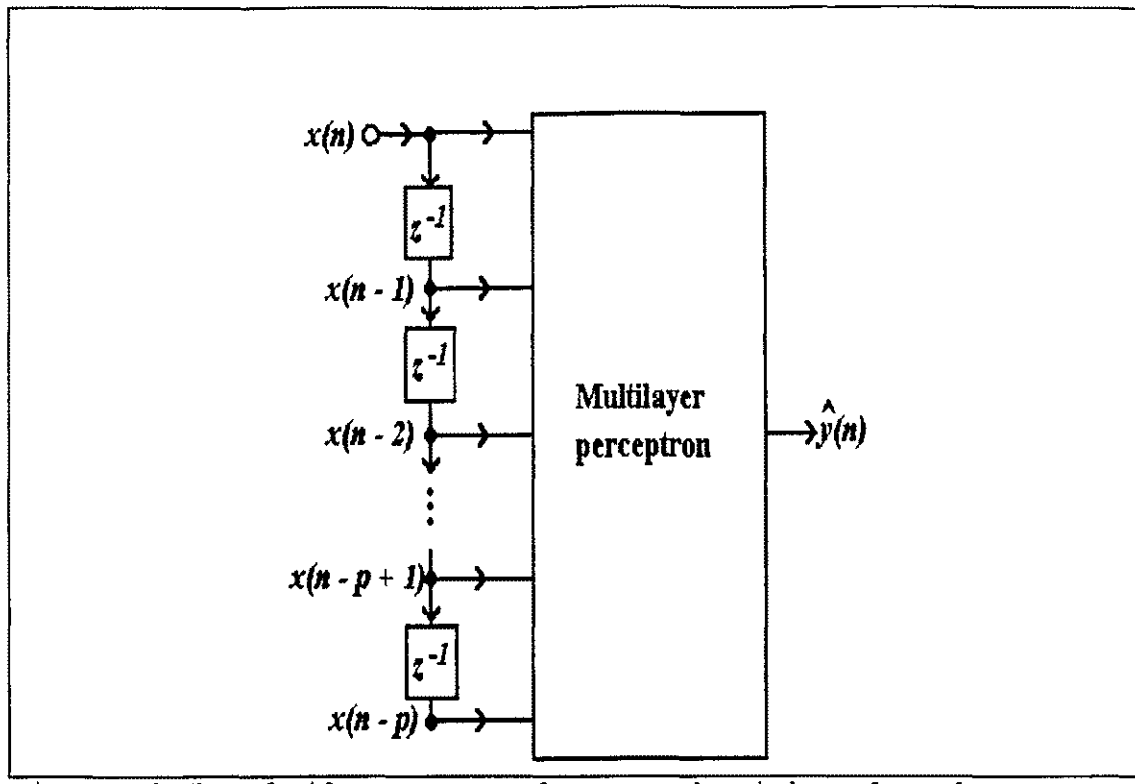


Figure 2.7 Multilayer Neural Network with Delayed Inputs

temporal data is processed. While the processed data begins as a delayed sequence of the input signal, temporal precedence relationships are not maintained during the training process. The temporal sequence is actually being given spatial representation on the input layer. The weighted and delayed input signals are summed up and transformed by each of the hidden nodes. The corresponding activation value for the each of the hidden nodes actually depends upon a spatial summation of the weighted and delayed input signals. This spatial summation of the inputs does not support explicit precedence relationships between events. Specifically, the hidden node information processing consists of integrating the information contained in the input signals over a fixed time period, the information being represented spatially on the input layer. Because there is no integration of temporal information over time involved in this process, the knowledge encoded within

the weights is created from implicit temporal precedences in the temporal sequences used to train the network [F2, p. 259].

A critical task in learning temporal patterns is the identification and storage of the temporal precedence relationships between events. Because a time delay network is only capable of learning these relationships implicitly and the storing of temporal patterns is a key factor in the predictive capabilities of the network, any improvements made in this processing of temporal information should improve the predictive capabilities of the network. Recurrent networks hold great promise in applications of temporal pattern recognition. These networks are capable of learning and storing temporal information [F2, pp. 259-260].

2.13 Design of Training Signals

2.13.1 Training Data Distribution

Training data distribution plays an important role in the formulation of the weight vector and the resulting performance of $F(\mathbf{x},\mathbf{w})$ as a predictor of the desired response. One significant aspect of the training data is its probability density function. This was discussed in detail earlier under distribution of the observations.

The probability density function of the training data defines the operating environment within which optimum performance of the neural network can be expected. Consequently, we cannot expect the network to generalize well outside of this environment. Previous research has demonstrated the sensitivity of neural network performance to the training data set distribution. A medical data base was used for a set

of experiments in which neural nets were applied to a medical diagnosis pattern recognition problem. Two sets of training data were used. One training set was typical data distributed over the classes with a number of patterns in the training set equally distributed for each class. The other training set used a number proportional to the real distribution for each class, representative of the real distribution among the classes. Results from this research indicated that the "real world" data distribution was a better choice for training data and created networks which generalized better. The real world data distribution can be obtained by randomly sampling typical data.

2.13.2 Informative Experiments

An experiment is said to be informative enough if it generates a data set that is informative enough. In particular, an open-loop experiment is informative if its input is persistently exciting [L3, p. 364].

Persistence of Excitation

A quasi-stationary signal {u(t)}, with spectrum $\phi_u(\omega)$ is said to be persistently exciting of order n if, for all filters of the form [L3, pp. 362-363]

$$M_n(q) = m_1 q^{-1} + \ldots + m_n q^{-n} \qquad (2.38)$$

the relation

$$|M_n(e^{jw}|^2 \phi_u(\omega) \equiv 0 \quad \Rightarrow \quad M_n(e^{jw}) = 0 \qquad (2.39)$$

It follows that u(t) is persistently exiting of order n if its spectrum is different from zero at least n points in the interval $-\pi < \omega \leq \pi$.

A strengthened version of the above concept results in the following: A quasi-stationary signal {u(t)} with spectrum $\phi_u(\omega)$ is said to be persistently exciting if $\phi_u(\omega) > 0$, $\forall\ \omega$.

2.14 Conclusions

Within this chapter, we have presented an overview of the design of multilayer feedforward neural networks applied to the system identification problem. We have introduced neural networks and examined how they can be applied in the identification of dynamical systems. The fundamental task in this application is the development of an appropriate generalization measure. We have examined the learning process from a statistical point of view and examined the issue of generalization from the perspective of nonparametric statistical inference. Within this context, multilayer feedforward neural networks were seen to be nonparametric estimators of regressions. We analyze generalization within this statistical perspective and formulate a generalization metric based on the resulting statistical interpretation in the next chapter.

CHAPTER THREE

GENERALIZATION PERFORMANCE METRICS AND MODEL VALIDATION

3.1 Introduction

The system identification process is an iterative one. Essential to this process is the criterion applied in the ranking of different neural network structures. A well designed feedforward neural network will realize a successful nonlinear modeling of the physical phenomenon from which the input/output signals used to train the network have been generated. Quantification of this objective is known as model validation.

3.2 Organization of the Chapter

Section 3.3 introduces the concept of model validation which is fundamental to the system identification procedure. In section 3.4 we discuss the background for the generalization metric and formulate a criterion function designed to measure the size of a trained neural network's prediction error.

3.3 Model Validation

The back-propagation algorithm will search for an optimal set of parameters in weight space, yielding the "best" neural network within the chosen multilayer feedforward network architecture. However, we need to ask if this particular "best" neural network is good enough. Answering the question of whether or not the "best" network structure

resulting from the gradient search is good enough is the problem of model validation. The actual question formulated by model validation is threefold [L3, p. 424].

First, model validation needs to measure the extent that the model has "captured" the training data. Has the knowledge about the environment of interest been successfully encoded within the weights of the neural network? The essential question being posed is this : Does the model explain the data well enough, that is, does it "agree sufficiently well with the observed data".

Second, model validation needs to address whether or not the model is good enough for its intended purpose. In particular, is this network good at generalizing? The essence of a trained neural network is its predictive ability. A well designed neural network applied to the system identification problem realizes a nonlinear filter characterized by good prediction or generalization capabilities. We need to be able to accurately measure the generalization capabilities or predictive powers of trained neural networks.

Third, model validation needs to address the extent that the model describes the "true system". The true system is an esoteric entity that cannot be captured with mathematical equations [L3, p. 430]. At best we will capture partial descriptions of the natural phenomena we seek to model.

3.4 The Generalization Performance Measure

3.4.1 Perspective of the Measure

We are going to view the learning process as a means to accomplish a complex curve fitting task. A trained neural network realizes an estimation of a nonlinear regression. More specifically, a trained neural network results in the formulation of a deterministic function of **x** that gives the mean value of y conditioned on **x**. In particular, the network is an estimate of the regression of y on **x**, that is E[y|**x**].

3.4.2 Curve Fitting and Regression

Statistical data contain a relationship between two variables. It is often desired to express this relationship in mathematical form by forming equations that connect the variables. The general problem of finding a mathematical relationship to represent the data is called curve fitting. The resulting curve is called a regression curve and the resulting equation is a regression equation. In order to create a "best" regression equation, a criterion of goodness of fit is employed. The most common goodness of fit criterion employed is the least-squares criterion. This same criterion is employed by the back-propagation training algorithm. Construction of a regression curve involves three basic components: data, a model structure to be fitted, and a goodness of fit criterion. It will be useful to briefly consider this criterion.

Designate the difference between the regression curve and corresponding value of y associated with x as

$$d_i, \; i = 1,2,...,n \tag{3.1}$$

The criterion of goodness of fit being employed is that

$$d_1^2 + d_2^2 + \ldots + d_n^2 = \text{a minimum} \tag{3.2}$$

The least-squares criterion weights errors on either side of the regression curve equally and weights larger errors more than smaller errors. In curve fitting applications, a popular model structure is a polynomial of the form

$$y = a + bx + cx^2 + \ldots + mx^p. \tag{3.3}$$

In general, the construction of a regression curve is accomplished by estimating the model structure parameters. These parameters are determined by minimizing J in eq. (2.9) with respect to them. The overall objective of the estimation process is construction of a regression curve which realizes a minimum distance from the corresponding values of y associated with x.

Regression Curves of Trained Feedforward Networks

The regression curve being constructed by training feedforward neural networks is a complex one. A weighted linear combination of nonlinear sigmoid functions transforms weighted input sequences of data (samples of past inputs to the system being identified). A multidimensional function is being constructed during the training process, resulting in a nonlinear regression curve. We are interested in creating a regression curve that best approximates the mean value of the data. With such a complex structure, how can a best fit possibly be assessed?

Assessment of Best Fit

Consider again the distance between the regression curve being estimated and the actual regression. We propose to measure the mean of this distance. That is, determine

the average distance from the regression curve to the regression. Furthermore, let us also take into consideration the variance of this distance. An acceptable mean value for the distance will still result in poor generalization if the spread of the mean is large. A large variance will allow for large distances between the constructed curve and the regression being estimated. Specifically, we are interested in determining the average value of the distance between the estimated regression curve and the regression, as well as the spread or the variance of this average. We have already examined a previous proposal for such a measure.

Reexamining Bias and Variance

We know that a natural measure of the effectiveness of the function realized by the network as a predictor of y is

$$(F(\boldsymbol{x}, \boldsymbol{w}) - \mathbf{E}[y|\boldsymbol{x}])^2 \tag{3.4}$$

Furthermore, the mean-squared error of f as an estimator of the regression is

$$\mathbf{E}_D[(F(x, w) - \mathbf{E}[y|\boldsymbol{x}])^2] \tag{3.5}$$

where the estimation is with respect to the training set D, where D is defined in eq. (2.24).

The estimator may vary substantially with D or it may be that the average over all possible D is rather far from the regression. We know these two sources of estimation error to be the bias and variance, previously defined to be

$$E_D[(F(\boldsymbol{x};D) - E[y|\boldsymbol{x}])^2] \quad \textbf{msee} \tag{3.6}$$

$$= (E_D[F(\boldsymbol{x};D)] - E[y|\boldsymbol{x}])^2 \quad \textbf{bias} \tag{3.7}$$

$$+ E_D[(F(\boldsymbol{x};D) - E_D[f(\boldsymbol{x};D)])^2] \quad \textbf{variance} \tag{3.8}$$

The sources of the estimation error are clear. An extremely effective measure can be formulated for a neural network's generalization performance in system identification: measure these two sources of estimation error.

It is necessary at this point to formally introduce a modification in the previously defined bias and variance components. We have introduced the measure within the context of pattern recognition. In such applications, the neural network is seen to be characterized by a static mapping. We are performing dynamical system identification with these networks. In Sect. 2-12 a time-delay neural network was introduced as a model of a dynamical system. The entire neural network may be viewed as a black box configuration. In the dynamic mapping realized by a TDNN, a neural network estimates the mean value of y(t) at each time instant t conditioned on past values of the input. We formalize these observations by redefining the deterministic function of **x** realized by the network. In particular, a time-delay neural network estimates

$$E[y(K_1T)|x(KT)]. \tag{3.9}$$

$$K \leq K_1 \tag{3.10}$$

$$K = 1,2,... \tag{3.11}$$

In order to simplify the notation, let

$$E[y|\mathbf{x}] \doteq E[y(K_1T)|\mathbf{x}(KT)]. \tag{3.12}$$

$$K \leq K_1 \tag{3.13}$$

$$K = 1,2,... \tag{3.14}$$

It will be informative to reconsider the bias and the variance in some detail. We will begin with the bias. The bias of the estimation error is mathematically defined to be the following

$$(E_D[F(\boldsymbol{x};D)] - E[y|\boldsymbol{x}])^2 \tag{3.15}$$

The bias can be physically interpreted from two distinct perspectives. To best understand these perspectives, it will be useful to recall several concepts based in the field of probability.

A numerical result of an experiment is called a random variable and is usually denoted with a capital letter like X,Y,Z. The expectation or expected value or mean value of a random variable X is a weighted average of the values of X, where each value x is weighted by the probability of its occurrence. If the expectation is denoted by E[X], then

$$\mathbf{E}[X] = \sum_x xP(X = x) \tag{3.16}$$

A sample space corresponding to an experiment is a set of outcomes such that exactly one of the outcomes occurs when the experiment is performed. Any outcome or collection of outcomes in a sample space is called an event [A1, p. 1, p. 76].

The present format of the bias expression directs our attention towards the expected value of the output(s) of many statistical estimators or networks. This averaged output for many estimators is compared against the regression being approximated. Within this format, each estimator or network is formed from an independent set of training examples and represents a single trial in a random experiment. Each network performs an estimation of the regression being approximated, the outcome being representative of an event. The same random pattern is used to evaluate the performance of each of the individual networks. Evaluation of the bias in this format involves the construction of many different estimators on statistically independent training sets, and averaging their performance on an identical previously unseen pattern. Estimation of the bias from this perspective appears to be characterized by computational intensity as well as time consuming. What about the other perspective of the bias?

Instead of defining the outcome of a single estimator for a specific random pattern to be an event in our random experiment, what if we define an event to be the outcome for a single point in a random pattern for a single estimator? That is, we will evaluate the average performance of a single estimator on many random patterns of the regression we are seeking to approximate instead of evaluating the performance of many estimators on the same random pattern. This perspective is identical to the concept of measuring the average distance of the regression curve from the regression. The random patterns used for this performance evaluation will be a white noise series and a random amplitude pulse train. The entire white noise series is a statistically independent pattern. The pulse train is not a statistically independent pattern, however the random amplitudes represent a

pattern the neural network has not seen. Furthermore, the amplitude of each sample is independent of the amplitude of the corresponding sample in the pattern used to construct the neural network.

The variance of the estimation error is mathematically defined to be

$$E_D[(F(\boldsymbol{x};D) - E_D[F(\boldsymbol{x};D)])^2] \qquad (3.17)$$

Like the bias, the variance can be physically interpreted from two different perspectives. The first perspective examines the spread or the variance of the output of many estimators for the same random pattern. The second perspective defines an event to be a single random outcome for a single estimator. From this perspective, the spread of a random number of outcomes for the same predictor are considered. Once again, we will implement the second perspecitive.

In summary, we want to implement a measure that estimates the mean value of the distance and the variance of this distance between the regression curve and the regression. Simply put, we want to measure the mean and the variance of the estimation error.

3.4.3 Mean and Variance of the Estimation Error

We prove the validity of the second perspective with the following analysis. The mean-square value of the estimation error is given by

$$E_D[(F(\boldsymbol{x};D) - E[y|\boldsymbol{x}])^2] \tag{3.18}$$

$$= (E_D[F(\boldsymbol{x};D)] - E[y|\boldsymbol{x}])^2 \quad \textbf{\textit{bias}} \tag{3.19}$$

$$+ E_D[(F(\boldsymbol{x};D) - E_D[F(\boldsymbol{x};D)])^2] \quad \textbf{\textit{variance}} \tag{3.20}$$

We desire to explicitly measure the bias and variance components of the error between the estimator and the regression. The neural network estimates the regression with an error. Let F(x;D) be replaced by the following expression

$$F(\boldsymbol{x};D) = \mathrm{E}[y|\boldsymbol{x}] + \epsilon\ , \tag{3.21}$$

where substitution of eq. (3.21) into eqs. (3.18), (3.19) and (3.20) results in the following:

$$E_D[(E[y|\boldsymbol{x}] + \epsilon - E[y|\boldsymbol{x}])^2] \tag{3.22}$$

$$= (E_D[E[y|\boldsymbol{x}] + \epsilon] - E[y|\boldsymbol{x}])^2 \tag{3.23}$$

$$+ E_D[(E[y|\boldsymbol{x}] + \epsilon - E_D[E[y|\boldsymbol{x}] + \epsilon])^2] \tag{3.24}$$

Simplification of the transformed expression leads to an expression for the mean-squared estimation error which depends explicitly on the error alone

$$E_D[\epsilon^2] = (E_D[\epsilon])^2 + E_D[(\epsilon - E_D[\epsilon])^2]. \qquad (3.25)$$

We note that the expression is defined to be the mean value of the estimation error squared (previously referred to as the bias) summed with the variance of the estimation error.

3.4.4 Numerical Evaluation of the Estimation Error

The predictive capabilities or generalization performance for each estimator will be evaluated on input sequences of Gaussian white noise and random amplitude pulse trains. Both sequences will be representative of a random pattern the networks have not been trained on and will be referred to as validation signals. The same random pattern will be used to evaluate all estimators constructed with training data characterized by the same sampling rate. We outline the evaluation of the mean-squared estimation error for a neural network applied to system identification in what follows.

Definition of Error

An event in our experimental measure is defined to be the difference between the target value y and the network output F(**x**,**w**) for a single sample i in a validation signal. We define this event mathematically to be:

$$\epsilon_i = y_i - F(\boldsymbol{x}, \boldsymbol{w}) \qquad (3.26)$$

where ε_i is an element of the previously defined mean-squared estimation error vector in eq. (3.25).

