Order Number 9300336

Function approximation using neural networks: A simulation study

Marquez, Leorey Orobia, Ph.D.

University of Hawaii, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

FUNCTION APPROXIMATION USING NEURAL NETWORKS: A SIMULATION STUDY

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION
OF THE UNIVERSITY OF HAWAI'I
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY
IN
COMMUNICATION AND INFORMATION SCIENCES

AUGUST 1992

BY

LEOREY O. MARQUEZ

Dissertation Commiittee:

William Remus, Chairperson
Tim Hill
Leon Jakobovits
Larry Osborne
David Yang

ABSTRACT

This dissertation analyzes the effect of different characteristics of data on the training and estimation accuracy of neural networks. The literature on the universal approximation property of neural networks is reviewed. An examination of the relationship of the neural network approach to traditional statistical methods of approximation brought about proposed enhancements to the neural network training procedure.

The study generated data samples characterized by different functional forms, levels of random noise, number and magnitude of outliers, and strength of multicollinearity. These samples were then used to train a neural network. The accuracy of the neural network estimate was tested and compared with the accuracy of the estimates obtained from the true model and those from Specht's GRNN model. Statistics on the length of training and the complexity of the neural network estimate were also collected and analyzed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

α	coefficient of momentum
β	vector of coefficients in a regression model
δ	coefficient of decay
ε	random error term in a regression model
μ	expected value of an observation
τ	learning rate coefficient; also called the step size
σ	size of an outlier in terms of standard deviations
θ	smoothing constant in the GRNN model
$\approx$	approximately equal to
$\Delta\tau$	change in the learning rate
ΔE	change in the cost function
ΔW_{ij}	change in the weight of the connection between node i and node j
E	error cost function
d_i	difference in MAPEs between the i^{th} estimates of two models
GRNN	general regression neural network model
MAPE	mean absolute percentage error
n	size of the estimation sample
NET	weighted sum of the inputs to a node of a neural network
NN	neural network model
OUT	output of a neural network
REG	linear regression model
W_{ij}	weight of the connection between node i and node j
X	vector of input values
Y	vector of output values

CHAPTER 1

THE PROBLEM AND ITS IMPORTANCE

Regression analysis is the most commonly used method of analysis in business. A survey of corporate executives conducted by Forgionne (1983) found that regression analysis was the most frequently used quantitative method. The survey found that 59.6% of the respondents used the method frequently, 38.7% were moderate users, and only 1.6% said that they never had used regression. Computer simulation came in second with 33.9% of the respondents indicating frequent use, 53.2% showing moderate use, and 12.9% not using this method. An earlier survey by Ledbetter and Cox (1977) lends support to these findings by ranking usage of quantitative tools in this order: regression, linear programming, simulation, PERT/CPM, queuing, dynamic programming, and game theory.

The popularity of regression analysis has also led to abuses, especially with respect to business data. In most business data, noise is always present, outliers frequently occur, and multicollinearity often exists between the independent variables. These characteristics of business data make the requirements of regression analysis too restrictive in many cases. As a result, the necessary transformations on the variables are skipped, outliers are ignored, and collinear variables are left unidentified in modeling business data.

Regression problems revolve around function approximation. The neural network approach is a logical choice for these problems since neural networks are universal approximators. This property will be discussed further in Chapter 2. Besides universal approximation, a neural network has other properties that makes it a logical alternative to regression analysis in modeling data that exhibit considerable random noise,

nonlinearity, outliers or multicollinearity. This study will investigate whether neural networks can provide accurate estimates under a specified range of conditions.

This chapter introduces neural networks, discusses the appropriateness of neural networks for regression data, and clarifies what this study hopes to accomplish. Section 1.1 relates a short history of neural networks and the backpropagation technique. Section 1.2 presents the wide range of applications of neural networks while Section 1.3 explains why neural networks have been successful in many of these applications. Finally, Section 1.4 states the general and specific objectives of this study.

1.1 A Brief History of Neural Networks

Neural networks (NN) are input-output models inspired by the workings of the human brain. For a given input vector X, the neural network will produce an output vector Y, such that $Y = g(X)$ for some function g. However, neural networks have very little in common with their biological counterpart. These models are extremely simplified when seen from the neurophysiological point of view. Nevertheless, neural networks can provide insight into the computational algorithms the brain uses for different tasks (Hertz et al., 1991, p.5).

McCulloch and Pitts (1943) proposed the first computational model of a neuron or nerve-cell. This model represented the neuron as a binary threshold unit. The unit computed a weighted sum of its inputs from other units, and output a one when this sum was above a certain threshold. Otherwise it output a zero. McCulloch and Pitts (1943) proved that a network of such neurons was capable of any type of computation provided a suitable set of weights was chosen.

In 1957, Rosenblatt (1957) introduced the perceptron, a neural network where the neurons are organized into layers with feedforward connections between the layers.

Rosenblatt's group focused on the problem of finding appropriate weights for the connections. For the one-layer perceptron, Rosenblatt (1962) proved the convergence of an algorithm that changed the weights iteratively for particular computational tasks. Unfortunately, the algorithm only applied to those tasks that the perceptron is capable of computing.

In their book *Perceptrons*, Minsky and Papert (1969) showed that one-layer perceptrons were not capable of computing some very simple problems such as the XOR problem. Multiple layers are required to overcome these limitations. However, Minsky and Papert doubted that a learning algorithm for multi-layer networks would be found (Hertz et al., 1991, p.7). As a result, the computer science community virtually lost interest in neural networks.

It was not until 1974 that the sought-after algorithm, backpropagation, was developed. In his master's thesis *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Paul Werbos introduced a set of techniques including what is now known as backpropagation. He referred to backpropagation then as the dynamic feedback method that "can be used to minimize or maximize any function we please..." (Werbos, 1974). The potential applications included the estimation of relationships and patterns so that error is minimized or probability is maximized. Backpropagation would later be rediscovered independently by Parker (1985) and by Rumelhart, Hinton and Williams (1986).

1.2 Applications of Neural Networks

Seventeen years after Werbos' thesis was published, interest in backpropagation and neural networks exploded. There are hundreds of papers published every year on the subject. Applications are as diverse as game playing, bankruptcy prediction, task

coordination and sonar signal processing. The commercial use of neural networks is likely to grow rapidly.

Neural networks have non-commercial applications too. Given the potential of neural networks for military applications, the Defense Advanced Research Projects Agency (DARPA) budgeted $33 million in 1989 to explore artificial neural network technology (Yoon, 1989). Its goals were:

1. to identify and investigate advantages of neural networks relative to conventional approaches,
2. to advance the state of the art in artificial neural network theory and modeling, and
3. to develop advanced hardware implementation technologies for future construction of large artificial neural network computers .