It is necessary to make the index of accuracy insensitive to the dynamic range of the input [L1, p. 448]. The most straightforward means of accomplishing this objective is to scale each error event by the magnitude of the desired target or output. It is also necessary to weight errors on both sides of the regression surface equally. This can be accomplished by taking the absolute value of both the error and the desired target. Based on these modifications, the error being approximated is defined to be

$$\epsilon_i = \frac{|e_i|}{|y_i|} \qquad (3.27)$$

for a single event. All events where $y_i \equiv 0$ are removed from the validation signal and are not included in the generalization measure. Based on this formal definition of the error, the bias and variance of the mean-squared estimation error are numerically estimated as follows:

Numerical Estimation of the Bias

The bias is mathematically defined to be

$$(E[\epsilon])^2 = \bar{\epsilon}^2 . \qquad (3.28)$$

We recognize this as the square of the mean of the estimation error. We define the numerical estimation of the bias to be

$$\left(\frac{1}{N}\sum_{i=1}^{N}\frac{|e_i|}{|y_i|}\right)^2 . \qquad (3.29)$$

Numerical Estimation of the Variance

The variance is mathematically defined to be

$$\mathbf{E}[(\epsilon - \mathbf{E}[\epsilon])^2] . \qquad (3.30)$$

We define the numerical estimation of the variance to be

$$\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{|e_i|}{|y_i|} - \bar{\epsilon}\right)^2 . \qquad (3.31)$$

Numerical Estimation of the Mean-Squared Error

The mean-squared error is simply the sum of the bias and variance components of the error. We define the numerical estimation of the mean-squared error to be

$$\mathbf{E}[\epsilon^2] = \left(\frac{1}{N}\sum_{i=1}^{N}\frac{|e_i|}{|y_i|}\right)^2 + \frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{|e_i|}{|y_i|} - \bar{\epsilon}\right)^2 . \qquad (3.32)$$

For each measure, N is defined to be the number of samples in the validation signal.

3.5 Conclusions

In this chapter, we have introduced a practical generalization metric to be used in the design of time-delay networks for system identification purposes. The metric is well known, but we did not find its application as a generalization/prediction measure for neural networks in control systems. The metric has been the focus of the research and will be used in several design applications.

CHAPTER FOUR

NEURAL MODEL DESIGNS FOR SYSTEM IDENTIFICATION

4.1 Introduction

In this chapter we present three examples in which we identify both linear and nonlinear systems by means of trained multilayer feedforward neural networks. In these examples, we illustrate the use of the generalization measure we have developed as the criterion for the selection of the best neural network architecture as well as the stopping criterion for training. We identify a first and second order linear system characterized by the same cutoff frequency of 1 rad/s. We also identify a nonlinear system which is descriptive of a mass-spring-damper with a stiffening spring.

4.2 Organization of the Chapter

In section 4.3 we design a multilayer feedforward network for a first order system with a bandwidth of 1 rad/s. We construct a total of 300 estimators and use the generalization metric to select the best predictor for the system. We compare these results with another cross-validation metric. In section 4.4 we illustrate the design process for a second order system characterized by the same cutoff frequency of 1 rad/s. We apply the same design process towards the identification of a nonlinear system in Section 4.5 followed by the conclusion of the chapter in Section 4.6.

4.3 Design Example One

Consider the dynamical system modeled by the following Laplace transform

$$G(s) = \frac{1}{s + 1} \qquad (4.1)$$

transfer function. This first order system is of a low pass nature and is characterized by a cutoff frequency of 1 rad/s. To construct an estimator for this system, we must generate an appropriate set of training examples and select appropriate neural network architecture to encode the system dynamics within.

4.3.1 Generation of Training Examples

We generate the training examples in software which supports the simulation of dynamical systems. Our training examples are formed by a computer simulated system transform of a pulse train characterized by an amplitude range between 0 and 1 followed by a time series of Gaussian white noise. Figure 4.1 illustrates the training data.

The pulse train contains the primary frequency components being passed by the low pass filter as well as higher order frequency components arising from the sudden amplitude change in the time domain. We define the length of time for a single pulse to be equivalent to the settling time of the system being identified. For this particular system, the settling time for a step input is known to be approximately 5 s. An appropriate sampling time for the training signals can be determined by considering the bandwidth of the system. A rule of thumb is to select a sampling frequency existing in the range of 10 - 30 times the bandwidth of the system. For this particular system, the bandwidth is known to be 1 rad/s. Based on the above design specifications, we generate

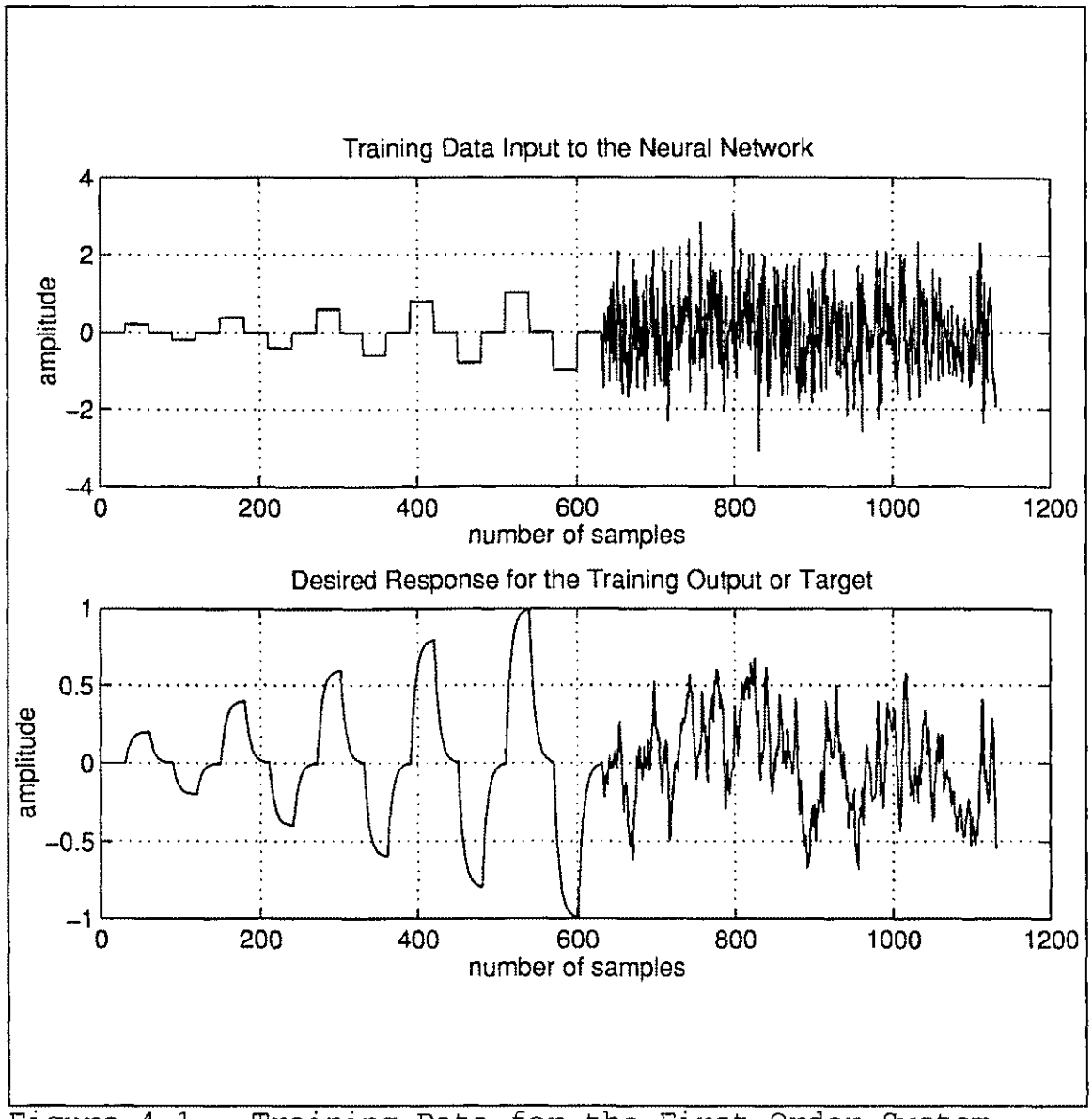


Figure 4.1. Training Data for the First Order System

three sets of training signals, characterized by three different sampling times. We choose our sampling times to be 0.25 s, 0.1667 s and 0.1 s.

4.3.2 Selection of Network Architecture

For this particular design example, we specify the neural architecture to be that of a time-delay neural network. In this approach, the input signal is delayed by a specific

number of uniform time units, defined to be the sampling time of the system being identified. The specific number of time units is dictated by the pulse train signal. Weighted samples of the input signal are summed and transformed by a single layer of hidden nodes characterized by hyperbolic tangent nonlinearities. These transformed signals are filtered through a single weight preceding the linear output neuron. The output signal is coded via the linear output neuron, each training instance represented by a discrete point in time.

Selection of Delay Nodes

The number of input nodes or delay nodes is representative of the number of uniform time units or time sequence delays defining the input layer of the network. The minimum number of delay nodes is constrained to be equivalent to the number of samples in a single pulse of the pulse train. Based on this constraint, we will need to construct three distinct estimators, each being characterized by a different number of delay nodes. The number of delay nodes for each of the networks can be determined by means of the following relationship:

$$\text{Pulse Width} = \text{Sampling Time} * \text{Samples/Pulse}, \tag{4.2}$$

where the pulse width is defined to be the time duration of a single pulse. The above relationship, coupled with previous sampling time and pulse width constraints, establishes the delay node requirements for the three network architectures. Each architecture is structurally distinguished by a sequence of 20, 35 or 50 delay nodes on the input layer of the network.

Selection of the Number of Hidden Units

The number of hidden layer nodes determines the complexity of the function represented by the trained neural network. Additional hidden layer nodes allow for the construction of a more complex and adaptable function, which we expect to be characterized by a higher variance. We shall see in these experiments that convergence issues are extremely important and introduce significant variation in the network's behavior, in particular, with respect to predictive capabilities and generalization performance. For this set of experiments, we vary the number of hidden nodes between 1 and 5.

4.3.3 Formation of the Estimators

In this system identification experiment, we construct a total of 300 estimators, each neural network distinguished by a specific number of delay nodes, hidden units and training cycles. Network configurations have a fixed number of delay nodes, imposed by the sampling time and are varied by means of the number of hidden units, ranging from one to five. We train each of the networks with the back-propagation algorithm for a total of 5,000 training cycles. Network configurations are saved every 250 training cycles during the training process. For each specific network architecture, a total of 20 different estimators are formed and evaluated.

Generalization performance for each estimator is evaluated on two cross-validation signals. These signals are a pulse train sequence characterized by random amplitudes and a Gaussian white noise sequence, independent of the sequence used to

construct the estimators. The criterion we employ in the model validation process is the previously defined generalization metric eq. (3.32) based on the bias and variance components of the estimation error. This criterion will be compared with an alternate cross-validation criterion that measures the two-norm of the prediction error. This metric is defined to be

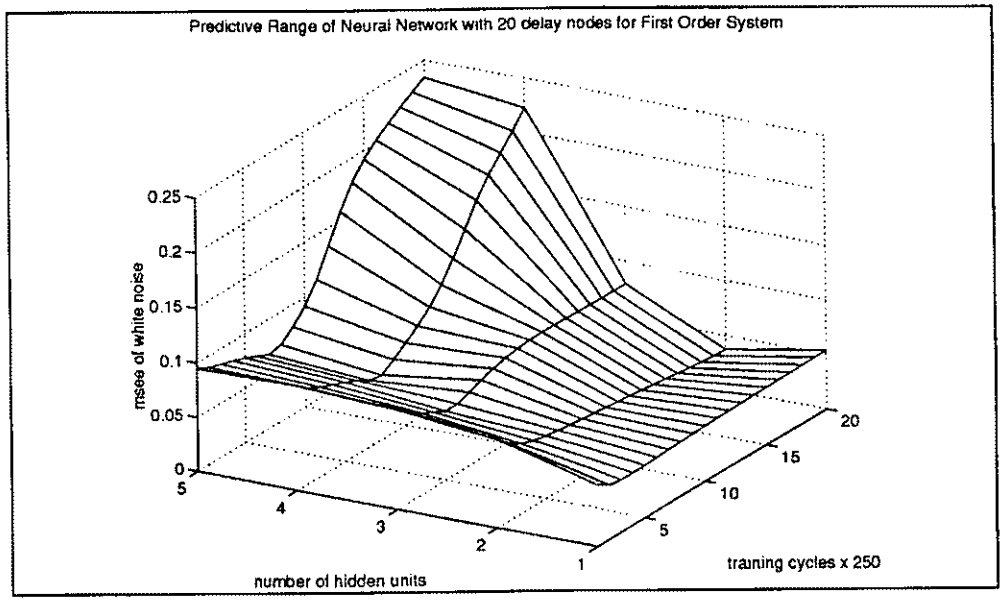
$$\frac{(\sum_{i=1}^{n} |\epsilon_i|^2)^{1/2}}{(\sum_{i=1}^{n} |y_i|^2)^{1/2}} \qquad (4.3)$$

where ε represents the error between the actual network output F(**x**,**w**) and the desired output y and n is the number of samples in the validation signal.

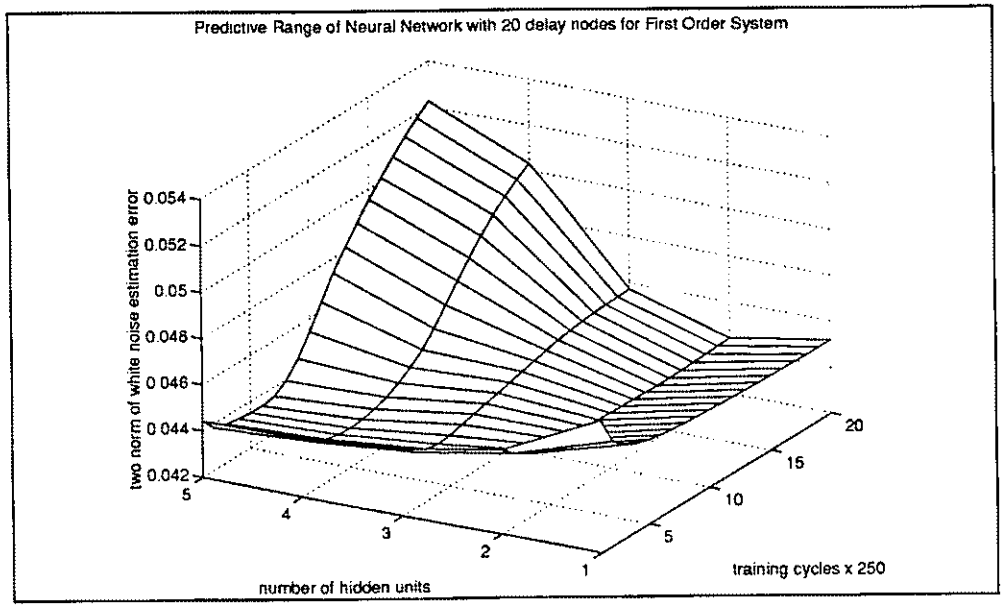
4.3.4 Experimental Results

Figures 4.2 - 4.7 illustrate the mean-square estimation error and the two norm applied to the estimation error for each of the 15 network architectures, as a function of the number of training cycles and the number of hidden nodes. Each criterion is employed on each network's transformation of the previously mentioned validation signals. Tables 4.1 - 4.6 summarize the results for the minimum bias, variance and msee achieved for each of the network configurations and number of training cycles. Tables 4.1 and 4.2 summarize results for the 20 delay node network. Tables 4.3 and 4.4 summarize results for the 30 delay node network. Tables 4.5 and 4.6 summarize results for the 50 delay node network. The bold numbers in each of the tables indicate the

lowest achievable minimums of the bias, variance and msee for specific network configurations and associated number of training cycles.

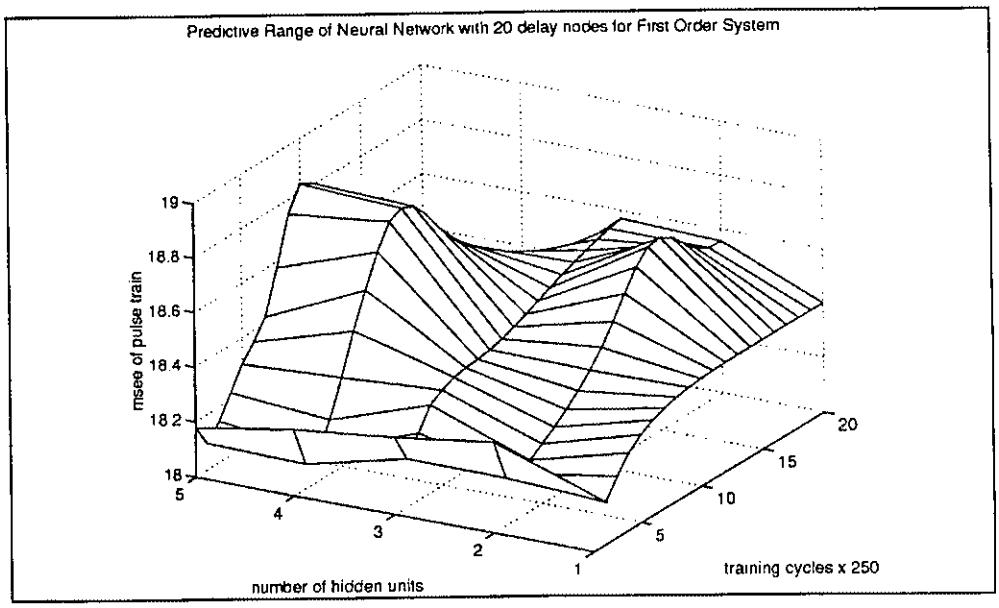


(a)

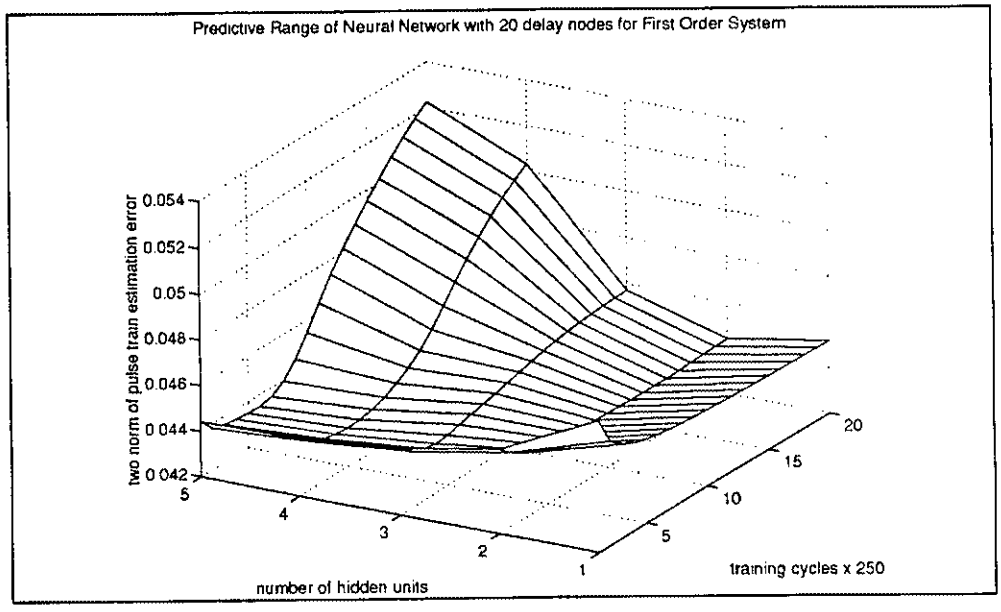


(b)

Figure 4.2. Relation between the number of training cycles and hidden nodes and the white noise estimation error for 1st order system: a) Mean-squared error. b) Two norm of error.

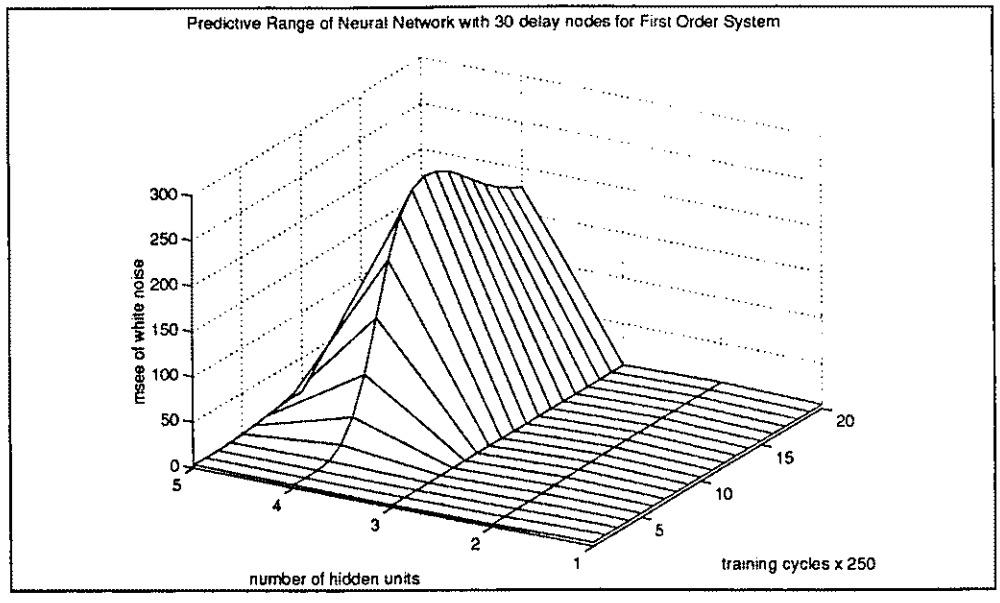


(a)

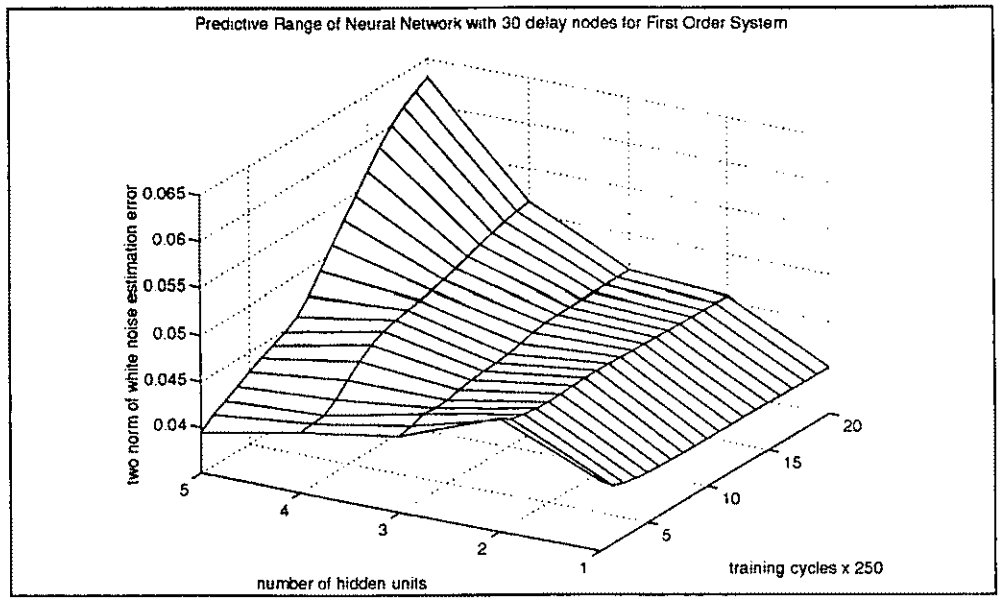


(b)

Figure 4.3. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for 1st order system: a) Mean-squared error. b) Two norm of error.

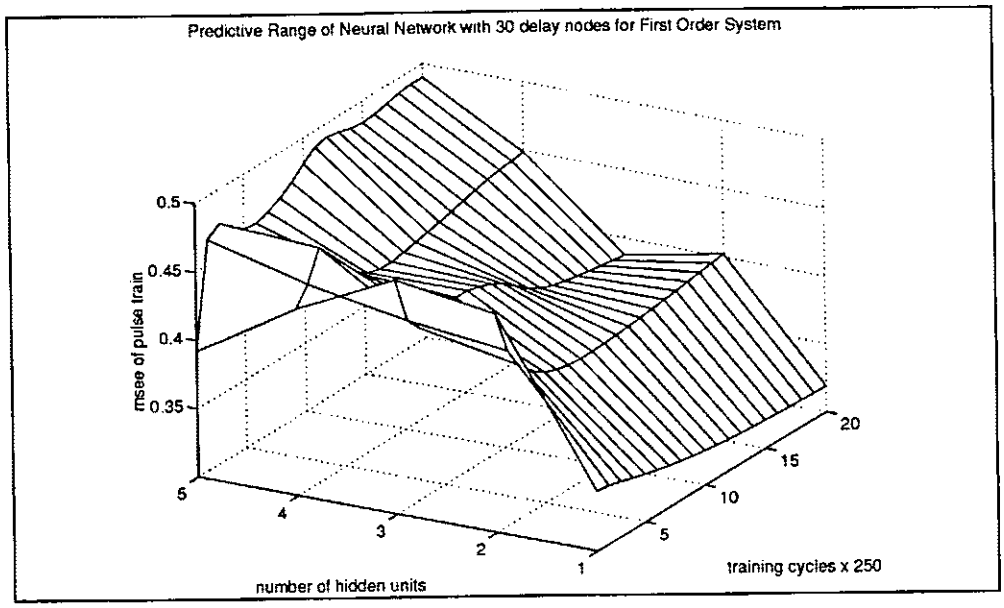


(a)

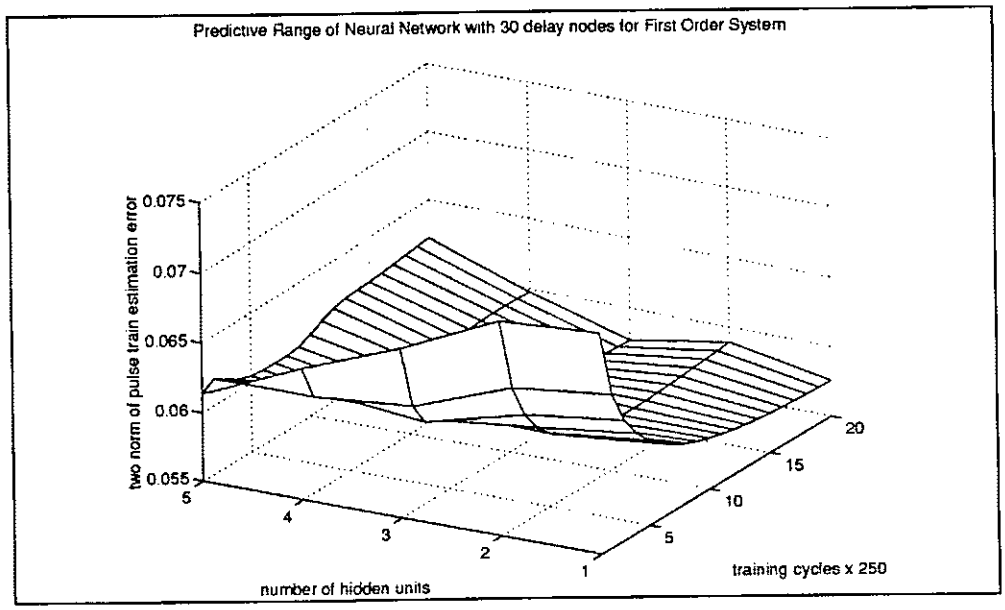


(b)

Figure 4.4. Relation between the number of training cycles and hidden nodes and the white noise estimation error for 1st order system: a) Mean-squared error. b) Two norm of error.

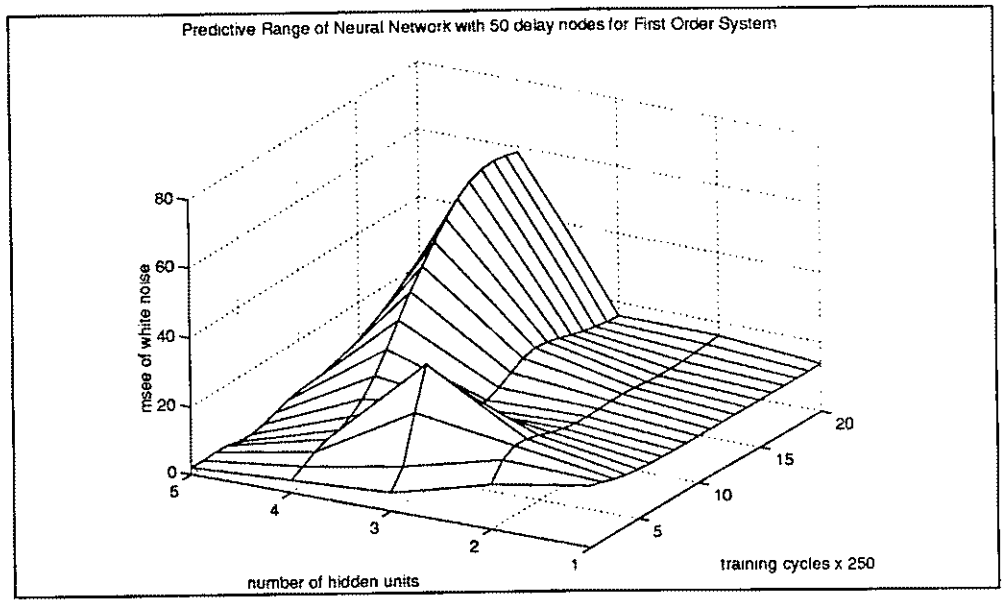


(a)

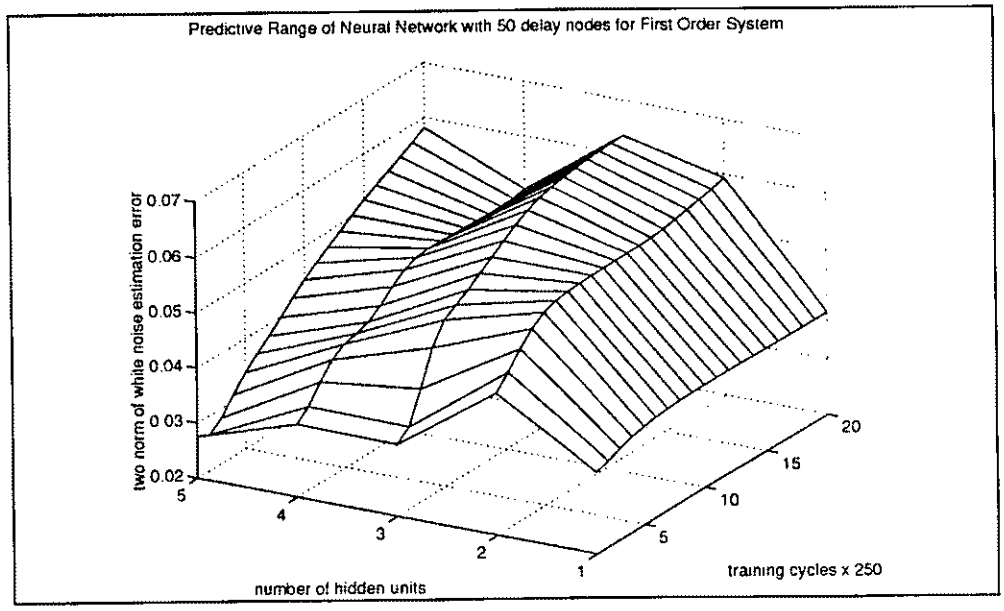


(b)

Figure 4.5. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for 1st order system: a) Mean-squared error. b) Two norm of error.

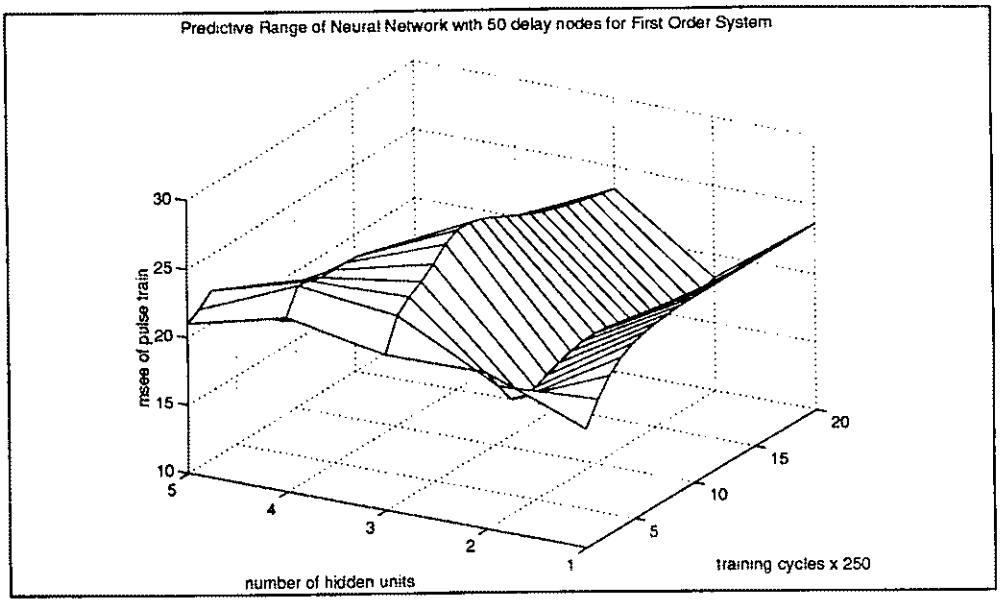


(a)

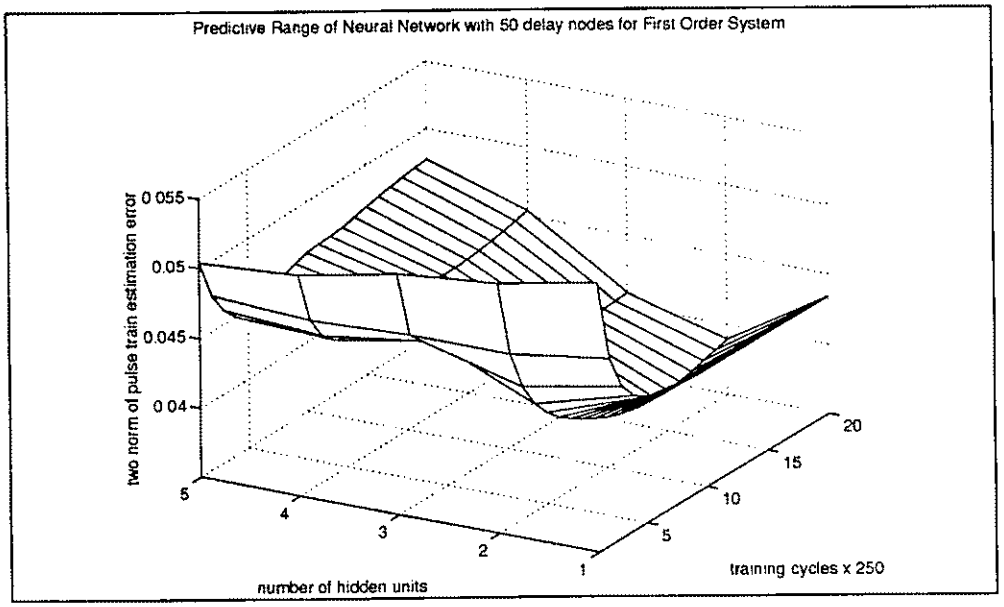


(b)

Figure 4.6. Relation between the number of training cycles and hidden nodes and the white noise estimation error for 1st order system: a) Mean-squared error. b) Two norm of error.



(a)



(b)

Figure 4.7. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for 1st order system: a) Mean-squared error. b) Two norm of error.

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
20 DN / 1 HN White Noise	0.04749 1,000	0.003867 1,250	0.04362 1,000
20 DN / 2 HN White Noise	**0.03731** **5,000**	0.003054 5,000	**0.03425** **5,000**
20 DN / 3 HN White Noise	0.06352 1,500	0.003235 1,000	0.05986 1,500
20 DN / 4 HN White Noise	0.05956 2,000	**0.003028** **750**	0.05593 2,000
20 DN / 5 HN White Noise	0.06801 1,750	0.00307 750	0.06401 1,750

Table 4.1 Generalization Metric Results for White Noise

NN with 20 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
20 DN / 1 HN Pulse Train	18.15 500	0.3031 250	17.79 500
20 DN / 2 HN Pulse Train	18.16 750	0.3031 250	17.81 500
20 DN / 3 HN Pulse Train	18.18 500	0.2759 250	17.84 500
20 DN / 4 HN Pulse Train	18.09 500	**0.2735** **250**	17.76 750
20 DN / 5 HN Pulse Train	**18.03** **5,000**	0.2888 250	**17.64** **5,000**

Table 4.2 Generalization Metric Results for Pulse Train

NN with 20 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
30 DN / 1 HN White Noise	4.297 5,000	0.05679 5,000	4.241 5,000
30 DN / 2 HN White Noise	3.027 250	0.03433 250	2.993 250
30 DN / 3 HN White Noise	**2.845** **250**	**0.02387** **250**	**2.821** **250**
30 DN / 4 HN White Noise	3.075 500	0.03124 500	3.044 500
30 DN / 5 HN White Noise	3.18 500	0.02986 500	3.15 500

Table 4.3 Generalization Metric Results for White Noise

NN with 30 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
30 DN / 1 HN Pulse Train	**0.319** **5,000**	**0.02261** **5,000**	**0.2964** **5,000**
30 DN / 2 HN Pulse Train	0.3892 2,000	0.02909 2,500	0.3601 2,000
30 DN / 3 HN Pulse Train	0.3877 5,000	0.02964 5,000	0.3581 5,000
30 DN / 4 HN Pulse Train	0.428 2,250	0.03282 250	0.3944 2,250
30 DN / 5 HN Pulse Train	0.3921 250	0.02915 250	0.3629 250

Table 4.4 Generalization Metric Results for Pulse Train

NN with 30 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
50 DN / 1 HN White Noise	13.85 4,000	0.2061 1,250	13.64 4,000
50 DN / 2 HN White Noise	13.18 250	0.1209 250	13.08 250
50 DN / 3 HN White Noise	5.417 250	0.064 250	5.353 250
50 DN / 4 HN White Noise	3.684 250	0.04343 250	3.64 250
50 DN / 5 HN White Noise	**1.24** **1,500**	**0.03642** **250**	**1.195** **1,500**

Table 4.5 Generalization Metric Results for White Noise

NN with 50 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
50 DN / 1 HN Pulse Train	18.71 250	**0.07698** **250**	18.63 250
50 DN / 2 HN Pulse Train	17.84 4,500	0.08231 4,250	17.76 4,500
50 DN / 3 HN Pulse Train	21.44 250	0.1107 250	21.33 250
50 DN / 4 HN Pulse Train	19.96 5,000	0.1136 5,000	19.84 5,000
50 DN / 5 HN Pulse Train	**12.4** **5,000**	0.08994 5,000	**12.31** **5,000**

Table 4.6 Generalization Metric Results for Pulse Train

NN with 50 Delay Nodes

Evaluation of the Results

In most instances, better generalization results are achieved by stopping the back-propagation algorithm well short of convergence. These findings are anticipated results associated with the use of the cross-validation signals which are a standard tool in statistics. The cross-validation signals are used to validate the model on a different data set than the one used for parameter estimation. Cross-validation is often used to decide when the training of a neural network on a training set should be stopped. In general, too few parameters results in underfitting, where too many parameters results in an overfitting. Good generalization may be achieved even if the neural network has too many parameters provided the training of the network on the training set is stopped at a minimum point of the msee-performance curve. In general, this minimum point is usually found to be well short of convergence of the back-propagation algorithm. Similar findings have been reported by J. Sjöberg and L. Ljung [S2], and S. Shekhar, M. Amin and P. Khandelwal [S1]. We do not observe the expected trends of decreasing bias and increasing variance as the number of hidden units is increased. Rather, we observe both the bias and the variance to be complex functions of the number of training iterations. In view of these results, we note that in this particular design example, we are constructing a model for a simple linear system with a more complex nonlinear model. In this sense, the model is overparameterized to begin with.