Why has so much attention been focused on neural networks? Neural networks and other parallel processing architectures offer new approaches to solving problems that have proved challenging for traditional serial processing. Neural network applications in image processing (Cottrell, Munro and Zipser, 1987; Burr, 1986, 1987), speech processing (Sejnowski and Rosenberg, 1987), and robotic control (Szu, 1986) have surmounted traditional barriers. Parallelism allows NN models to perform extremely fast processing in these tasks in spite of the presence of incomplete and noisy data. These tasks exceed the capacity of many supercomputers. In addition, the high connectivity of neural networks renders a few errors in the computation inconsequential. This makes the system fault-tolerant. In contrast, an ordinary sequential computation may easily be ruined by a single bit error (Hertz et al., 1991, p.5).

Neural network models can be formed for almost any type of data. Naturally, applications have been sought in areas considered strongholds of traditional methods of analysis. These areas include time series forecasting, function approximation, classification, operations research, and decision support systems. NN techniques are

being used to provide solutions to problems in these areas where traditional methods have been least effective. Already, studies have shown that, under certain conditions, NN models provide forecasts for time series that are at least as accurate as those provided by the Box-Jenkins method (Sharda and Patil, 1990a,b). In addition, NN models have been found to be at least as effective as discriminant analysis in classification problems (Odom and Sharda, 1990; Surkan and Singleton, 1990).

The performance differences between neural networks and traditional statistical models can be substantial. Comparisons are based on the applicability of each model, on the quality of the estimates, and on the overall significance of the parameters. When asked to cite the performance and elegance differences between neural networks and statistical techniques (e.g., discriminant analysis), Marvin Minsky (1991) wrote:

> The NN methods form a wider class of clustering problems. Wider for several reasons:
>
> 1. The result is not constrained to be unique. The same data can produce different classifications, because the learning trajectory can depend on the order in which the data is presented.
>
> 2. NN methods are still largely empirical. A method becomes popular if enough researchers claim that it gives good results. In most cases, very little has been proven about the range and reliability of the method. Statistical methods, in contrast, are not considered 'scientific' unless accompanied by theorems about their behavior.
>
> 3. In order to prove a theorem about classification, the theorem's antecedent must precisely describe a class of classification problems. In the real world, this is really hard to do -- so mathematical statistics tends to confine itself to idealized well-defined cases. The AI and NN researchers don't restrict themselves that way.

Neural network methods for signal processing, pattern recognition, and even time series analysis have gained prominence because of the empirical successes achieved in these areas. As a result, the neural network approach is seen as an alternative to the traditional techniques used in these areas. This study hopes to accomplish a similar result in the area of regression analysis.

1.3 Characteristics of Neural Networks

What makes neural networks appropriate for data analysis? First, neural networks are universal approximators. A neural network can provide an approximation to any function of the input vector **X**, provided the network has a sufficient number of nodes. Because of this property, neural networks appear well suited for applications in pattern recognition, classification, forecasting, process control, image compression, and other related tasks (White, 1989c).

Second, neural networks are claimed to be capable of learning, generalization and abstraction. According to Wasserman (1989), neural networks can learn from experience, can generalize and "see through" noise and distortion, and can abstract essential characteristics in the presence of irrelevant data. Lippman (1987) adds that these models provide a high degree of robustness and fault tolerance because of the number of processing nodes that can have primarily local connections.

These claims attain special significance for neural network models since the characteristics apply directly to those conditions in which regression models have difficulty. For example, regression models require suitable transformations for the variables, have difficulty estimating weak linear relationships, and have poor parameter estimation when outliers occur. NNs' capacity for learning and self-transformation promises an alternative to the guesswork involved in identifying the distributions and transformations required in a linear model. NNs' generalizability may allow them to "see through" the noise and distortion of weak relationships and provide acceptable levels of forecasting accuracy. NNs' capacity for abstraction may offer a new way of dealing with outliers and irrelevant data.

Neural networks are not without their shortcomings, foremost of which is the difficulty of the training when using backpropagation. Since backpropagation estimates

the network parameters using an iterative process, the amount of time and computation needed for a neural network to converge to an appropriate set of values for the parameters can be significantly larger than those of a non-iterative process. In some cases, the network may not converge at all. There is also the danger of converging and getting trapped in a set of weights that corresponds to a local minimum of the estimation error.

In addition, every researcher is faced with the daunting task of selecting a suitable network structure. A myriad of alternatives is available in deciding the number of nodes, the number of layers, the number of nodes in each layer, and other components of the network. This structure will then have to be initialized with a set of starting weights. Hopefully, a "good" set of weights is chosen since the initial settings play a significant role in the length and success of the training.

1.4 Objectives of the Study

The general objective of this study is to investigate the training and performance of neural networks in approximating the function underlying a given set of data. Specifically, the study aims to:

1. explore how different characteristics of data, such as noise level, sample size, and functional form affect a neural network's accuracy in estimating the data,
2. evaluate the estimation accuracy of the neural network model against that of the true generating model,
3. demonstrate the effectiveness of supplementing the neural network approach with a simple nonlinear optimization technique,
4. explore the automation of the network training process, and
5. develop heuristics for training neural networks.

This study will evaluate NN performance on simulated data in order to control such characteristics as sample size, the level of random noise, the presence of outliers,

the level of multicollinearity, and the functional form of the underlying distribution. The study will attempt to determine what properties of the generated data are responsible for the claimed characteristics of NNs, and how these characteristics relate to the nature of network training and the structure of the resulting model.

The accuracy of the neural network estimates will be compared with that of the true model using several sets of test data. The study will examine how close the neural network performance comes to that of the true model and under what conditions this performance occurs. In addition, the neural network estimate will be compared to other misspecifications, including a "nearest-neighbor" estimate. These comparisons will be used to identify the conditions where the neural network model can be expected to provide a more accurate estimate than other misspecified models.

As mentioned earlier, a network's initial condition has a significant effect on the length and success of the training. In the early stages of training, a simple nonlinear optimization method will be applied prior to backpropagation. This optimization method, called the downhill simplex, will again be used after convergence slows to improve the estimates produced by backpropagation. The study will determine whether the addition of the downhill simplex can be effective in decreasing the length of training and improving the network estimates.

The suitability of neural networks as a practical tool could be enhanced by decreasing the number of decisions required by the researcher during the training process. A procedure for automating the decisions related to pruning and parameter updating will be described.

Finally, the study presents a set of guidelines or a heuristic for training neural networks. These guidelines relate to matters such as the scaling of the data, the initialization of the network parameters, the choice of model to use, and other aspects of

the training process. These guidelines will help future users of neural networks obtain satisfactory models with a minimum of training time.