It is of interest to note that the bias and variance components of the msee were markedly similar in the manner in which they varied with number of delay nodes and training cycles. The primary observable difference was the magnitude of each

component. The variance component was observed to be the extensive contributor to the msee in all cases. Figure 4.8 illustrates this trend for the 30 delay node network prediction performance on the random amplitude pulse train. This same trend was apparent in each of the design examples. A sharp contrast in prediction capabilities on the random pulse time series is observed between estimators constructed with different sampling times and associated delay nodes. The networks constructed with 20 and 50 delay nodes exhibit high variance when predicting the random amplitude pulse time series. In contrast with these networks, superior prediction performance on the same time series is seen to be exhibited by the 30 delay node network. The 50 delay node network is observed to be a poor predictor on both the random amplitude pulse and the white noise time series. The 20 delay node network exhibits superior prediction performance on the white noise time series, however prediction performance is severely degraded on the random amplitude pulse time series. Based on our results, the 30 delay node network is the best predictor on both the white noise and random amplitude pulse time series. We select a 30 delay node network with 3 hidden nodes trained with 250 training cycles to be the best predictor for our first order system.

In this model validation process, it will be of interest to employ additional measures from initial research applications and compare these results with the measure we have developed. In previous research efforts, three different measures were applied on the estimation error of each cross-validation signal used to evaluate an estimator. Each measure is a designated induced norm, in particular the one norm, two norm and infinity norm. We formally state the definition of these norms in what follows:

Let V represent an arbitrary vector with n components. The one norm is mathematically defined to be

$$\sum_{i=1}^{n} |v_i| . \tag{4.4}$$

The two norm is mathematically defined to be

$$(\sum_{i=1}^{n} |v_i|^2)^{\frac{1}{2}} . \tag{4.5}$$

The infinity norm is mathematically defined to be

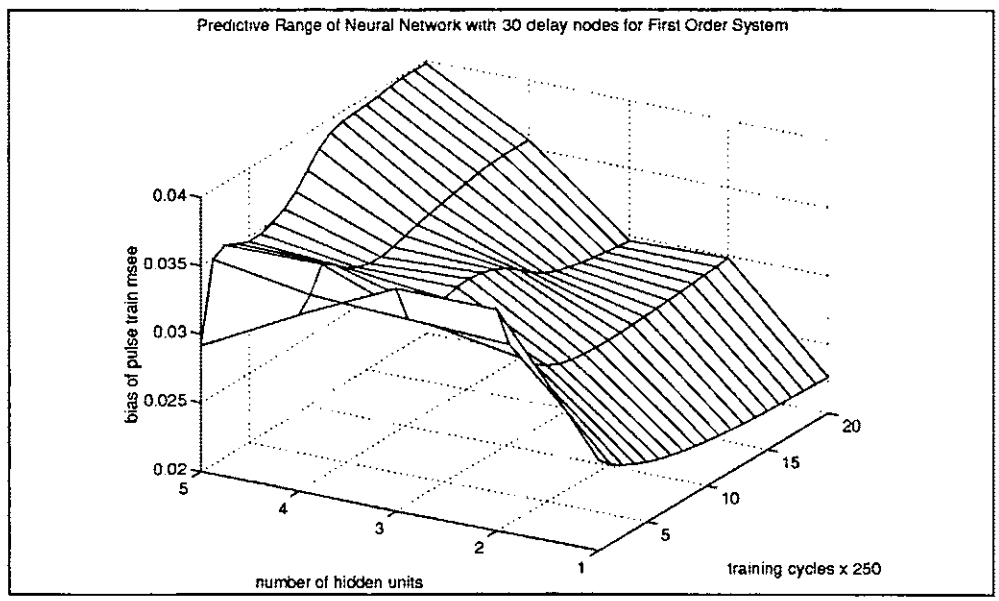
$$\max\ i\ |v_i| . \tag{4.6}$$

Each of the above norms is applied to an error signal defined to be the difference between the actual network output and the desired output for each sample in the validation signal. Scaling is accomplished by applying the same norm to the desired output and dividing the transformed error signal by this result. If we designate ε to represent the error between the actual network output F(**x**,**w**) and the desired output y, then the applied metric in the form of the two norm is defined to be

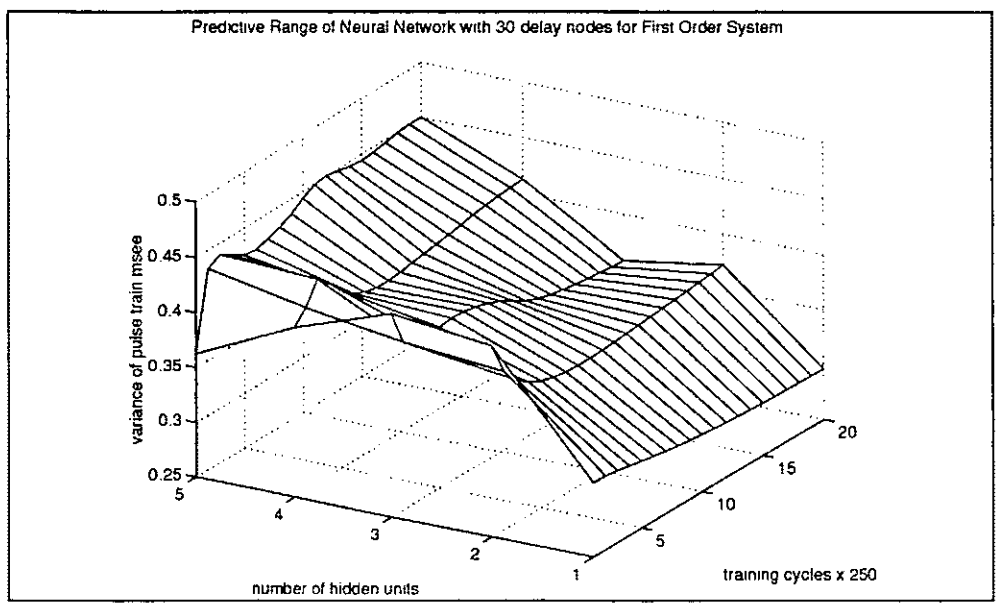
$$\frac{(\sum_{i=1}^{n} |\epsilon_i|^2)^{1/2}}{(\sum_{i=1}^{n} |y_i|^2)^{1/2}} \tag{4.7}$$

where n is the number of samples in the validation signal.

Each of the 300 estimators are evaluated on the validation signals by means of the induced norms. Results from the induced norms are compared with results from the generalization measure we have developed in Tables 4.7 to 4.12.



(a)



(b)

Figure 4.8. Relation between the number of training cycles and hidden nodes and the pulse train estimation error components for 1st order system: a) Bias. b) Variance.

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
20 / 1	0.04749 1,000	0.04519 5,000	0.02193 5,000	0.2088 5,000
20 / 2	**0.03731** **5,000**	0.04427 3,500	0.0196 2,500	0.2088 3,750
20 / 3	0.06352 1,500	0.04414 1,000	0.01869 750	0.209 5,000
20 / 4	0.05956 2,000	0.04369 1,250	0.01732 1,000	0.2088 2,500
20 / 5	0.06801 1,750	**0.04348** **1,500**	**0.01693** **1,000**	0.2088 2,500

Table 4.7 Comparison of Induced Norms and MSEE Metrics

NN with 20 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
20 / 1	18.15 500	**0.07063** **5,000**	0.05736 5,000	0.1372 3,500
20 / 2	18.16 750	0.07066 2,500	0.05702 2,500	0.138 4,000
20 / 3	18.18 500	0.07097 5,000	0.05759 1,000	0.1384 5,000
20 / 4	18.09 500	0.07113 1,500	0.05661 1,250	0.1385 5,000
20 / 5	**18.03** **5,000**	0.07082 5,000	**0.05638** **1,250**	**0.1372** **5,000**

Table 4.8 Comparison of Induced Norms and MSEE Metrics

NN with 20 Delay Nodes

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
30 / 1	4.297 5,000	0.04013 5,000	0.026 1,750	**0.1433** **5,000**
30 / 2	3.027 250	0.04557 5,000	0.02914 1,250	0.1437 5,000
30 / 3	**2.845** **250**	0.04331 250	0.02536 250	0.1456 5,000
30 / 4	3.075 500	0.04156 250	0.02363 750	0.1464 750
30 / 5	3.18 500	**0.03938** **250**	**0.01995** **250**	0.1461 750

Table 4.9 Comparison of Induced Norms and MSEE Metrics

NN with 30 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
30 / 1	**0.319** **5,000**	**0.05756** **5,000**	0.04583 5,000	0.1479 5,000
30 / 2	0.3892 2,000	0.05848 3,000	0.04795 3,750	0.143 4,500
30 / 3	0.3877 5,000	0.0577 5,000	**0.04572** **5,000**	0.1305 5,000
30 / 4	0.428 2,250	0.05909 2,000	0.04843 1,750	0.1235 5,000
30 / 5	0.3921 250	0.06034 2,000	0.04754 1,500	**0.1171** **4,500**

Table 4.10 Comparison of Induced Norms and MSEE Metrics

NN with 30 Delay Nodes

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
50 / 1	13.85 4,000	0.03473 250	0.03241 250	**0.04818** **3,500**
50 / 2	13.18 250	0.04567 250	0.03737 250	0.1048 250
50 / 3	5.417 250	0.03295 250	0.0272 250	0.05305 250
50 / 4	3.684 250	0.03319 250	0.02695 250	0.06373 250
50 / 5	**1.24** **250**	**0.02651** **500**	**0.02126** **500**	0.05085 500

Table 4.11 Comparison of Induced Norms and MSEE Metrics

NN with 50 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
50 / 1	18.71 250	0.04347 4,250	0.03759 1,000	0.0982 3,500
50 / 2	17.84 250	**0.03789** **3,500**	**0.0322** **3,250**	0.0789 3,750
50 / 3	21.44 250	0.0408 4,250	0.03557 3,750	**0.07028** **4,250**
50 / 4	19.96 5,000	0.04441 1,500	0.038 1,500	0.07759 5,000
50 / 5	**12.4** **5,000**	0.04461 1,250	0.03779 1,000	0.07135 2,750

Table 4.12 Comparison of Induced Norms and MSEE Metrics

NN with 50 Delay Nodes

Discussion of Results

20 Delay Node Network

We find the minimum msee on the white noise sequence is achieved with a network configuration of 2 hidden nodes by stopping the training after 5,000 iterations. Both the minimum two and one norm are achieved for a network configuration of 5 hidden nodes. However, the two norm is minimum at 1,500 iterations and the one norm is minimum at 1,000 iterations. The minimum infinite norm is found to be the same for nearly all network configurations, however it is reached for different numbers of training cycles. We note that none of the applied metrics agree with each other.

We find the minimum msee on the random amplitude pulse train is achieved with a network configuration of 5 hidden nodes by stopping the training after 5,000 iterations. The minimum of both the one norm and the infinite norm is also achieved with 5 hidden nodes, however training must be stopped at 1,250 cycles for the one norm and 5,000 cycles for the infinite norm. The two norm is found to be minimum at 5,000 iterations for a network with only 1 hidden node. We note that the msee and infinite norm metric agree with each other.

30 Delay Node Network

We find the minimum msee on the white noise sequence is achieved with a network configuration of 3 hidden nodes by stopping the training after 250 iterations. The minimum of both the one norm and the two norm is achieved with a network configuration of 5 hidden nodes by stopping training at 250 iterations in both instances.

The infinite norm is found to be minimum at 5,000 iterations for a network with only 1 hidden node. We note the agreement of the one and two norm metrics.

We find both the minimum msee and the two norm on the random amplitude pulse train are achieved with a network configurations of 1 hidden node by stopping the training after 5,000 iterations. The one norm is found to be minimum after 5,000 iterations with a network configuration of 3 hidden nodes. The infinite norm is found to be minimum at 4,500 iterations for a network with 5 hidden nodes. We note the agreement of the msee and two norm metrics.

50 Delay Node Network

We find the minimum for the msee on the white noise sequence is achieved with a network configuration of 5 hidden and 5,000 training cycles. Both the one norm and the two norm are found to be minimum with network configurations of 2 hidden nodes. Minimum value for the two norm is achieved after 3,500 iterations. Minimum value for the two norm is achieved after 3,250 iterations. The infinite norm is found to be minimum at 4,250 iterations with a network configuration of 1 hidden node. We find no agreement with any of the applied metrics.

We find the minimums for the msee, the two norm and the one norm on the white noise sequence are each achieved with network configurations of 5 hidden nodes. The msee is minimum at 250 iterations. Both the one norm and the two norm are found to be minimum after 500 iterations. The infinite norm is found to be minimum at 3,500 iterations for a network with 1 hidden node. We note the agreement of the one and two norm metrics.

4.4 Design Example Two

We identify a second order linear system of a low pass nature characterized by the same cutoff frequency as the previously identified system. This dynamical system is represented by the transfer function model:

$$G(s) = \frac{1}{s^2 + 1.4s + 1}. \qquad (4.8)$$

4.4.1 Generation of Training Examples

We generate our training examples in the same manner as previously discussed in design example one. We construct a similar pulse train sequence followed by a Gaussian white noise sequence. We generate three sets of training signals, characterized by three different sampling times of 0.25 s, 0.1538 s and 0.1 s.

4.4.2 Selection of Network Architecture

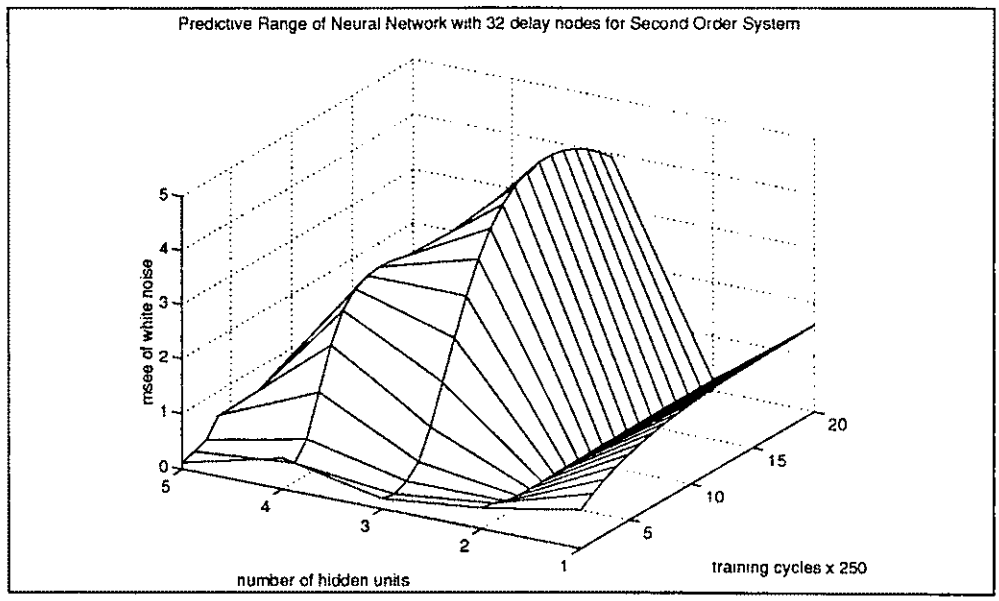
The network architecture is the time-delay neural network. Delay node requirements are established by both sampling time and settling time intervals. The settling time for a step input applied to this system is found to be approximately 8 seconds. We select network architectures based on sequences of 32, 52 and 80 delay nodes. We choose to vary the number of hidden nodes between one and five.

4.4.3 Formation of the Estimators

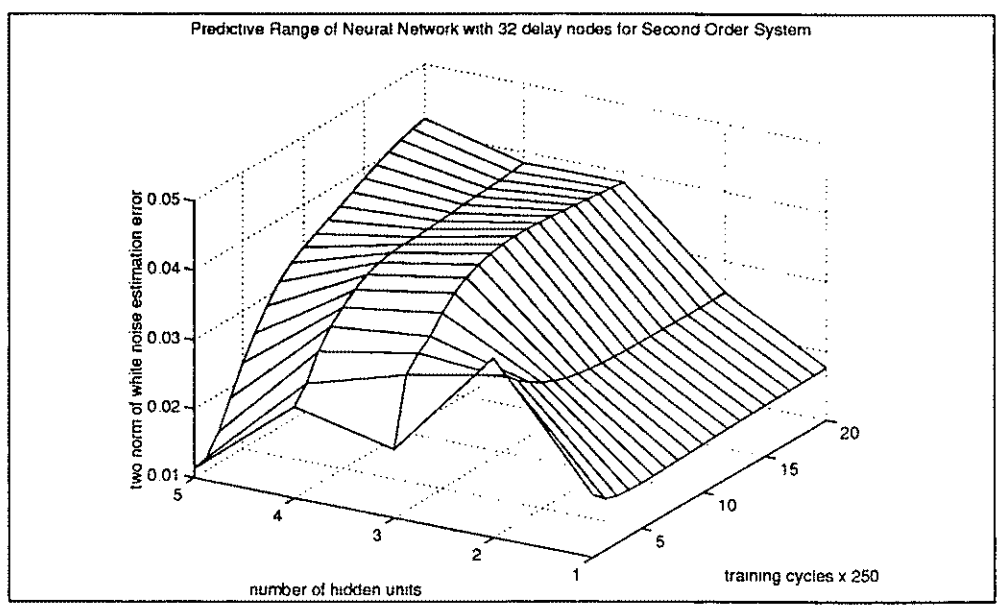
We construct a total of 300 estimators for this experiment, each estimator distinguished by a specific number of delay nodes, hidden units and training cycles. Networks are trained with the back-propagation training algorithm for a total of 5,000 training cycles. Network configurations are saved every 250 training cycles. For each specific network architecture, a total of 20 different estimators are formed and evaluated on the random amplitude pulse train and Gaussian white noise time series. We apply both the generalization metrics and the set of induced norms on the estimation error associated with these validation signals.