CHAPTER 2

A REVIEW OF LITERATURE

The previous chapter cited the potential of neural networks for modeling any type of data. This chapter presents the theoretical foundation of this potential and explains why neural networks represent a new class of nonlinear regression models. In addition, this chapter critiques several studies that applied the neural network approach to regression problems.

Section 1.1 reviews the general regression problem while Section 2.2 discusses the traditional way of estimating models using regression analysis. Section 2.2 also mentions the difficulty of searching for an appropriate model in the absence of any theoretical background for the data. Under these conditions, neural networks offer an attractive alternative. Section 2.3 cites the literature that provides proof of the universal approximation property of neural networks. Section 2.4 shows why the use of backpropagation in training neural networks produces estimates that are equivalent to those obtained for regression models by the method of least squares. Section 2.5 describes neural networks that use training methods other than backpropagation. Finally, Section 2.6 analyzes the success or failure of the neural network approach in previous studies that modeled regression data.

2.1 The General Regression Problem

Regression analysis is a statistical tool that can be used to predict a variable based on its relationship to other quantitative variables (Neter et al., 1989, p.23). The regression problem requires the calculation of the parameters of the model that best fits a given set of data. This model is usually defined to minimize the squared residuals between the predicted values and the observed values.

The general nonlinear regression model can be stated as (Bates and Watts, 1988, p.32):

$$Y_i = f(X_i, ß) + \varepsilon_i \qquad (2.1)$$

where

Y_i is the value of the response variable for the ith observation,

f is the expectation function,

X_i is the vector of associated regressor variables or independent variables for the ith observation,

$ß$ is the vector of parameters to be estimated, and

ε_i is the random error associated with the ith observation.

When the expectation function f is linear with respect to the vector of parameters $ß$, equation (2.1) reduces to the familiar linear regression model:

$$Y_i = ß^t X_i + \varepsilon_i \qquad (2.2)$$

2.2 Estimating the Model

When f is linear with respect to X_i and $ß$, the estimates of $ß$ are obtained directly using ordinary least squares (Bates and Watts, 1988, p.33). When f is linear with respect to $ß$ only, transformations are performed on one or more of the independent variables before ordinary least squares is applied. This step often involves trial and error. The problem of finding the appropriate transformation has been extensively studied (e.g., Atkinson, 1985; Box and Cox, 1964; Daniel and Wood, 1980; Box and Tidwell, 1962; Mosteller and Tukey, 1977; Neter et al., 1989).

For relationships that cannot be transformed into linear form, the least squares estimates are obtained by iterative methods (Neter et al., 1989; Kennedy and Gentle, 1980; Gallant, 1987). Traditionally, the Gauss-Newton method (Hartley, 1961) had been used for fitting nonlinear regression. The current standard, the Levenberg-Marquardt

method, combines the best features of the Gauss-Newton method and the method of steepest descent (Neter et al., 1989, p.562).

Just as in the linear case, an important step in any nonlinear analysis is the specification of the model. This step includes the selection of the form of the expectation function and the assumption of the characteristics of the error term. This step requires considerable effort and skill from the researcher. Ideally, physical, biological, chemical or other theoretical considerations lead to a mechanistic model for the expectation function. If theoretical considerations are nonexistent, a search of the literature may turn up models applied in similar situations (Bates and Watts, 1988, p.69). Otherwise, the researcher will have to advance a model that mimics the behavior of the data. Unfortunately, this step can easily degenerate into a trial-and-error process.

As we have seen, researchers using regression analysis, both linear and nonlinear, face the task of specifying the functional form of the model to be estimated. If the researcher is familiar with the data, the identification of the functional form becomes a routine procedure. Otherwise, considerable time and effort may be needed to find a suitable model. In these cases, neural network models present a particularly viable alternative.

2.3 Neural Networks as Universal Approximators

Neural networks have shown impressive successes in applications involving pattern recognition, classification, process control, image compression, feature detection and forecasting. These successes are not without theoretical foundation.

Cybenko (1988) and Lapedes and Farber (1988) proved that neural networks with two hidden layers can approximate arbitrary continuous functions, both linear and nonlinear. Lapedes and Farber (1988) found that the accuracy of approximation is

controlled by the number of nodes in each layer and not the number of layers. They concluded that a network approximation of a function with p input variables requires 2p nodes in the first hidden layer and one node in second hidden layer for each critical point in the function. They emphasized that while two hidden layers may be sufficient, such a model may not be efficient in terms of accuracy and cost.

Subsequently, several researchers (Gallant and White, 1988; Irie and Miyake, 1988; Funahashi, 1989; Cybenko, 1989; Carroll and Dickinson, 1989; and Hecht-Nielsen, 1989) found that any continuous mapping can be approximately realized by a neural network with at least one hidden layer. Hornik, Stinchcombe, and White (1989) provided a rigorous proof that neural networks using arbitrary sigmoid functions, and no squashing at the output layer are universal approximators. The approximation can be achieved to any desired degree of accuracy provided sufficiently many hidden units are available. They attributed any failure in application to inadequate learning, insufficient number of hidden units or the lack of a deterministic relationship between input and target.

The above conclusions assume that the single hidden layer contains a sufficiently large, perhaps infinite, number of nodes. Chester (1990) found the assumptions for approximation too restrictive for most practical applications. He proceeded to show that a network with four nodes in two hidden layers can achieve any level of accuracy in approximating any continuous function; this would require just the adjustment of a few parameters.

As mentioned earlier, one of the principal reasons why a neural network may fail to approximate a given function is inadequate learning. The universal approximation capability of neural networks cannot be realized without an appropriate training procedure. Several training algorithms have been proposed, the most commonly used of

which is backpropagation. The next section explains why backpropagation is equivalent to performing nonlinear least squares approximation to a set of data.

2.4 Backpropagation as Nonlinear Least Squares

In the same way that nonlinear regression models use nonlinear least squares to obtain the model parameters, neural networks use backpropagation to obtain the weights of the connections. White (1989a, 1989c) showed that neural network learning methods, specifically backpropagation, are equivalent to well-known statistical estimation methods for nonlinear regression models. (For a review of neural network training and backpropagation, please refer to Appendix B.) This links neural networks to the nearly 40 years of methodology development in stochastic approximation and system identification. Statistical theory can now be applied to make general statements about the properties, advantages, and disadvantages of backpropagation and neural networks.

Instead of searching for the functional form of f as is the procedure in ordinary nonlinear regression, the neural network approach represents f with a network containing one or more hidden layers. The network inputs are the independent variables, and the weights are the regression parameters. The architecture of the network will determine the precise form of the function f. Thus, instead of estimating the parameters of f, the ß vector, with ordinary least squares or nonlinear least squares, the researcher can use backpropagation to train the neural network. The estimates of ß will be represented by the weights of the connections in the trained network. Note that different neural network structures will lead to different forms for f (White, 1989b).