4.4.4 Experimental Results

Figures 4.9 - 4.14 illustrate the mean square estimation error and the two norm applied to the estimation error for each of the 15 network architectures, as a function of the number of training cycles and the number of hidden nodes. Each criterion is employed on the estimation error associated with each network's transformation of the validation signals. Tables 4.13 - 4.18 summarize the results for the minimum bias, variance and msee achieved for each of the network configurations and number of training cycles. Tables 4.13 and 4.14 and Figures 4.9 and 4.10 summarize results for the 32 delay node network. Tables 4.15 and 4.16 and Figures 4.11 and 4.12 summarize results for the 52 delay node network. Tables 4.17 and 4.18 and Figures 4.13 and 4.14 summarize results for the 80 delay node network.

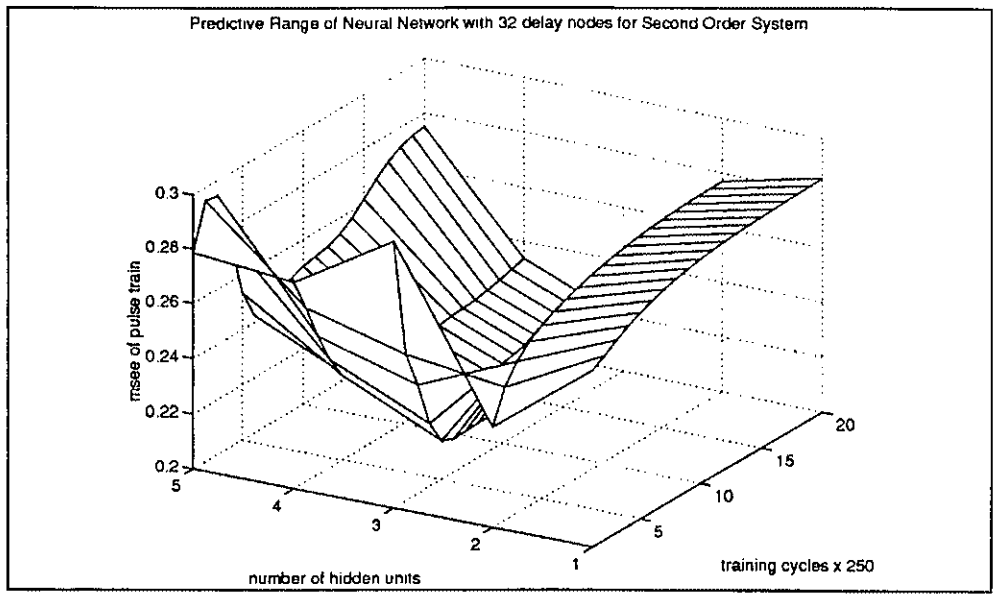


(a)

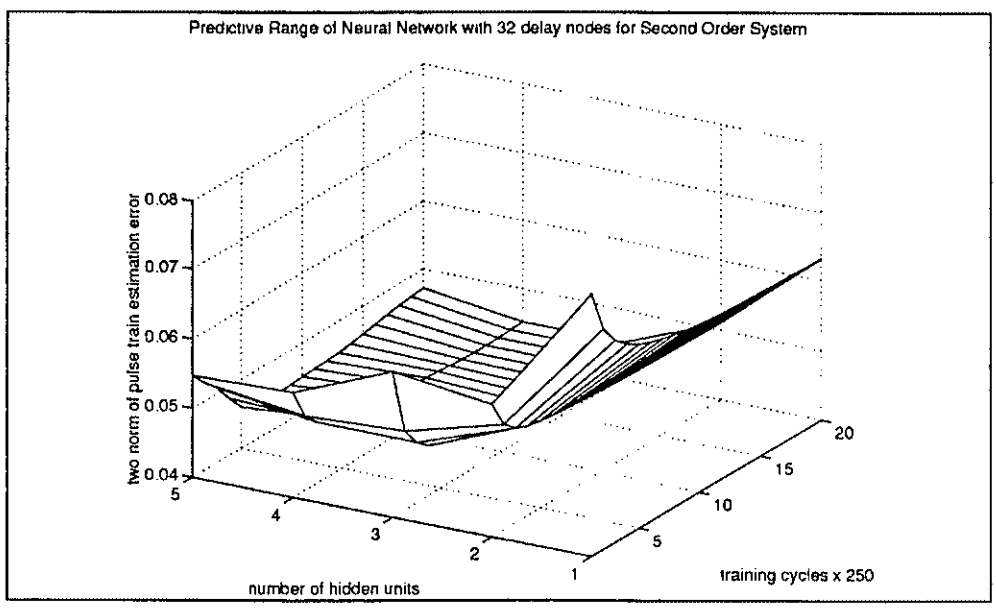


(b)

Figure 4.9. Relation between the number of training cycles and hidden nodes and the white noise estimation error for 2nd order system: a) Mean-squared error. b) Two norm of error.

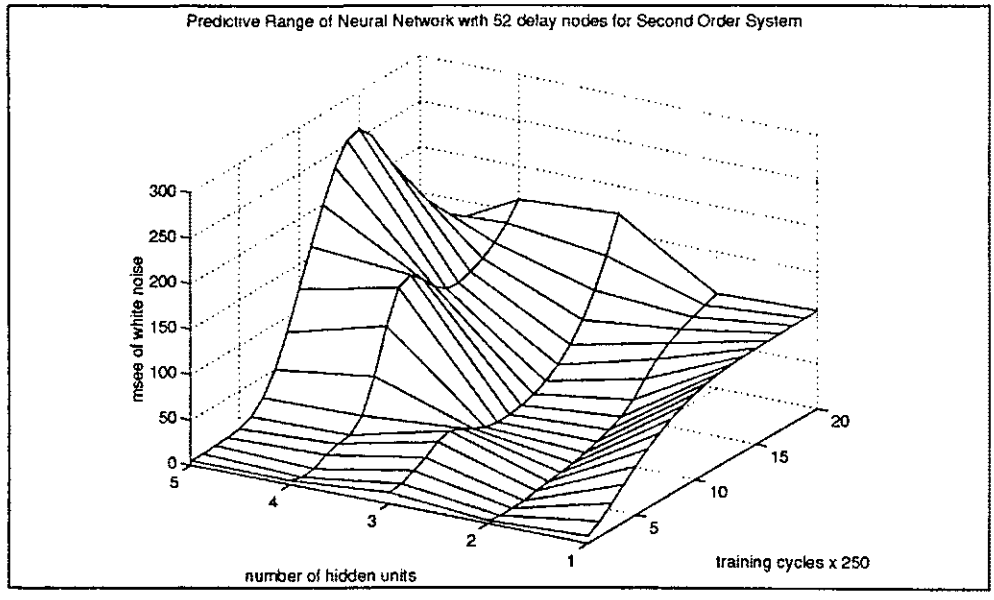


(a)

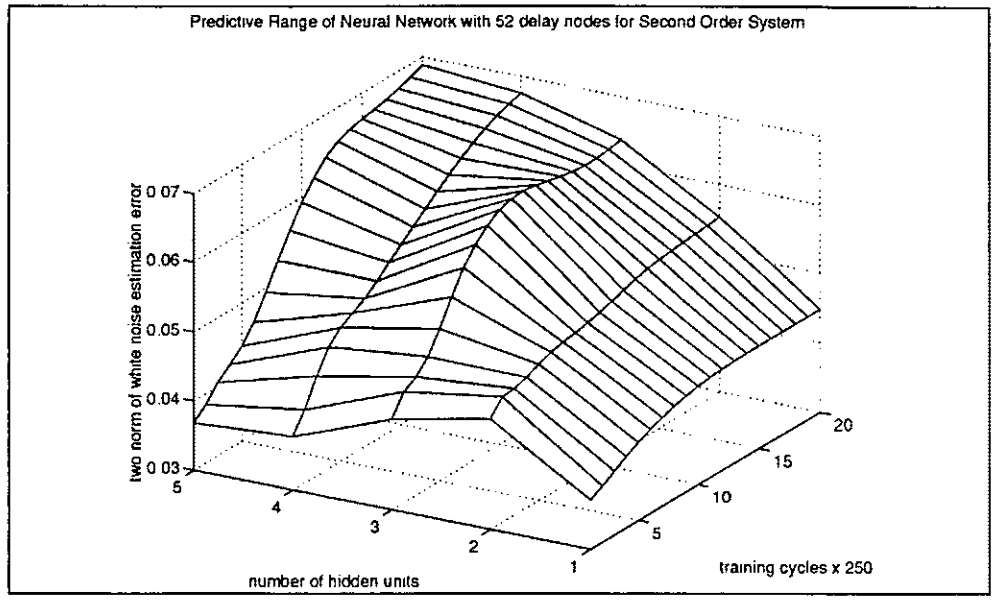


(b)

Figure 4.10. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for 2nd order system: a) Mean-squared error. b) Two norm of error.

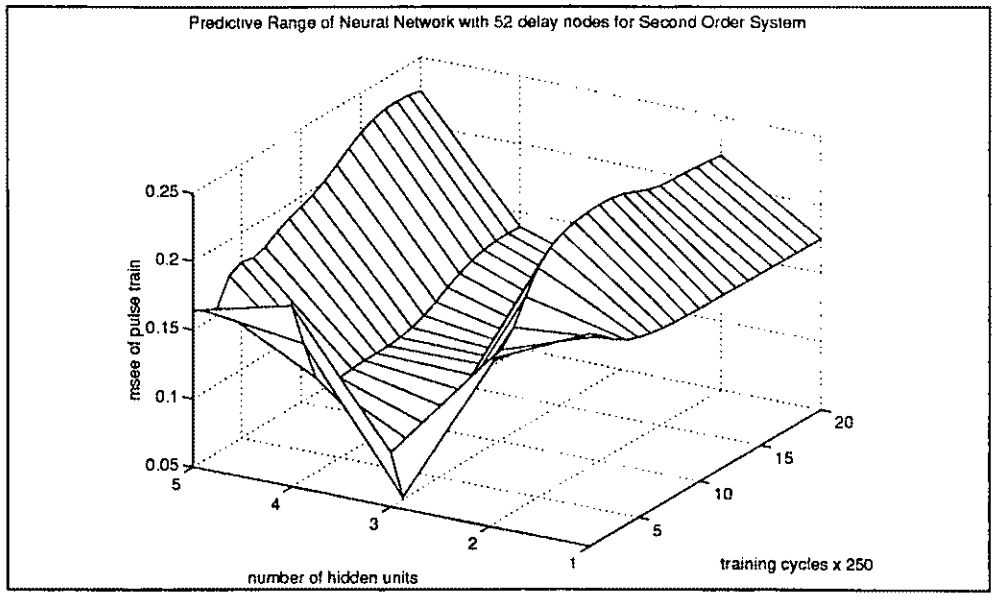


(a)

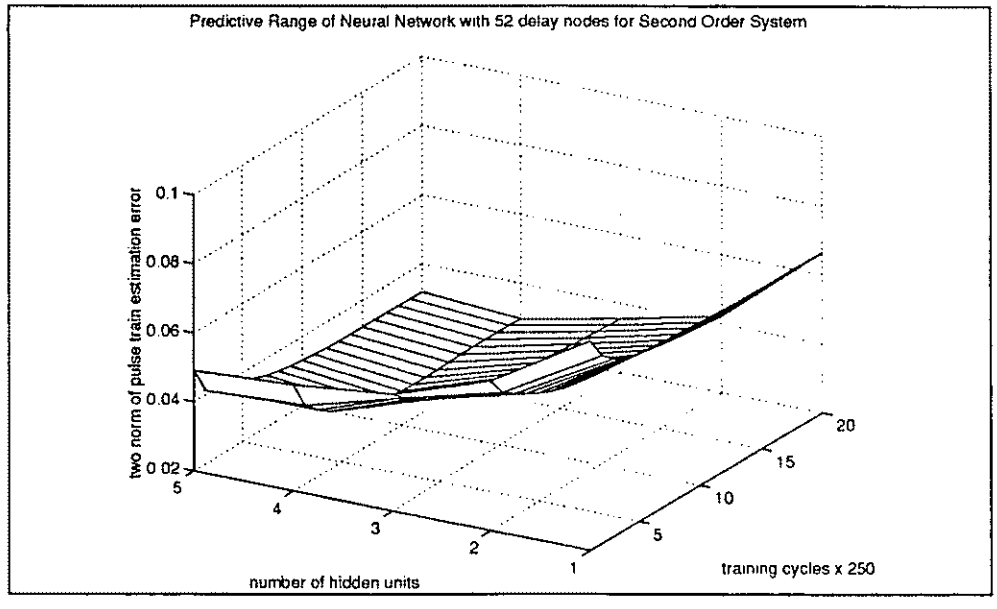


(b)

Figure 4.11. Relation between the number of training cycles and hidden nodes and the white noise estimation error for 2nd order system: a) Mean-squared error. b) Two norm of error.

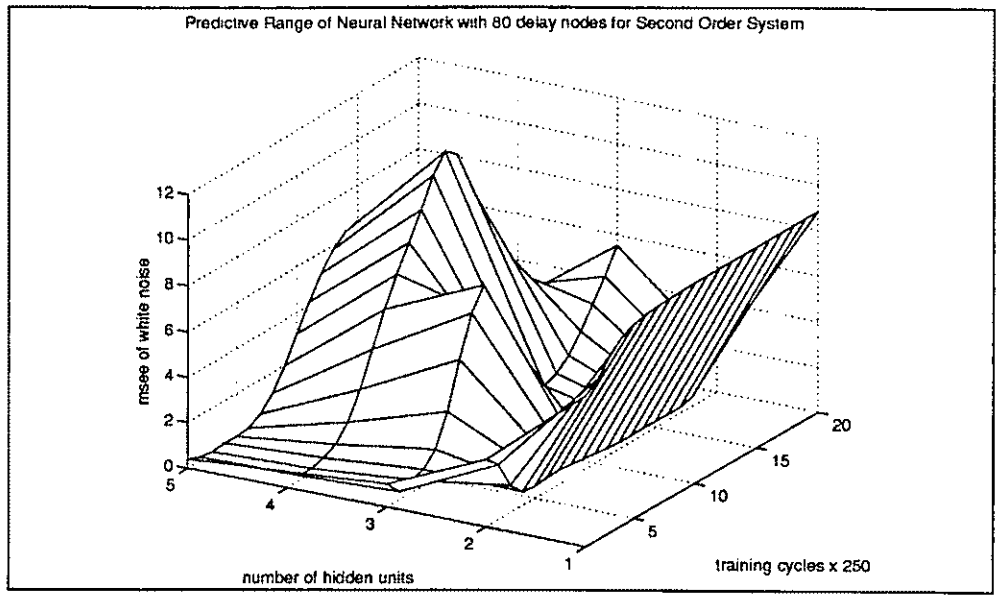


(a)

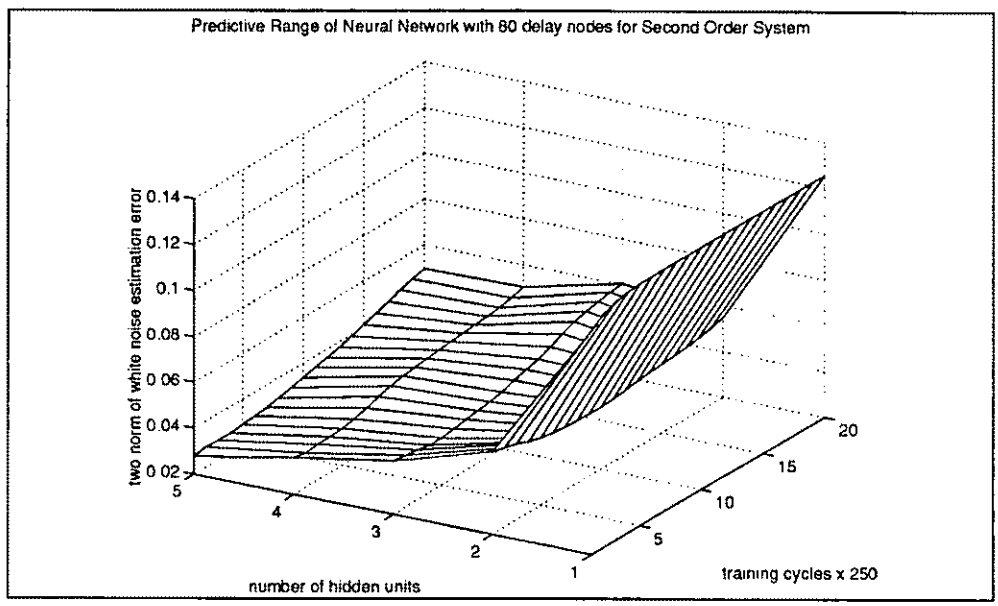


(b)

Figure 4.12. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for 2nd order system: a) Mean-squared error. b) Two norm of error.

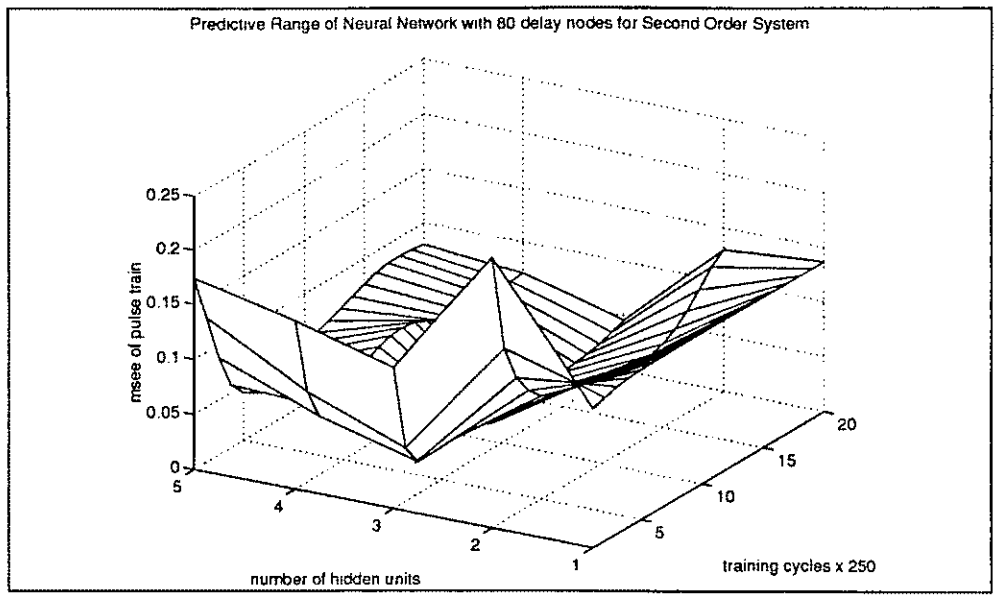


(a)

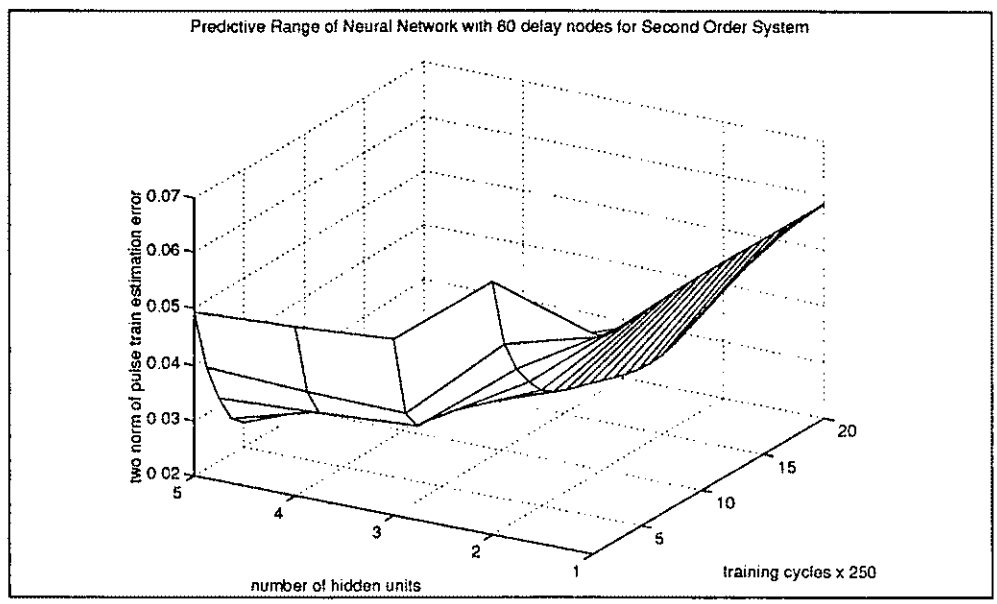


(b)

Figure 4.13. Relation between the number of training cycles and hidden nodes and the white noise estimation error for 2nd order system: a) Mean-squared error. b) Two norm of error.



(a)



(b)

Figure 4.14. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for 2nd order system: a) Mean-squared error. b) Two norm of error.

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
32 DN / 1 HN White Noise	0.7027 250	0.01112 250	0.6916 250
32 DN / 2 HN White Noise	0.1964 1,750	0.006378 5,000	0.1899 1,750
32 DN / 3 HN White Noise	0.1406 500	0.003801 250	0.1353 500
32 DN / 4 HN White Noise	0.3297 500	0.007567 500	0.3221 500
32 DN / 5 HN White Noise	**0.1126** **250**	**0.001715** **250**	**0.1109** **250**

Table 4.13 Generalization Metric Results for White Noise

NN with 32 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
32 DN / 1 HN Pulse Train	0.2646 250	0.02159 1,250	0.2422 250
32 DN / 2 HN Pulse Train	0.2369 250	0.01753 250	0.2193 250
32 DN / 3 HN Pulse Train	**0.2122** **1,500**	**0.01454** **2,500**	**0.1975** **1,500**
32 DN / 4 HN Pulse Train	0.2229 2,250	0.01541 2,000	0.2075 2,250
32 DN / 5 HN Pulse Train	0.241 1,750	0.0163 1,750	0.2247 1,750

Table 4.14 Generalization Metric Results for Pulse Train

NN with 32 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
52 DN / 1 HN White Noise	8.234 250	0.04976 250	8.184 250
52 DN / 2 HN White Noise	0.5646 500	0.01704 750	0.5465 500
52 DN / 3 HN White Noise	12.52 250	0.07367 250	12.44 250
52 DN / 4 HN White Noise	**0.3938** **500**	**0.01166** **500**	**0.3821** **500**
52 DN / 5 HN White Noise	4.625 500	0.03105 500	4.593 500

Table 4.15 Generalization Metric Results for White Noise

NN with 52 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
52 DN / 1 HN Pulse Train	0.1754 5,000	0.01235 5,000	0.1631 5,000
52 DN / 2 HN Pulse Train	0.1712 250	0.009661 5,000	0.1605 250
52 DN / 3 HN Pulse Train	**0.05151** **500**	0.005018 500	**0.04649** **5,000**
52 DN / 4 HN Pulse Train	0.1084 1,250	**0.005003** **2,250**	0.103 1,250
52 DN / 5 HN Pulse Train	0.1554 750	0.007421 750	0.148 750

Table 4.16 Generalization Metric Results for Pulse Train

NN with 52 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
80 DN / 1 HN White Noise	6.6722 250	0.1986 250	6.474 250
80 DN / 2 HN White Noise	0.2569 4,000	0.01669 4,000	0.2376 4,250
80 DN / 3 HN White Noise	0.3774 500	0.01188 500	0.3655 500
80 DN / 4 HN White Noise	0.2217 500	0.008494 500	0.2132 750
80 DN / 5 HN White Noise	**0.07449** **750**	**0.003926** **750**	**0.07056** **750**

Table 4.17 Generalization Metric Results for White Noise

NN with 80 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
80 DN / 1 HN Pulse Train	0.1267 250	0.01618 250	0.1106 250
80 DN / 2 HN Pulse Train	0.07197 3,250	0.004689 3,250	0.06728 3,250
80 DN / 3 HN Pulse Train	**0.02136** **2,500**	**0.00304** **2,750**	**0.01825** **2,500**
80 DN / 4 HN Pulse Train	0.05296 1,000	0.00397 4,500	0.04819 750
80 DN / 5 HN Pulse Train	0.04209 1,500	0.003398 1,500	0.03869 1,500

Table 4.18 Generalization Metric Results for Pulse Train

NN with 80 Delay Nodes

Evaluation of Results

Again, analysis of the experimental results reveal convergence issues to be of a complex nature for feedforward networks learning temporal sequences. Results from these experiments confirm the previous experimental findings in which we noted that in most instances, better generalization results are achieved by stopping the back-propagation algorithm well short of convergence. The bias and variance components of the estimation error are observed to be complex functions of the number of training iterations. Again, we note the variance to be the primary contributor to the estimation error in all cases. We do not observe a sharp contrast in prediction capabilities on the random pulse time series between estimators constructed with different sampling times and associated delay nodes. The best predictor based on these results is seen to be the network constructed with 80 delay nodes and 5 hidden units. The 52 delay node network is observed to be a poor predictor on the white noise time series. The 32 delay node network performs reasonably well on both the white noise and random amplitude pulse time series, however the 80 delay node network with 5 hidden nodes exhibits superior prediction performance on both time series. We note the superior prediction performance of the 80 delay node network on the random amplitude pulse time series to be the result of an extremely low variance. Based on our results, the 80 delay node, 5 hidden node network is the best predictor on both the white noise and random amplitude pulse time series. We select this network with 750 training cycles as the predictor for our second order system.

Again, it will be of interest to employ the previously defined induced norms as measures and compare these results with the measure we have developed. We illustrate the comparison of these results in Tables 4.19 - 4.24.

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
32 / 1	0.7027 250	0.01658 1,250	0.01478 750	**0.02582** **5,000**
32 / 2	0.1964 1,750	0.02552 4,250	0.02057 4,500	0.05952 2,000
32 / 3	0.1406 500	0.01992 250	0.01633 250	0.0517 250
32 / 4	0.3297 500	0.02321 250	0.01944 250	0.0569 250
32 / 5	**0.1126** **250**	**0.0113** **250**	**0.01009** **500**	0.02785 250

Table 4.19 Comparison of Induced Norms and MSEE Metrics

NN with 32 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
32 / 1	0.2646 250	0.06321 5,000	0.05873 5,000	**0.09167** **1,750**
32 / 2	0.2369 250	0.05107 5,000	0.04268 5,000	0.1063 5,000
32 / 3	**0.2122** **1,500**	0.04496 3,250	0.03488 4,000	0.1037 3,500
32 / 4	0.2229 2,250	0.0451 4,250	0.0351 4,000	0.1016 4,500
32 / 5	0.241 1,750	**0.04459** **1,750**	**0.03448** **2,000**	0.09229 250

Table 4.20 Comparison of Induced Norms and MSEE Metrics

NN with 32 Delay Nodes

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
52 / 1	8.234 250	0.0372 250	0.02196 250	0.09654 250
52 / 2	0.5646 500	0.04608 250	0.03126 250	0.09861 2,250
52 / 3	12.52 250	0.04314 250	0.02926 250	0.09775 250
52 / 4	**0.3938** **500**	0.03777 250	0.02258 250	0.09614 250
52 / 5	4.625 500	**0.03684** **250**	**0.02141** **250**	**0.096** **250**

Table 4.21 Comparison of Induced Norms and MSEE Metrics

NN with 52 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
52 / 1	0.1754 5,000	0.06654 5,000	0.0604 5,000	0.0866 5,000
52 / 2	0.1712 250	0.0423 5,000	0.03511 5,000	0.08447 5,000
52 / 3	**0.05151** **500**	0.03563 5,000	0.02789 5,000	0.0794 750
52 / 4	0.1084 1,250	**0.02965** **3,000**	**0.0236** **3,000**	**0.06046** **3,000**
52 / 5	0.1554 750	0.03192 5,000	0.02526 5,000	0.06048 3,250

Table 4.22 Comparison of Induced Norms and MSEE Metrics

NN with 52 Delay Nodes

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
80 / 1	6.672 250	0.1174 250	0.1144 250	0.142 250
80 / 2	0.2569 4,000	0.04865 2,500	0.04397 2,250	0.07392 4,250
80 / 3	0.3774 500	0.04045 1,250	0.03842 1,250	0.05929 250
80 / 4	0.2217 500	0.0348 500	0.0331 1,500	0.04767 250
80 / 5	**0.07449** **750**	**0.02775** **250**	**0.02724** **1,000**	**0.03464** **500**

Table 4.23 Comparison of Induced Norms and MSEE Metrics

NN with 80 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
80 / 1	0.1267 250	0.05808 750	0.05348 500	0.09791 500
80 / 2	0.07197 3,250	0.03404 3,500	0.03091 3,500	0.05774 3,750
80 / 3	**0.02136** **2,500**	0.0273 4,250	0.02452 4,250	0.04815 3,250
80 / 4	0.05296 1,000	0.02215 5,000	**0.02133** **5,000**	0.04579 5,000
80 / 5	0.04209 1,500	**0.02181** **5,000**	0.02151 5,000	**0.03806** **1,250**

Table 4.24 Comparison of Induced Norms and MSEE Metrics

NN with 80 Delay Nodes

Discussion of Results

32 Delay Node Network

We find the minimums for the msee and the one and two norms on the white noise sequence are achieved with network configurations of 5 hidden nodes. Both the msee and the two norm are found to be minimum after 250 iterations. Minimum value for the one norm is achieved after 500 iterations. The infinite norm is found to be minimum at 5,000 iterations with a network configuration of 1 hidden node. We note the agreement of the msee and two norm metrics.

We find the minimum for the msee on the random amplitude pulse train sequence is achieved with a network configuration of 3 hidden nodes and 1,500 iterations. Both the one and two norms are found to be minimum with network configurations of 5 hidden nodes. Minimum value for the two norm occurs at 1,750 iterations. Minimum value for the one norm is achieved after 2,000 iterations. The infinite norm is found to be minimum at 1,750 iterations with a network configuration of 1 hidden node. We find no agreement between any of the applied metrics.

52 Delay Node Network

We find the minimum for the msee on the white noise sequence is achieved with a network configuration of 4 hidden nodes and 500 iterations. All three induced norms are found to be minimum with network configurations of 5 hidden nodes and 250 iterations. We note the agreement of all three of the induced norm metrics.

We find the minimum for the msee on the random amplitude pulse train sequence is achieved with a network configuration of 3 hidden nodes and 500 iterations. Both the

one, two and infinite norms are found to be minimum with network configurations of 4 hidden nodes and 3,000 iterations. We find agreement between all three of the induced norms.

80 Delay Node Network

We find the minimum for each of the metrics on the white noise sequence is achieved with a network configuration of 5 hidden nodes. Minimum msee is reached after 750 iterations. The minimum two norm occurs after 250 iterations. The one norm minimum occurs after 1,000 iterations and the infinite norm minimum occurs after 500 iterations. We use this example to note the complexity of convergence issues involved with training multilayer feedforward networks. Each of the minimums for the applied metrics are attained with the same number of hidden nodes, but different numbers of iterations. We find the minimum for the msee on the random amplitude pulse train sequence is achieved with a network configuration of 3 hidden nodes and 2,500 iterations. Both the one and infinite norms are found to be minimum with network configurations of 5 hidden nodes. The one norm is minimum after 5,000 iterations and the infinite norm is minimum after 1,250 iterations. The two norm realizes a minimum with a network configuration of 5 hidden nodes and 5,000 iterations. We find no agreement between any of the applied metrics.

4.5 Design Example Three

We identify a nonlinear system which is known to be descriptive of a mass-spring-damper with a stiffening spring. Linearization of this system about a steady state

operating point of (0,0) results in the same cutoff frequency as the previously identified linear systems, one radian per second. This dynamical system is represented by the differential equation:

$$\ddot{y} + \dot{y} + y + y^3 = u \tag{4.9}$$

The above differential equation for the system can be rewritten into a set of first-order differential equations, called state equations. The state-variable description for the nonlinear system is

$$\dot{x}_1 = x_2 \tag{4.10}$$

$$\dot{x}_2 = -x_2 - x_1 - x_1^3 + u \tag{4.11}$$

The linearized system around the origin is described by the transfer function model

$$\frac{1}{s^2 + s + 1} \tag{4.12}$$

4.5.1 Generation of Training Examples

We generate our training examples in the same manner as the previous design examples. We determine the settling time of the linearized system to a step input to be approximately 10 seconds. We construct a pulse train sequence followed by a Gaussian white noise sequence. The training sequences are generated by simulating the nonlinear system with computer software. We generate three sets of training signals, characterized by three different sampling times of 0.2 s, 0.1493 s and 0.1 s.

4.5.2 Selection of Network Architecture

The network architecture is the time-delay neural network. Delay node requirements are established by both sampling time and settling time intervals and we select network architectures based on sequences of 50, 67 and 100 delay nodes. We choose to vary the number of hidden nodes between one and five.

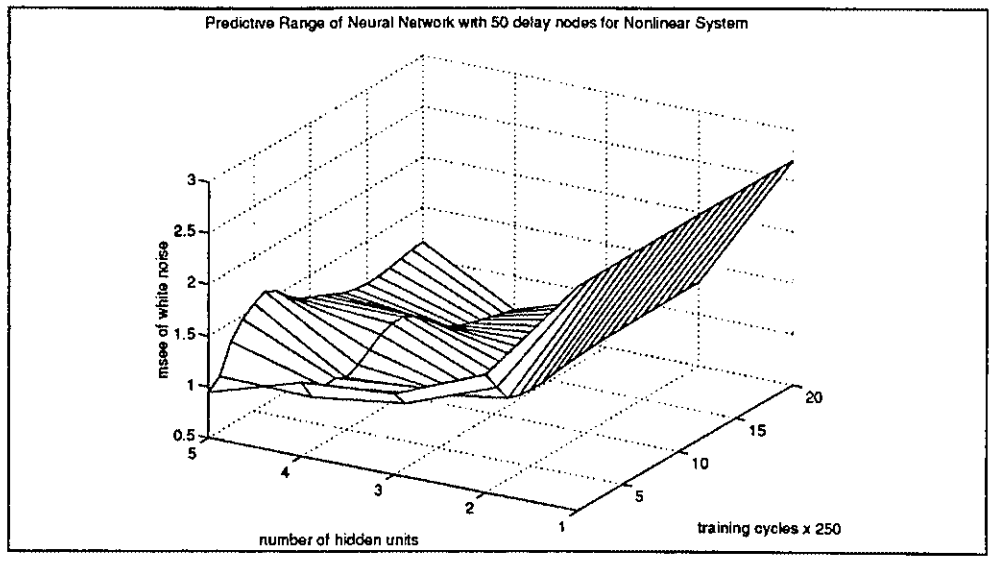
4.5.3 Formation of the Estimators

We construct a total of 300 estimators for this experiment, each estimator distinguished by a specific number of delay nodes, hidden units and training cycles. Networks are trained with the back-propagation training algorithm for a total of 5,000 training cycles. Network configurations are saved every 250 training cycles. For each specific network architecture, a total of 20 different estimators are formed and evaluated on the random amplitude pulse train and Gaussian white noise time series. We apply both the generalization metrics and the set of induced norms on the estimation errors associated with these validation signals.

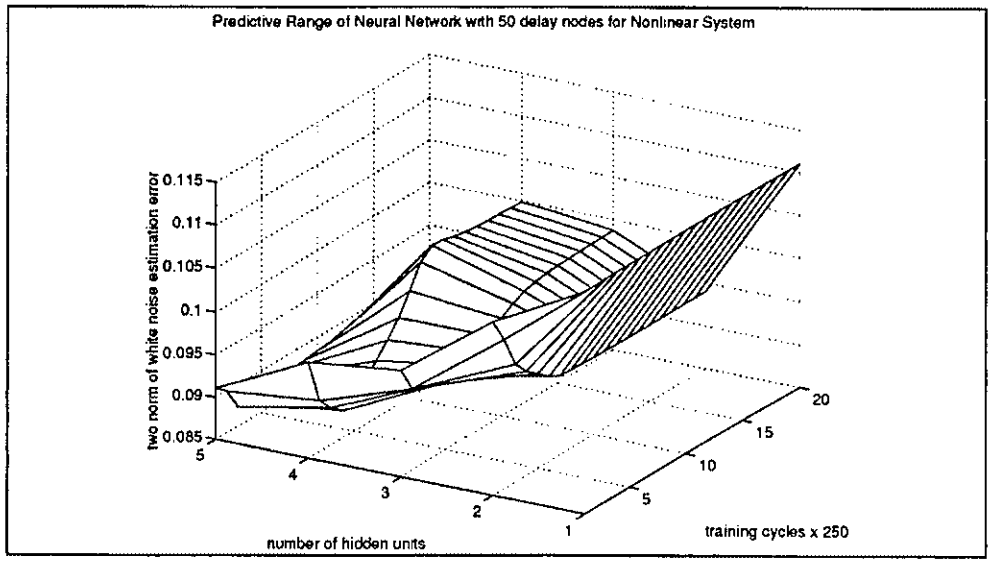
4.5.4 Experimental Results

Figures 4.15 - 4.20 illustrate the mean square estimation error and the two norm applied to the estimation error for each of the 15 network architectures, as a function of the number of training cycles and the number of hidden nodes. Each criterion is employed on the estimation error associated with each network's transformation of the validation signals. Tables 4.25 - 4.30 summarize the results for the minimum bias,

variance and msee achieved for each of the network configurations and number of training cycles. Tables 4.25 and 4.26 and Figures 4.15 and 4.16 summarize results for the 50 delay node network. Tables 4.27 and 4.28 and Figures 4.17 and 4.18 summarize results for the 67 delay node network. Tables 4.29 and 4.30 and Figures 4.19 and 4.20 summarize results for the 100 delay node network.