Using statistical theory, White (1989a, 1989b, 1989c) reached the following conclusions concerning backpropagation:

1. Under appropriate conditions, backpropagation learning either diverges or converges to a set of weights providing (locally) optimal predictions for Y_i.

2. The backpropagation estimates are asymptotically normal.

3. A Wald statistic can be used to test the hypothesis that some group of inputs is of no value in attaining the locally best approximation.

4. A Wald statistic can be used to test the hypothesis that a particular group of hidden units is of no value in attaining the locally best approximation.

5. Compared to nonlinear least squares, backpropagation fails to make efficient use of all the statistical information contained in the training set. However, certain modifications to backpropagation can lead to recovery of this information.

To make backpropagation more efficient and increase its chance of converging to an optimal set of weights, White suggests the following:

1. The learning rate used in training should be adapted dynamically and should gradually decrease to zero (White, 1989b).

2. Once backpropagation has converged to a set of weight estimates, White (1989c) suggests that one or more additional iterations of the Newton-Raphson method or any other nonlinear optimization method be applied to the neural network.

White's suggestions and other modifications for improving the backpropagation technique will be discussed in detail in Chapter 3.

2.5 Non-Backpropagation Neural Networks

Because of the inefficiencies associated with backpropagation, other implementations of neural networks for regression analysis have been explored. Specht (1991) proposed a "general regression neural network (GRNN)" that computes its output using a variation of the "nearest-neighbor" approach. Using a dynamic plant modeling example, Specht showed that the GRNN model can be 100 times faster than a backpropagation model and still produce the same magnitude of error. This result stems from the fact that Specht's model does not require iterative training since it obtains its estimate in a single pass. This study will make performance comparisons between Specht's model and the

backpropagation neural network under different conditions. A more detailed discussion of Specht's technique will be presented in the next chapter.

A promising approach offered by Friedman is called adaptive spline networks (Friedman, 1991b), a network adaptation of his multivariate adaptive regression splines (MARS) procedure (Friedman, 1991a). In these networks, spline functions are used as activation functions, instead of the usual sigmoid. In addition, the input values to a node are multiplied instead of added, enabling the network to produce local approximations. Friedman claims that spline networks are more flexible and are faster to train than backpropagation networks. He adds that spline approximations provide considerable interpretable information concerning the target function, one feature that is lacking in backpropagation networks.

Neural networks have been shown to possess the capability to approximate any mapping, including linear and nonlinear functions. The backpropagation method has been shown to provide estimators that are asymptotically equivalent to those obtained by the method of least squares. With a few enhancements, backpropagation can be modified to operate efficiently, and automatically, while reducing the complexity of neural networks. These enhancements have been implemented in some studies. The next section reviews the existing literature on the application of backpropagation neural networks in regression problems.

2.6 Comparisons with Regression Analysis

Several attempts have been made to apply neural networks in problems where regression analysis has been traditionally used. Studies focusing on the relationship between neural network models and regression models have been conducted using actual as well as simulated data. Unfortunately, for most of these studies, the information given on the

training and structure of the network is insufficient to allow researchers to duplicate successes in one area into other areas.

Duliba (1991) compared neural network models with four types of regression models in predicting airline performance. She found that the neural network model outperformed the random effects regression model but not the fixed effects model. Dutta and Shekhar (1988) used 10 factors in predicting the ratings of corporate bonds where conventional mathematical modeling have yielded poor results. In this nonconservative domain, the authors experimented with 2-layer and 3-layer backpropagation networks. The results showed that the neural networks outperformed the regression models.

Another nonconservative domain is the area of process and quality control. Smith and Dagli (1990) initially used stepwise regression to determine the four most significant variables (of 18 variables) in predicting the quality of plastic pipes. Regression models and backpropagation models were then estimated using these four variables. The results showed that the backpropagation models performed at nearly the same level as the regression models. The authors concluded that neural network models and regression models can be separate but complementary modules in a data modeling system.

Fishwick's study (1989) compared neural network methods with linear regression and surface response methods in fitting simulated ballistics data. His results show that the forecasting ability of neural networks was inferior to that of simple linear regression and surface response methods. The traditional methods showed superior performance in both execution time and approximation value. He traced the inadequacy of his neural network models to their inability to capture the system structure. He called for more systematic ways of upgrading, refitting and training of neural network models.

Remus and Hill (1990) compared the quality of the decision based on neural networks and those based on linear decision rules, as applied to a production scheduling

problem. Decision-makers using the neural network model performed as well but not better than those using the linear regression models. The experiment showed the implementation advantages of neural networks when used in intelligent systems.

Marquez et al. (1991) generated data from three known distributions and then compared the performance of the neural network models against the true regression models. The results showed that neural network models perform within 2% of the mean absolute percentage error (MAPE) for the linear-based and reciprocal-based regression models but had difficulty learning the data from the log-based model.

White (1989c) generated 4,000 data points based on the Henon map $Y_t = 1 - 1.4Y_{t-1}^2 + 0.3Y_{t-2}$ (t = 1,.. 4000) with randomly selected starting values Y_0 and Y_{-1}. The resulting series was highly erratic. He used backpropagation and Newton-Raphson to train three different single-hidden layer neural networks. He also obtained the least squares estimates for the corresponding linear models. His results showed that the linear models gave a poor approximation of the Henon map but the neural networks provided a good fit.

Logistic regression is commonly used in classification problems where the response variable has a binary value. Neural networks also have solid application potential for these problems. Bell et al. (1989) compared backpropagation networks against logistic regression models in predicting commercial bank failures. The neural network model performed well in failure prediction and the expected costs for misclassification by the neural models were found to be lower than those of the logit model. Roy and Cosset (1990) used neural network and logistic regression models in predicting country risk ratings using economic and political indicators. Their results showed that neural network models had lower mean absolute error in their predictions and reacted more evenly to the indicators than their logistic counterparts.

The results of the above studies generally paint a bright picture for neural network applications in regression analysis. Most of the comparisons show that the neural network performance was comparable if not superior to the performance of the corresponding regression model. This is primarily because of the significant amount of nonlinearities and noise in the data. Unfortunately, a deeper analysis of the performance of the neural networks is hampered by the almost universal lack of specifics on the training and architecture of the networks. It would be almost impossible to replicate the results of the authors based solely on the information presented in the papers.

For those experiments where the neural networks performed poorly, the question is whether a suitable network configuration was used, and whether the network was trained sufficiently. Of the above papers, only White (1989c) clearly showed the use of a minimal network configuration. The others arrived at their final configuration by a process that was basically trial and error.