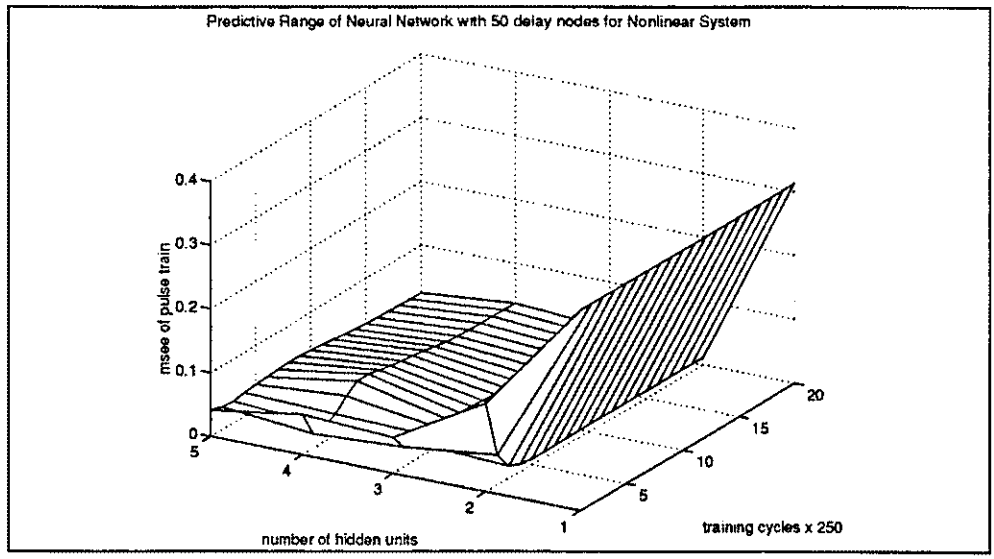
(a)



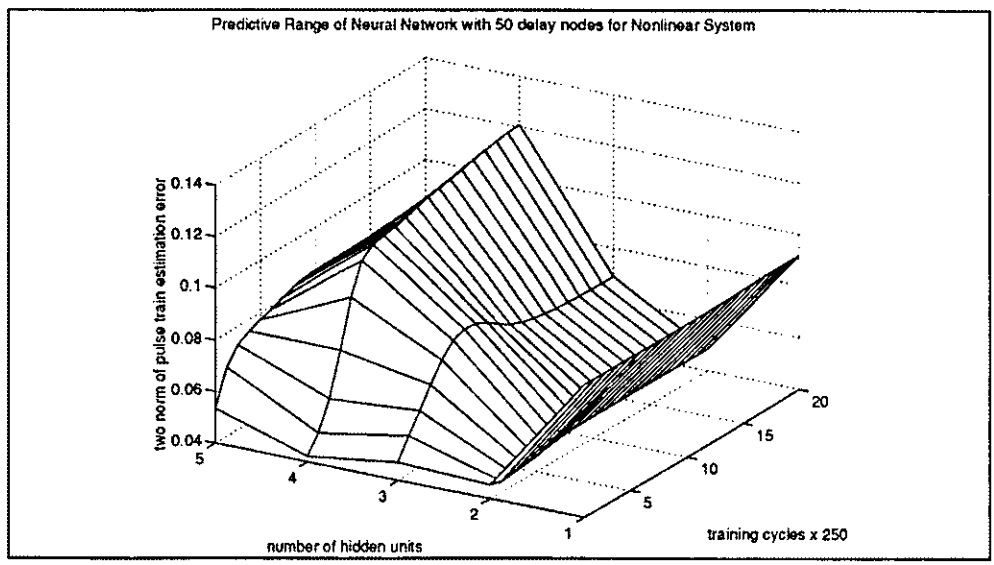
(b)

Figure 4.15. Relation between the number of training cycles and hidden nodes and the white noise estimation error for the nonlinear system:

a) Mean-squared error. b) Two norm of error.



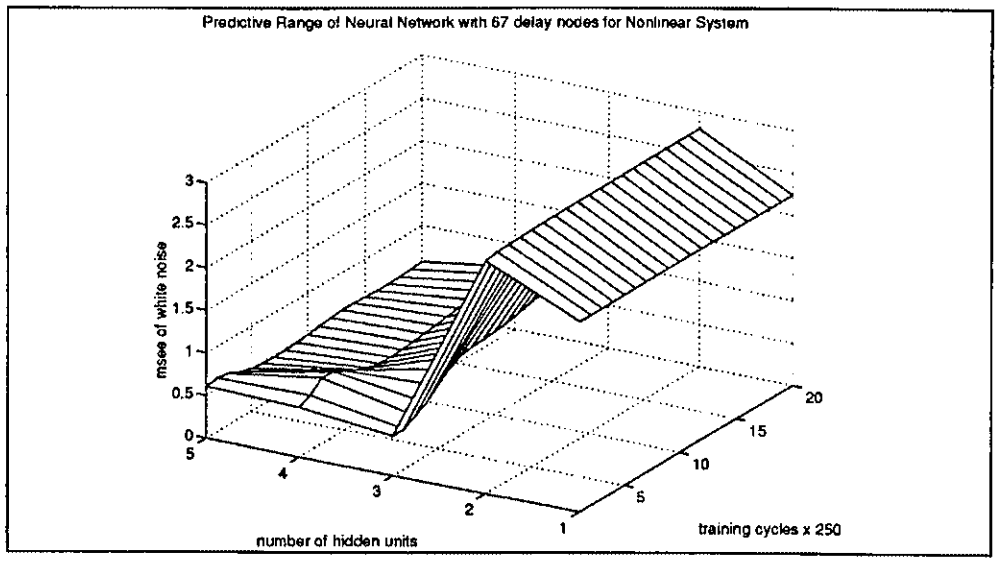
(a)



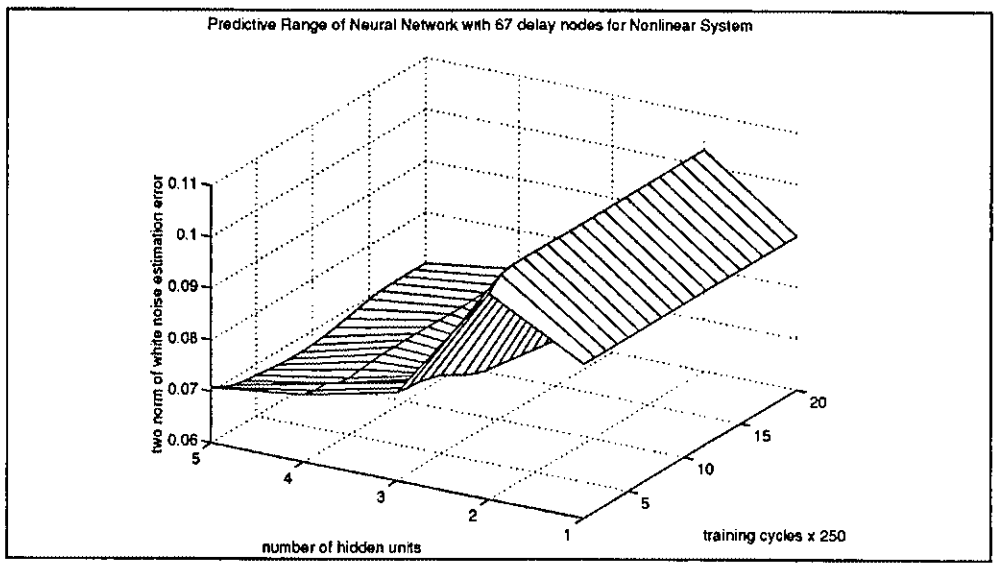
(b)

Figure 4.16. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for the nonlinear system:

a) Mean-squared error. b) Two norm of error.



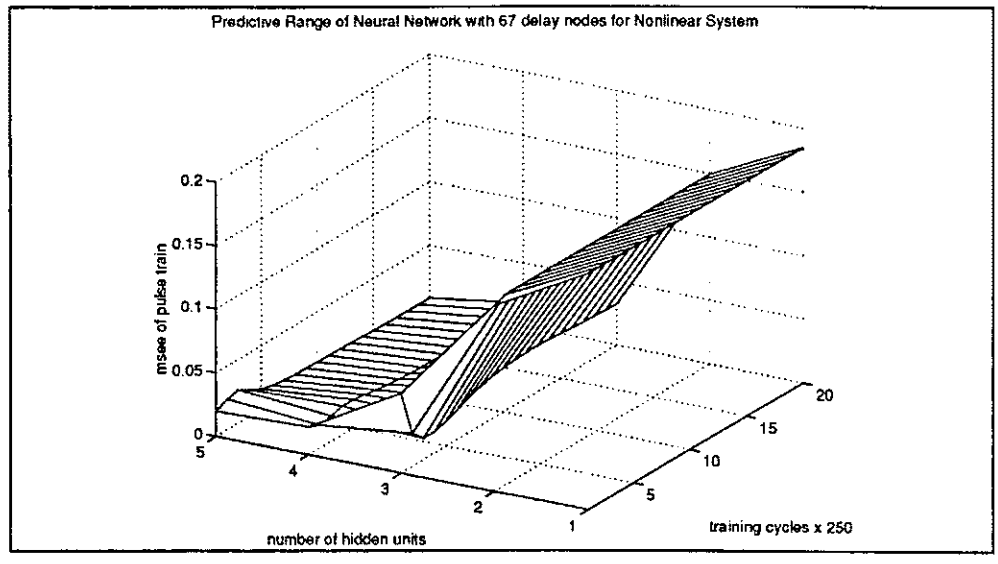
(a)



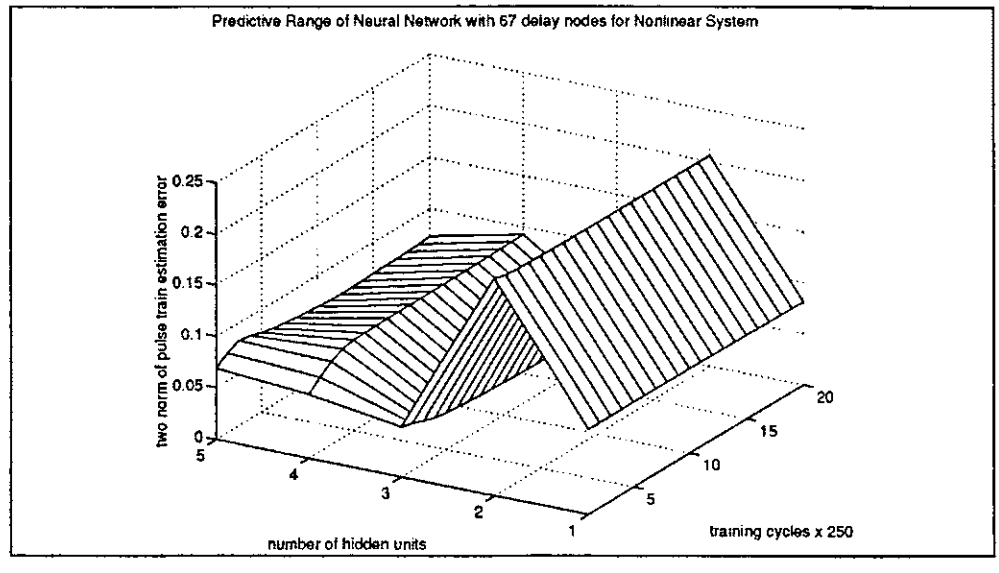
(b)

Figure 4.17. Relation between the number of training cycles and hidden nodes and the white noise estimation error for the nonlinear system:

a) Mean-squared error. b) Two norm of error.



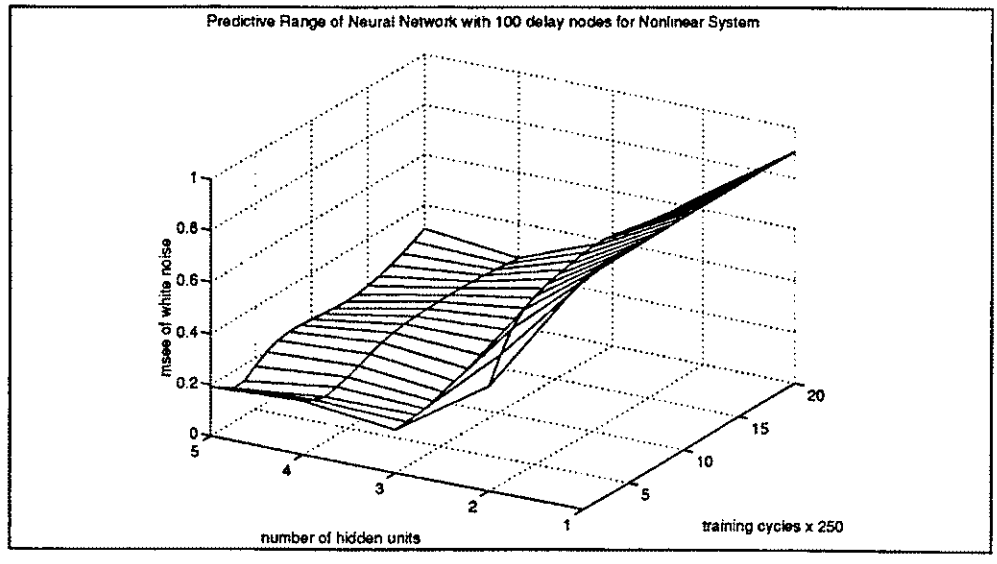
(a)



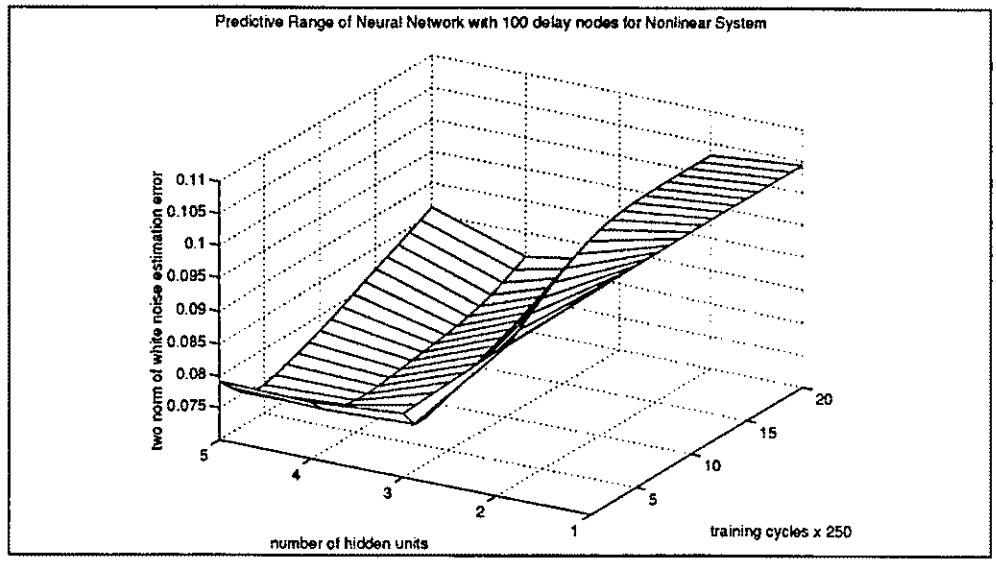
(b)

Figure 4.18. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for the nonlinear system:

a) Mean-squared error. b) Two norm of error.



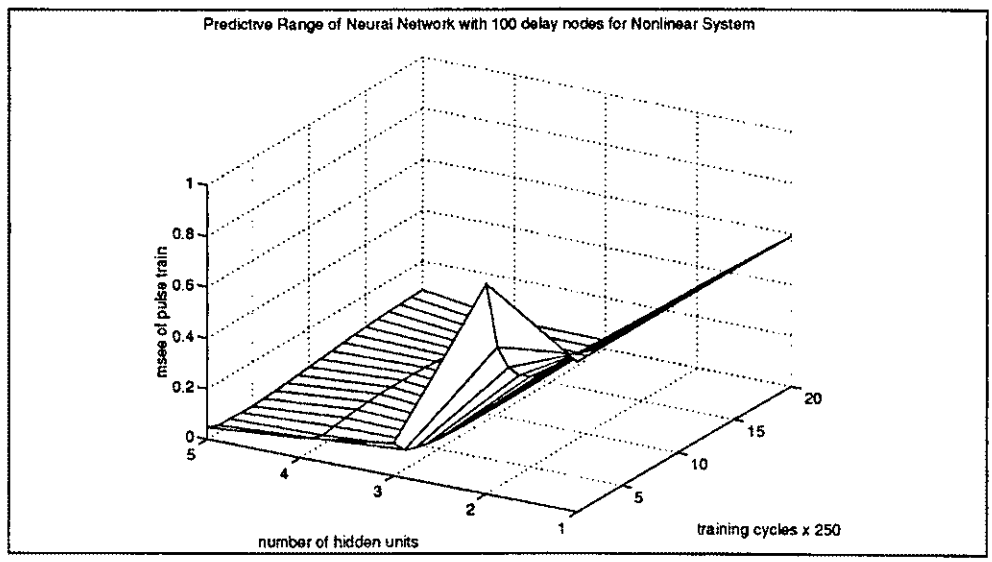
(a)



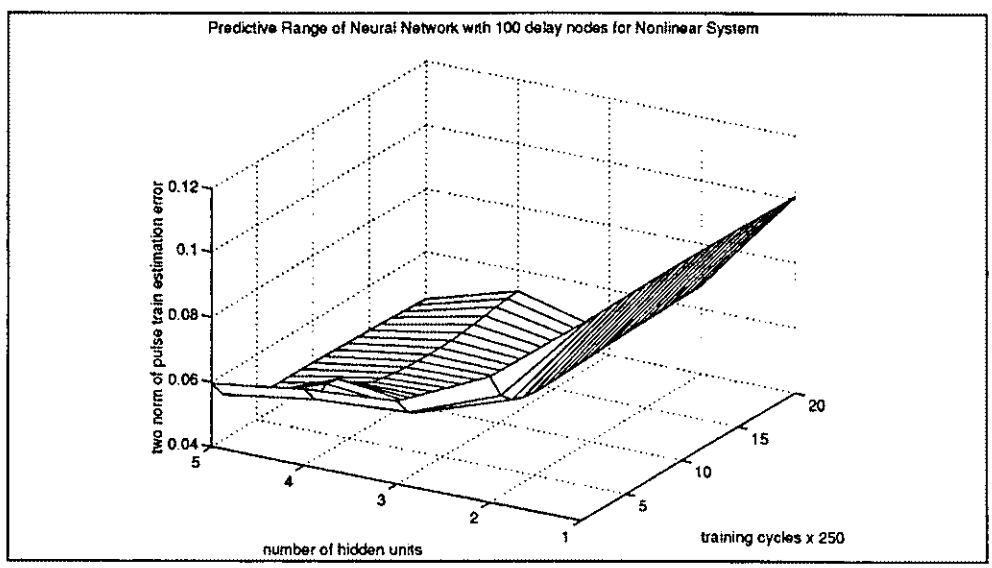
(b)

Figure 4.19. Relation between the number of training cycles and hidden nodes and the white noise estimation error for the nonlinear system:

a) Mean-squared error. b) Two norm of error.