Most of the studies neglected to identify the values for learning rate and momentum. More importantly, the studies failed to clarify whether the learning rate was fixed or adapted dynamically. For Marquez et al. (1991), the network's difficulty in learning the log-based data can be traced to the use of a fixed learning rate and momentum. Fishwick (1989) also fixed these parameters and had similar results.

Although majority of the authors seemed conscious of overfitting the data, none of the networks were pruned to minimal size. This may be due to the large number of patterns used in the training data as compared to the size of the networks.

As a whole, these studies prove the capability of neural networks to provide estimates competitive to those of regression analysis. However, the papers failed to provide sufficient information on how the authors accomplished their results. Very little

light is shed on what characteristics of the data, which features of the network and what aspects of the training lead to good approximation.

In contrast, this study will provide a comprehensive assessment of the whole experiment. The procedure for generating data will be described. The network structure will be given and its initial settings will be specified. Most importantly, the training process will be outlined. This information should enable other researchers to verify the results of the simulations. Chapter 3 discusses the methodology used in conducting this study.

CHAPTER 3

METHODOLOGY

This study investigates the relationship between the characteristics of the data to be modeled, the nature of the neural network modeling process, and the accuracy of the model estimates. To accomplish this objective, the methodology employed in this study has been divided into three stages. The initial stage requires the identification of specific data characteristics to be included in this study. Once the data characteristics have been identified, samples exhibiting these characteristics are generated. These samples will be used for model estimation as well as for testing the model estimates. Model estimates are then obtained in the second stage. The models are estimated using three well-known approaches. In the final stage, the models are analyzed and their forecasting accuracy are compared.

This chapter will provide a detailed discussion of the methodology employed in this study. Section 3.1 presents the characteristics of data that will be the focus of this study. Section 3.2 presents the three types of models that will be estimated from the data. Section 3.3 outlines the activities involved in estimating the models. Finally, Section 3.4 discusses the procedure for comparing the performance of the different model estimates. This section will also describe how neural network training will be evaluated.

3.1 Data Characteristics

The purpose of modeling is to estimate the functional relationship between the input values and the output values of a sample of data. With an estimate of the functional relationship, forecasts can be made of the expected output value for any given set of input values.

Unfortunately, the estimation of the functional form of the relationship based on a sample of data can be a difficult process. The functional form may be complicated. The sample may be too small to provide any indication of the functional form. Considerable noise may be distorting the data.

Noise may take several forms. For most observed data, the environment adds random noise to the observations. The sample may also include several outliers that exert undue influence on the sample statistics. Finally, a linear relationship may exist between some of the input values further complicating the functional relationship between the input values and the output values.

The above conditions describe the characteristics of data that will be the subject of this study. The focus will be on the functional form of the relationship and on the level of random noise in the data. Random noise or random error is important because it is always present in the data. Most models include a component to represent random error. In linear regression, the random errors are assumed to have a mean of 0 and a constant standard deviation. The same assumptions will be adopted in this study.

Among the characteristics, the functional form of the model clearly contains the widest range of possibilities. The uncertainty of the functional form coupled with the infinite number of forms to choose from makes the modeling process difficult for all techniques, even for established ones like regression analysis.

3.1.1 Functional forms

The crucial step in performing data analysis is the specification of the functional form of the model. If sufficient theoretical background on the data is known, then the model can be specified and estimated in a straightforward manner using regression analysis. However, in the absence of theoretical background, one or more functional forms will

have to be proposed and tested. If the functional form hypothesized is similar to the true underlying distribution, the forecast error is expected to be small. Otherwise, the researcher will have to continually search for a functional form until one that mimics the behavior of the data and produces a satisfactory level of error is found. This search can deteriorate into a long and difficult process.

Among the functional forms, the simplest are those that can be represented by a relation of the form $Y = X^P$. When the power p takes on the values ...2, 1, 0.5, 0, −1, etc., a series of curves result forming what is called "a ladder of reexpressions" (Mosteller and Tukey, 1977). This ladder is an important tool in regression analysis. Given a set of points from an unknown curve, the idea is to move up and down the ladder, searching for the expression that will transform the points into a straight line.

Since the value of p=0 in the ladder leads to a constant and is useless, Mosteller and Tukey chose to replace p=0 with p=# to represent the instance when the relationship is $Y = \ln(X)$. This is important since the logarithmic function is one of the most commonly used transformation in regression analysis. In addition, the ladder of curves can be thought of as coming from the $\int X^{(p-1)}\, dX$. For positive values of X, the curves are monotonically increasing. The curves are concave upward for p>1, straight for p=1, and concave downward for p<1.

For this study, the ladder consisting of the five curves $Y=X^2$, $Y=X$, $Y=\sqrt{X}$, $Y=\ln(X)$, and $Y=1/X$ will be used to generate data for model estimation and for testing. This ladder consists of the powers 2, 1, 1/2, #, and -1. These five curves represent five of the most common transformations used in regression analysis. From these five curves, more complicated expressions can be obtained. For example, the quadratic expression $Y=X^2+2*X$ represents a sum of two simple curves while the expression $Y=X*\ln(X)$ represents a product.

Three more complicated univariate functional forms will be used. These functions were derived from models that are commonly used in nonlinear regression. The three functions are:

Y=X/(X+5) (from the Michaelis-Menten model),

Y=39*(1-(.828*exp(-.159*X))) (from the exponential rise model), and

Y=2+(1/(1+exp(0.477*X))) (from the logistic model).

To obtain representations of the multivariate case, we use combinations of the five simple curves on two independent variables.

$$Y = X_1 + X_2 \quad \text{and} \quad Y = X_1 + \ln(X_2)$$

represent sums of simple curves in two independent variables while

$$Y = X_1 + X_1*X_2 + X_2 \quad \text{and} \quad Y = X_1 + X_1/X_2 + X_2$$

are bivariate functions with interaction terms.

To recap, a total of fourteen functional forms will be used to generate data. The functional forms represent the ideal transformation that will convert the curve into a straight line. These functions will be divided into three groups.

The first group consists of the five simple curves that form a "ladder of reexpressions." The ladder is used as a tool for finding the ideal transformation for a set of data. The idea is to move up or down the ladder until an expression that transforms the data into a straight line is found. The closer the function is to the ideal form, the more linear the transformed data would appear.

The second group consists of five functions that are combinations of the simple curves or are based on models commonly used in nonlinear regression. The third group consists of combinations of the simple curves but utilizing two independent variables. Table 3.1 shows the three groups of functional forms. The label that will be used to refer to the functional form is given in brackets.

Table 3.1
Models for Generating Data

1. Simple functional forms	
a. $Y = X + \varepsilon$	[LINEAR]
b. $Y = X^2 + \varepsilon$	[SQUARED]
c. $Y = \ln(X) + \varepsilon$	[LOG]
e. $Y = \sqrt{X} + \varepsilon$	[SQRT]
f. $Y = 1/X + \varepsilon$	[RECIP]
2. Combinations of the simple functional forms	
a. $Y = X^2 + 2*X + \varepsilon$	[XSQX]
b. $Y = X * \ln(X) + \varepsilon$	[XLNX]
c. $Y = X / (X+5) + \varepsilon$	[MICHAELIS]
d. $Y = 39 * (1 - (.828 * \exp(-.159*X))) + \varepsilon$	[EXRIS]
e. $Y = 2 + (1 / (1 + \exp(0.477*X)) + \varepsilon$	[LOGISTIC]
3. Simple multivariate forms	
a. $Y = X_1 + X_2 + \varepsilon$	[X1+X2]
b. $Y = X_1 + \ln(X_2) + \varepsilon$	[X1LOGX2]
c. $Y = X_1 + X_1*X_2 + X_2 + \varepsilon$	[X1*X2]
d. $Y = X_1 + X_1/X_2 + X_2 + \varepsilon$	[X1/X2]

3.1.2 Random noise

In Table 3.1, the models that will be used to generate data consist of a functional form and a random error term. The random error term has been included to simulate the

random noise that is always present in observed data. To obtain a complete picture of the effect of random noise in modeling, three levels of random noise will be introduced for each functional form.

Random noise will be measured in terms of the coefficient of determination (R^2) obtained from the true regression model. When the R^2 is around 90%, the random noise is said to be low. An R^2 approximately equal to 60% will correspond to medium noise, while a high noise level will bring about an R^2 of around 30%. Samples of data points will be generated that will characterize each functional form under each level of noise. For a detailed discussion of the process of generating a sample of points (**X**, **Y**) that follows a given functional form and yields a given level of R^2, the reader is referred to Appendix C.

3.1.3 Samples for estimation and testing

Every sample of data that is generated will be used either for model estimation or for model testing. When a sample is used for model estimation, the points in the sample are used to compute the parameters of the model. In linear regression, the sample is used to compute for the coefficients ß in the model. For neural networks, the estimation sample is used to train the network. In the latter case, the sample for model estimation is also called the training sample.

Samples for model estimation will consist of sixty points. Although sixty observations is considered large for most applications, it is small enough to keep the length of time needed to obtain the model estimate (especially for neural networks) at a minimum.

After a model estimate has been obtained, the forecasting accuracy of this estimate is evaluated using a (separate) testing sample. The input values (**X**) in the testing sample

are used to obtain forecasts from the model estimate. These forecasts are then compared with the actual or target output values (Y) in the testing sample to obtain the magnitude of the errors.

All testing samples contain 100 points. One hundred points is considered large enough (n>30) to allow the assumption of normality of errors for the model estimates.

For each of the above functional forms, 100 samples for model estimation with low noise ($R^2 \approx 90\%$, $s_X \approx 1.0$), 100 samples with medium noise ($R^2 \approx 60\%$, $s_X \approx 1.0$), and 100 samples with high noise ($R^2 \approx 30\%$, $s_X \approx 1.0$) will be generated. (Recall each sample consists of 60 points.) This means that for each functional form and each level of noise, 100 model estimates will be obtained. Each of these 100 estimates will be tested for forecasting accuracy.

To test the forecasting accuracy of each model estimate, three test samples will be used. One of these will have random noise in the data and will be referred to as the regular test data; the level of noise in the test data will be the same as the level of noise when the model was estimated. The regular test data is intended to show how well a model estimates data with the same characteristics as those in the sample from which the model was estimated.

In addition, one test sample with no noise (R^2=100%, $s_X \approx 1.0$) and one test sample with no noise and a wider range of values (R^2 is 100%, $s_X \approx 1.25$) will be generated. The former will be referred to as the noiseless test data, and the latter will be called the extended test data. The noiseless test data is intended to determine how well the model recovers the pure distribution underlying the estimation sample. The extended test data will show how well the model extrapolates outside the range from which it was estimated.

To illustrate, suppose we would like to examine how well a neural network mimics the LOGISTIC functional form. Models obtained from the samples for model estimation with low noise will be tested using the regular test data with low noise, the noiseless test data, and the extended test data. Models obtained from the samples for model estimation with medium noise will be tested using the regular test data with medium noise, the noiseless test data, and the extended test data. Similarly, models estimated using the samples for model estimation with high noise will forecast the regular test data with high noise, the noiseless test data, and the extended test data.

The estimation and testing of models for the 14 functional forms and 3 levels of noise constitute the central experiment of this study. The central experiment will cover a total of 42 (14*3) cases. In each case, model estimates will be obtained from each of 100 samples for model estimation. Each of these estimation samples contains 60 points. Each model estimate is then tested using a different sample of 100 points.

Apart from the central experiment, three side experiments will be performed to evaluate model estimates with respect to other characteristics of data. The first side experiment will focus on the effect of the size of the estimation sample and the level of noise on the quality of model estimates. The second side experiment will focus on outliers while the third side experiment will examine multicollinearity.

3.1.4 Side experiment 1: sample size

All of the samples for model estimation generated to investigate the different functional forms consist of 60 points. Sixty is considered large and will allow the Z-test to be performed on the model estimates. However, obtaining a large sample usually entails a high cost. Large samples may be impossible in some actual situations. Thus, smaller samples are often used. However, small samples may not provide the same amount of

information as large samples. The resulting model estimates may be significantly less accurate. In addition, T-tests are usually performed on the model estimates instead of Z-tests.

To investigate the effect of smaller estimation samples on the quality of the model, estimation samples of size 15 and 30 will be generated. Samples of size 15 will be called small samples while samples of size 30 will be referred to as medium samples. The medium samples represent a compromise between the information offered by large samples and the economy offered by small samples.

As in the central experiment, three levels of noise will be introduced for each sample size. However, only the LINEAR model will be used for data generation. The LINEAR model is deemed sufficient to provide an indication of the effect of sample size and level of noise on the quality of model estimate. The labels that will be used to refer to the sample sizes are:

1. Small samples (n = 15) [LIN15]
2. Medium samples (n = 30) [LIN30]
3. Large samples (n = 60) [LINEAR]

In summary, side experiment 1 will consist of 9 cases (3 sample sizes x 3 levels of noise) all based on the LINEAR model. For each case, 100 model estimates will again be obtained and evaluated using testing samples of 100 points.

3.1.5 Side experiment 2: outliers

Besides random noise, outliers also produce distortion in the data. Outliers, aside from representing noise, may also be an indication of the inappropriateness of the hypothesized model. To study the effect of outliers on the forecasting ability of a neural

network, two analyses will be performed. Both analyses will use data generated from the LINEAR model.