(a)



(b)

Figure 4.20. Relation between the number of training cycles and hidden nodes and the pulse train estimation error for the nonlinear system:

a) Mean-squared error. b) Two norm of error.

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
50 DN / 1 HN White Noise	2.697302 250	0.190284 250	2.507018 250
50 DN / 2 HN White Noise	1.245214 1,250	0.099515 2,000	1.133632 1,250
50 DN / 3 HN White Noise	0.890120 2,000	0.083625 1,500	0.802446 2,000
50 DN / 4 HN White Noise	**0.676680** **5,000**	**0.079358** **5,000**	**0.597321** **5,000**
50 DN / 5 HN White Noise	0.945837 250	0.084542 4,000	0.852490 250

Table 4.25 Generalization Metric Results for White Noise

NN with 50 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
50 DN / 1 HN Pulse Train	0.313871 4,500	0.022002 250	0.291842 4,500
50 DN / 2 HN Pulse Train	**0.009201** **1,500**	**0.002555** **1,500**	**0.006645** **1,500**
50 DN / 3 HN Pulse Train	0.026962 5,000	0.004113 500	0.020074 5,000
50 DN / 4 HN Pulse Train	0.022647 500	0.003244 500	0.019403 500
50 DN / 5 HN Pulse Train	0.024752 5,000	0.004187 250	0.019906 5,000

Table 4.26 Generalization Metric Results for Pulse Train

NN with 50 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
67 DN / 1 HN White Noise	2.238605 500	0.163686 500	2.074918 500
67 DN / 2 HN White Noise	2.735356 250	0.167542 250	2.567814 250
67 DN / 3 HN White Noise	**0.463743** **250**	0.046313 500	**0.417036** **250**
67 DN / 4 HN White Noise	0.542965 2,250	0.039287 2,000	0.502786 2,250
67 DN / 5 HN White Noise	0.480696 1,750	**0.036605** **1,750**	0.444090 1,750

Table 4.27 Generalization Metric Results for White Noise

NN with 67 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
67 DN / 1 HN Pulse Train	0.185113 3,500	0.011136 1,000	0.173976 3,500
67 DN / 2 HN Pulse Train	0.145332 250	0.028248 4,500	0.116908 250
67 DN / 3 HN Pulse Train	0.016642 750	**0.001919** **1,250**	0.014540 750
67 DN / 4 HN Pulse Train	0.015356 3,250	0.003248 250	0.011322 3,500
67 DN / 5 HN Pulse Train	**0.008160** **4,500**	0.002449 4,500	**0.005703** **4,750**

Table 4.28 Generalization Metric Results for Pulse Train

NN with 67 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
100 DN / 1 HN White Noise	0.905012 2,000	0.097168 1,500	0.807843 2,000
100 DN / 2 HN White Noise	0.412925 250	0.044621 250	0.368303 250
100 DN / 3 HN White Noise	0.169524 1,250	0.021505 750	0.147024 1,250
100 DN / 4 HN White Noise	0.174784 750	0.019585 750	0.155199 750
100 DN / 5 HN White Noise	**0.134069** **750**	**0.018259** **750**	**0.115810** **750**

Table 4.29 Generalization Metric Results for White Noise

NN with 100 Delay Nodes

NN Architecture Validation Signal	MSEE Tr Cy	BIAS Tr Cy	VARIANCE Tr Cy
100 DN / 1 HN Pulse Train	0.591884 2,500	0.018382 2,500	0.573501 2,500
100 DN / 2 HN Pulse Train	0.319119 5,000	0.009466 5,000	0.309653 5,000
100 DN / 3 HN Pulse Train	0.031363 2,750	0.002430 5,000	0.028911 2,750
100 DN / 4 HN Pulse Train	0.033699 5,000	0.002111 5,000	0.031587 5,000
100 DN / 5 HN Pulse Train	**0.026274** **1,250**	**0.001820** **1,500**	**0.024432** **1,250**

Table 4.30 Generalization Metric Results for Pulse Train

NN with 100 Delay Nodes

Evaluation of Results

Results from these experiments confirm the previous experimental findings in which we noted that in most instances, better generalization results are achieved by stopping the back-propagation algorithm well short of convergence. The bias and variance components of the estimation error are again observed to be complex functions of the number of training iterations. Again, we note the variance to be the primary contributor to the estimation error in all cases. We note in this design example that the bias does tend to decrease as the number of delay nodes is increased. We expect the bias to decrease as the number of coefficients is increased based on the theory. These results confirm our expectations. It is worth noting that in this particular design example, we are constructing a model for a nonlinear system. Neural networks are most appropriate as models for nonlinear systems. The best predictor based on the bias and variance results is seen to be the network constructed with 100 delay nodes and 5 hidden units. We select this network configuration with 750 training cycles to be the predictor for our nonlinear system. Prediction performance improves for all of the networks as the number of hidden units is increased.

We employ the previously defined induced norms as measures and compare these results with the measure we have developed. We illustrate the comparison of these results in Tables 4.31 - 4.36.

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
50 / 1	2.697302 250	0.111045 2,000	0.107966 250	0.189743 2,500
50 / 2	1.245214 1,250	0.094021 5,000	0.080612 2,000	0.180073 5,000
50 / 3	0.890120 2,000	0.092946 1,250	0.080603 1,250	0.174739 5,000
50 / 4	**0.676680** **5,000**	0.088218 1,000	0.075109 1,000	0.170436 1,000
50 / 5	0.945837 250	**0.087179** **1,000**	**0.073978** **750**	**0.169419** **5,000**

Table 4.31 Comparison of Induced Norms and MSEE Metrics

NN with 50 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
50 / 1	0.313871 4,500	0.091658 250	0.076988 250	0.178367 2,250
50 / 2	**0.009201** **1,500**	0.044215 500	0.035367 1,000	0.098495 1,000
50 / 3	0.026962 5,000	0.047023 250	0.036800 250	0.108922 250
50 / 4	0.022648 500	**0.042156** **250**	**0.331940** **250**	**0.097441** **500**
50 / 5	0.024752 5,000	0.053464 250	0.039993 250	0.113426 250

Table 4.32 Comparison of Induced Norms and MSEE Metrics

NN with 50 Delay Nodes

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
67 / 1	2.238605 500	0.089868 1,750	0.090087 1,750	0.115051 500
67 / 2	2.735356 250	0.099943 250	0.096896 250	0.155250 250
67 / 3	**0.463743** **250**	0.067944 5,000	0.063001 5,000	**0.102250** **5,000**
67 / 4	0.542965 2,250	0.071036 1,250	0.063528 3,500	0.105574 4,000
67 / 5	0.480696 1,750	**0.067177** **2,250**	**0.059506** **2,500**	0.105073 5,000

Table 4.33 Comparison of Induced Norms and MSEE Metrics

NN with 67 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
67 / 1	0.185113 3,500	0.081514 250	0.061360 1,500	0.165441 1,500
67 / 2	0.145332 250	0.206008 4,500	0.146811 3,250	0.332226 5,000
67 / 3	0.016642 750	**0.028674** **5,000**	**0.023492** **5,000**	**0.060085** **5,000**
67 / 4	0.015356 3,250	0.061306 250	0.041822 250	0.134689 500
67 / 5	**0.008160** **4,500**	0.068091 3,250	0.045331 3,250	0.130127 250

Table 4.34 Comparison of Induced Norms and MSEE Metrics

NN with 67 Delay Nodes

Validation Signal: Gaussian White Noise Sequence				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
100 / 1	0.905012 2,000	0.104336 2,500	0.102701 2,500	0.121494 750
100 / 2	0.412925 250	0.092526 500	0.082033 250	**0.112512** **750**
100 / 3	0.169524 1,250	0.077213 750	0.063169 750	0.112799 250
100 / 4	0.174784 750	0.074983 1,500	0.060949 750	0.116213 250
100 / 5	**0.134069** **750**	**0.074427** **1,000**	**0.059472** **1,000**	0.119720 250

Table 4.35 Comparison of Induced Norms and MSEE Metrics

NN with 100 Delay Nodes

Validation Signal: Random Amplitude Pulse Train				
DN / HN	MSEE Tr Cy	2 Norm Tr Cy	1 Norm Tr Cy	Inf Norm Tr Cy
100 / 1	0.591884 10	0.100891 15	0.070416 15	0.187374 2
100 / 2	0.319119 5,000	0.065673 1,500	0.048360 1,250	0.188750 5,000
100 / 3	0.031363 2,750	0.046593 5,000	0.035257 5,000	0.100570 5,000
100 / 4	0.033699 5,000	0.051058 3,250	0.033114 5,000	0.132229 2,500
100 / 5	**0.026274** **1,250**	**0.045587** **5,000**	**0.030022** **2,750**	**0.098618** **5,000**

Table 4.36 Comparison of Induced Norms and MSEE Metrics

NN with 100 Delay Nodes

Discussion of Results

50 Delay Node Network

We find the minimums for the one, two and infinite norms on the white noise sequence are achieved with network configurations of 5 hidden nodes. The two norm is found to be minimum after 1,000 iterations. Minimum value for the one norm is achieved after 750 iterations and the infinite norm is found to be minimum at 5,000 iterations. Minimum msee is achieved with a network configuration of 4 hidden nodes and 5,000 iterations. We find no agreement between any of the applied metrics.

We find the minimum for the msee on the random amplitude pulse train sequence is achieved with a network configuration of 2 hidden nodes and 1,500 iterations. Each of the induced norms are found to be minimum with network configurations of 4 hidden nodes. Minimum value for the one and two norm occurs at 250 iterations. Minimum value for the infinite norm is achieved after 500 iterations. We note the agreement between the one norm and the two norm.

67 Delay Node Network

We find the minimum for the msee on the white noise sequence is achieved with a network configuration of 3 hidden nodes and 250 iterations. Both the one and the two norms are found to be minimum with network configurations of 5 hidden nodes. The two norm is minimum at 2,250 iterations and the one norm is minimum at 2,500 iterations. The minimum for the infinite norm is achieved with a network configuration of 3 hidden nodes and 5,000 iterations. We find no agreement between any of the applied metrics.

We find the minimum for the msee on the random amplitude pulse train sequence is achieved with a network configuration of 5 hidden nodes and 4,500 iterations. Both the one, two and infinite norms are found to be minimum with network configurations of 3 hidden nodes and 5,000 iterations. We find agreement between all three of the induced norms.

100 Delay Node Network

We find the minimum for the msee and the one and two norm on the white noise sequence is achieved with a network configuration of 5 hidden nodes. Minimum msee is reached after 750 iterations. Both the two norm and the one norm achieve minimums after 1,000 iterations. The infinite norm is found to be minimum with a network configuration of 2 hidden nodes and 750 iterations. We note the agreement between the one and the two norms.

We find minimums for the random amplitude pulse train sequence are achieved with a network configuration of 5 hidden nodes for all of the applied metrics. Both the two and infinite norms are found to be minimum after 5,000 iterations. The one norm is minimum after 2,750 iterations and the msee is minimum after 1,250 iterations. We note the agreement between the two and infinite norms.

4.6 Conclusions

In this chapter we have presented three design examples in which we have designed time-delay neural networks for the purpose of system identification. We have used the mean-squared estimation error as defined in Chapter three and the induced norms

defined in this chapter to evaluate and rank the numerous networks we trained. We have compared the results from the metric we have developed with the induced norms. We have illustrated these comparisons with 3-D plots and tables. For the linear systems, there was some agreement between the applied metrics. Sharp contrasts are observed between the msee and the induced norms for the linear systems, however similar trends can be observed between the metrics. The two norm and the one norm were observed to be markedly similar. As previously noted, use of a neural network to model a linear system is using an overly complex model to model simpler system dynamics. Use of an overly complex model results in an overparameterized model to begin with. However, as previously noted, overparameterized models can be used to model less complex systems if training is stopped at an appropriate number of training cycles. For the nonlinear system, the msee and the induced norms were markedly similar in the manner in which they varied with number of delay nodes and training cycles. Distinctive gradations were observed between the msee results for the validation signal estimation errors. Gradations of an extensively smaller magnitude were observed between the induced norm results for the validation signals.

It is also of interest to note that the bias and variance components of the msee were markedly similar in the manner in which they varied with number of delay nodes and training cycles. The primary observable difference was the magnitude of each component. The variance component was observed to be the extensive contributor to the msee in all cases. Bias did tend to decrease as the number of hidden nodes was increased. This result was anticipated from the theory. The phenomenon of overfitting

the training data was also observed in the experimental results. A better prediction performance was achieved in most instances by stopping the back-propagation training algorithm well short of convergence. The variance of the msee did tend to increase with the number of training iterations in some instances. These effects were observed in both the first order and second order systems responses to the white noise and the pulse train validation signals. In some instances, troughs were observed in both the msee and the induced norm metrics. These shallow places indicate network configurations which exhibit improved prediction performance. Peaks were also observed in both the msee and the induced norm metrics. Both the troughs and the peaks were observed to vary with the number of delay nodes. The msee and induced norms for the nonlinear system were markedly high when only one delay node was used to construct the estimator. These results indicate the need for a more complex model achieved by adding more hidden nodes. Both the second order linear and nonlinear system were best modeled by neural networks characterized by the highest sampling rate. The first order system performed best on the intermediate sampling rate. The second order and nonlinear system are more complex than the first order system. This increased complexity may account for the higher sampling rates and increased number of delay nodes. The need for higher sampling rates with more complex systems leads the designer into considering recurrent networks in which plant feedback is used as part of the estimation process. Such recurrent networks are parsimonious in their parameters and yield considerably simpler models. However, the construction of these networks is much more difficult and involves

modifying the back-propagation training algorithm to a form which is known as back-propagation through time.

In conclusion, we have successfully demonstrated the use of our generalization metric to select the best among a set of neural networks of various functional forms and sizes. In particular, we have used the metric to select among a set of sampling times, the number of hidden nodes and number of training cycles.

CHAPTER FIVE

CONCLUSION

5.1 Conclusions

In this thesis, we have developed a practical generalization metric for the measurement of the prediction performance of trained time-delay neural networks. This metric is useful in system identification. It is used to determine an appropriate number of training cycles for each neural network, an appropriate number of delay nodes, and an appropriate number of hidden nodes. This choice leads to the best predictor among a set of several hundred estimators.

We have presented three design examples in which we have identified a first order and second order linear system, and a nonlinear system. In these examples, the metric was used to select the best predictor by determining the network with the highest prediction performance corresponding to the lowest msee. In all of these examples, better prediction performance was achieved by stopping the back-propagation training algorithm well short of convergence.

5.2 Suggestions For Further Research

This thesis serves as a foundation for understanding generalization or prediction performance of neural networks. We have developed a practical measure for generalization performance in neural networks characterized by a dynamical mapping.

A natural extension of this measure is the incorporation of the measure into the back-propagation training algorithm. Incorporation of the measure would involve a modification of the cost function, such that both the bias and variance terms would be minimized during the training process.

Another important area for future research is the use of recurrent networks for system identification. These networks are parsimonious in their parameters and hold great promise in predictive control applications.

BIBLIOGRAPHY

A1 Ash, C. (1993), The Probability Tutoring Book, IEEE Press NY, NY.

B1 Barron, A.R. (1984), "Predicted Squared Error: A Criterion for Automatic Model Selection," Self-Organizing Methods in Modeling, Marcel Dekker, Inc., NY, NY, 87-103.

B2 Baum, E.B. and Haussler, D. (1989), "What Size Net Gives Valid Generalization?," Neural Computation, 1, 151-160.

B3 Billings, S.A. and Fadzil, M.B. (1985), "The Practical Identification of Systems with Nonlinearities," IFAC, Identification and System Parameter Estimation.

B4 Billings, S.A., Jamaluddin, H.B. and Chen, S. (1992), "Properties of Neural Networks with applications to modeling nonlinear dynamical systems," Int. J. Control, vol. 55, no.1, 193-224.

C1 Capra, F. (1983), The Tao of Physics, Bantam Books, Inc.

F1 Fu, B., Hajela, P. and Berke, L. (1993), "Functional Synthesis Using Neural Networks: A Comparative Study," Structures and Controls Optimization, AD-vol.38, American Society of Mechanical Engineering.

F2 Fu, L. (1994), Neural Networks in Computer Intelligence,

G1 German, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma," Neural Computation, 4, 1-58.

H1 Haykin, S. (1994), Neural Networks, Macmillan College Publishing Company, Inc. New York, New York 10022.

H2 Huang, S. and Huang, Y. (1991), "Bounds on the Number of Hidden Neurons in Multilayer Perceptrons," IEEE Transactions on Neural Networks, vol. 2, no. 1.

H3 Hush, D.R. and Horne, B.G. (1993), "Progress in Supervised Networks: What's new since Lippmann?," IEEE Signal Processing Magazine, 10, 8-39.

L1 Lapedes, A. and Farber, R. (1988), "How Neural Nets Work," American Institute of Physics, 442-456.

L2 Le Cun, Y., Denker, J.S. and Solla, S.A. (1990), "Optimal Brain Damage," Advances in Neural Information Processing Systems, 2, 598-605.

L3 Ljung, L. (1987), System Identification: Theory for the User, Prentice Hall, Englewood Cliffs, NJ 07362.

M1 Moody, J.E. (1993), "Note on Generalization, Regularization and Architecture Selection in Nonlinear Learning Systems," Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop, 1-10.

M2 Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," Advances in Neural Information Processing Systems, 4, Morgan Kaufmann Publishers, San Mateo, CA.

R1 Romelhart, D.E., Hinton, G.E. and Williams, R.J. (1985), "Learning Internal Representations by Error Propagation," Parallel Distributed Processing: Explanations in the Microstructure of Cognition vol. 1, ch. 15, MIT Press, Cambridge, MA.

S1 Shekhar, S., Amin, M.B. and Khandelwal, P. (1992), "Generalization Performance of Feed-Forward Neural Networks," Neural Networks: Advances and Applications, 2, 13-38.

S2 Sjoberg, J. and Ljung, L. (1992), "Overtraining, Regularization and Searching for Minimum in Neural Networks," IFAC Adaptive Systems in Control and Signal Processing, 73-78.

W1 White, H. (1990), "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings," Neural Networks, vol. 3, pp. 535-549.

W2 White, H. (1989), "Learning in Artificial Neural Networks: A Statistical Perspective," Neural Computation, 1, 425-464.