The first analysis will focus on the effect of the number of outliers and their magnitude on the quality of the model estimate. This analysis will determine how much the accuracy of a model decreases with increases in the number of outliers or in the magnitude of the outliers. Five variations of the LINEAR model will be considered. The first variation will be the original LINEAR model with no outlier. The second variation will produce an estimation sample with one outlier, two standard deviations from its expected value. The third variation has estimation samples containing one outlier, four standard deviations from its expected value. The last two variations are similar to variations two and three except that two outliers of that magnitude are present instead of one. As in the central experiment, all estimation samples will have 60 points and all testing samples will have 100 points. The testing samples will not include outliers. A total of 15 cases (5 variations x 3 levels of noise) will be examined. The five variations of the LINEAR model and their labels are as follows:

1. No outlier [LINEAR]
2. One outlier, two stand. dev. [LIN1O2σ]
3. One outlier, four stand. dev. [LIN1O4σ]
4. Two outliers, two stand. dev. [LIN2O2σ]
5. Two outliers, four stand. dev. [LIN2O4σ]

It is believed that an outlier creates a greater distortion on a small sample than in a large sample. Thus, the second outlier analysis will look at the combined effects of a single outlier and the estimation sample size on the performance of a model estimate. The size of the estimation sample will again vary between small (n=15), medium (n=30) and large (n=60).

All estimation samples will be generated using that variation of the LINEAR model that produces one outlier that is two standard deviations from its expected value. A total of 9 cases (3 sample sizes x 3 levels of noise) will be investigated. In each case, 100 model estimates will again be obtained and tested. As in the first analysis on outliers, the testing samples will contain 100 points and will have no outliers. The labels that will be used in this analysis are:

1. One outlier, 2 s.d., n = 15 [LIN15N2σ]
2. One outlier, 2 s.d., n = 30 [LIN30N2σ]
3. One outlier, 2 s.d., n = 60 [LIN1O2σ]

Note that the LIN1O2σ data will be the same one used in the first analysis on outliers.

3.1.6 Side experiment 3: multicollinearity

Multicollinearity represents another condition that can complicate the relationship between predictor variables and response variables. This side experiment will study the effect of the strength of multicollinearity between two predictor variables on the performance of a model estimate.

This side experiment will generate data based on the bivariate linear model

$$Y = X_1 + X_2 + \varepsilon_1$$

where $X_2 = X_1 + \varepsilon_2$.

Since the linear relationship between X_1 and X_2 is similar to that between X_1, X_2 and Y, the strength of multicollinearity can also be measured in terms of the R^2 between X_1 and X_2. The level of multicollinearity will be denoted by R_{12}^2. Just like the random noise, three levels of multicollinearity will be examined. Low multicollinearity occurs when R_{12}^2 is approximately 30%. Medium multicollinearity occurs when R_{12}^2 is around 60% while high multicollinearity corresponds to an R_{12}^2 of about 90%.

Just like the previous experiments, each level of multicollinearity will be embedded with three levels of random noise. This will create a total of nine cases (3 levels of multicollinearity x 3 levels of noise) to be investigated. Each case will then have 100 models estimated from samples of size 60. These models will be tested using samples of size 100. The three cases and their corresponding labels are:

1. Low multicollinearity [MX2LOW]
2. Medium multicollinearity [MX2MED]
3. High multicollinearity [MX2HIGH]

Overall, this study will cover 84 cases. Forty-two cases will focus on the functional forms, 9 cases will look at the sample size, 24 cases will examine outliers, and 9 cases will study multicollinearity. Each case will present 100 estimation samples. These samples are used to obtain 100 model estimates for each case using any given model estimation technique. This study will look at three different techniques for obtaining the estimates. The estimates derived using the three techniques will be evaluated and compared with respect to their forecasting accuracy.

3.2 Three Estimation Techniques

This study investigates the training and performance of neural networks in forecasting data with unknown functional forms, different levels of random noise, different sample sizes, one or more outliers, or several collinear variables. For each of the samples for model estimation that will be generated, a neural network model will be obtained. To evaluate the forecasting accuracy of the neural network model, two other model estimates will be obtained. One model will be derived using linear regression while the other will be estimated using Specht's GRNN approach (Specht, 1991).

3.2.1 The linear regression estimate

The linear regression estimate will be based on a model that uses the true functional form of the data. In other words, the estimated linear regression model will be obtained after the appropriate transformation has been performed on the data. The regression model thus represents the ideal estimate that should be obtained from the training data. Its accuracy should be considered as the best that can be achieved for the given set of data.

For example, suppose $f(X) = X^2$ was the functional form used to generate a given estimation sample, then the linear regression model will be given by

$$Y = \beta_0 + \beta_1 * X^2 + \varepsilon.$$

The estimates of β_0 and β_1 will be obtained by applying the method of least squares on the sample.

3.2.2 The GRNN estimate

Since the true models are linear, the neural network estimate represents one way to misspecify the true model. Whether the accuracy of the misspecification is close to that of the true model will be the subject of this study. Another way of obtaining a forecast is to use the GRNN technique proposed by Specht (1991). In Specht's network, the forecast is computed using what amounts to a "nearest-neighbor" approach. The forecast for an input vector **X** will be a weighted average of the outputs in the training sample. The closer an input vector is to **X**, the larger the weight of its output.

To illustrate the GRNN technique for the univariate case, let us assume that the training sample consists of the points (x_1, y_1), (x_2, y_2), ... (x_n, y_n). Suppose a forecast is required for the input value x_{new}. The forecast for x_{new}, $y(x_{new})$, will be obtained from

$$y(x_{new}) = \frac{\sum_{i=1}^{n} y_i \exp(-(x_i - x_{new})^2 / 2\theta^2)}{\sum_{i=1}^{n} \exp(-(x_i - x_{new})^2 / 2\theta^2)}$$

The value of θ called the smoothing constant, is chosen prior to the computation. The larger the value of θ, the smoother the surface produced by the estimate will be.

To determine the value of θ from among a set of possible values, Specht suggests a holdout technique. In this technique, one point in the training sample is chosen as the holdout point. The remaining n-1 points are then used to provide a forecast to the holdout point using a particular value of θ. This is repeated until all points in the training sample have been used as a holdout point. The value of θ that gives the smallest average error is chosen.

In this study, instead of choosing θ from among a set of values, a golden section search will be performed to find the best value of θ in the interval (0, 10). The GRNN model obtained can then be considered an "optimal" estimate since it uses the best value of θ in the interval (0, 10).

For example, let us assume that an estimation sample consists of the two points (5.33, 2.06) and (3.72, 2.12). The GRNN forecast for x_{new} will be given by $y(x_{new})$ which is equal to

$$\frac{2.06*\exp(-(5.33-x_{new})^2/2\theta^2) + 2.12*\exp(-(3.72-x_{new})^2/2\theta^2)}{\exp(-(5.33-x_{new})^2/2\theta^2) + \exp(-(3.72-x_{new})^2/2\theta^2)}$$

For $\theta = 0.7$ and $x_{new} = 4.5$, then $y(x_{new}) = 2.16/1.03 = 2.09$.

Just like the regression estimate, the GRNN estimate is obtained using a non-iterative process. Only one pass is needed to obtain the GRNN forecast. The

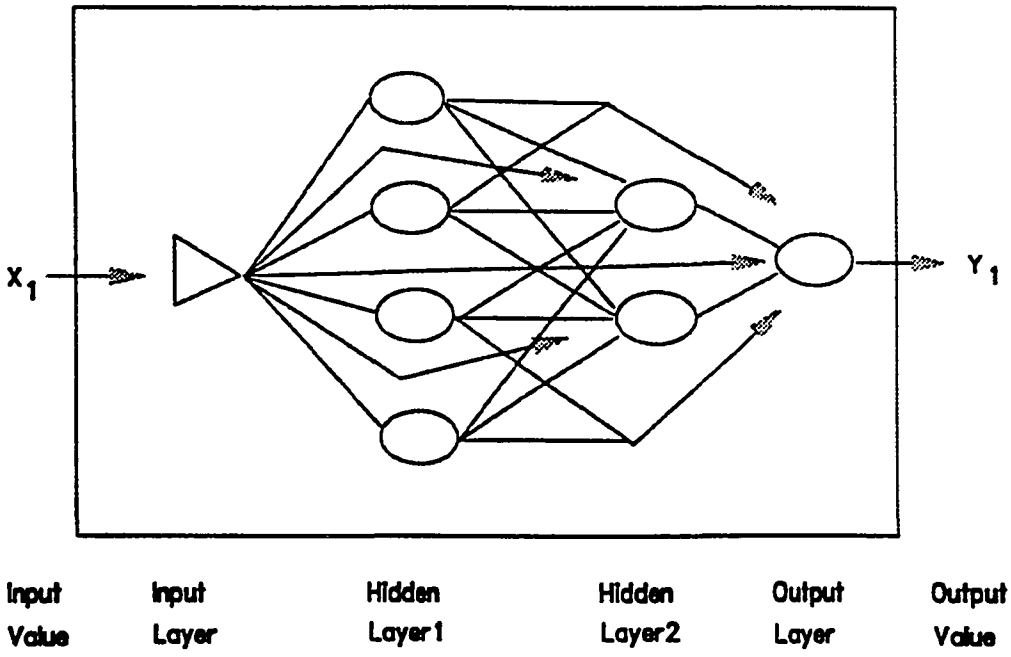
computations involved are simple and short. In addition, the forecasting accuracy of the GRNN estimates will also be compared to that of the neural network model. Although the true regression model is expected to be more accurate than the neural network model, the same cannot be said of GRNN models over neural network models. Actually, neural networks may have the edge over GRNN models since neural networks process data more thoroughly than GRNN models.

3.2.3 The neural network estimate

One major difference between regression models and GRNN models on one hand and neural network models on the other hand is that the former are usually static models while neural network models are transformed continuously during the estimation process. Because of this, the estimation of a neural network requires a large number of passes through a sample. Each pass is called an epoch. With each pass, the values of the parameters in a neural network are adjusted to obtain a better fit of the estimation sample. It is not surprising for some neural networks to require hundred of thousands of epochs before an acceptable estimate is obtained. For a review of the components of a neural network, the reader is again referred to Appendix A.

In all estimations, the neural network estimate will start with two hidden or middle layers. The first hidden layer will contain four nodes while the second will have two nodes. The top layer or output layer will always contain one node. The bottom layer or input layer will contain the number of nodes equal to the number of predictor or independent variables. In the univariate case, this network will have one input node, four nodes in the first hidden layer, two nodes in the second hidden layer and one output node. This configuration, commonly referred to as a 1-4-2-1 network, is illustrated in Figure 3.1. For data with two input variables, a 2-4-2-1 network will be used.

Figure 3.1
A 1-4-2-1 Neural Network



Each node in the neural network is responsible for processing the signals it receives and for sending its output to the correct nodes. The outputs are produced after the net input has been subjected to an activation function. As indicated in Appendix A, this study will use the logistic function as the activation function. Since the output of the logistic function is limited to the range (0, 1), a small amount of scaling will be required to make the output values in the estimation sample compatible with the outputs of a neural network.

3.2.4 Scaling the data

To scale any set of output values into the range (0, 1), all that is needed are the minimum and maximum values in the set. The scaled value of the i^{th} output value, Y_i, is given by

$$\text{scaled}(Y_i) = \frac{Y_i - \min(Y)}{\max(Y) - \min(Y)}.$$

Using this formula, the maximum value in the set is mapped to 1 while the minimum value is mapped to 0. All other values will be mapped proportionately between 0 and 1.

Aside from mapping the data values into a range compatible with that of the logistic function, scaling also provides other important benefits. In cases where the output range is very small, scaling can be used to expand the output range in order to magnify the differences between the response values. What may appear as a horizontal line may actually be a series of peaks and valleys once the range has been magnified. Alternatively, what may appear as a simple curve may actually be a function with a cyclical component.

In some cases, it might be more desirable to train the network on the reverse image of the training data rather than the original set. This occurs when the reverse image behaves more like the logistic function than the original curve. In these cases, the maximum value in the data will be mapped to 0, while the minimum value will be mapped to 1. The scaled value for any output Y_i will now be given by

$$\text{scaled}(Y_i) = \frac{\max(Y) - Y_i}{\max(Y) - \min(Y)}.$$

Scaling can be performed on the input values as well as the output values. In this study, scaling will be performed on both input and output values of the data to enhance the training of the network. Aside from scaling, other procedures will also be performed to speed up training and to obtain smaller and more accurate networks.

3.3 Training the Network

The heart of the modeling process for a neural network is its training. Training refers to the process of obtaining a (minimal) network whose set of weights best reflects the behavior of the data. Thus, the sample used for estimating the neural network is also called the training sample or training data. Unfortunately, training usually requires a large amount of time and computing resources and, in some cases, may be unsuccessful in obtaining an optimal network.

As indicated earlier, this study will apply the backpropagation method in training neural networks. Backpropagation adjusts the weights and biases in the network so that the error or cost function E given by

$$E = \frac{\sum (Y_i - O_i)^2}{n}$$

is minimized. In this cost function, Y_i is the i^{th} target value in the estimation sample while O_i is the corresponding network forecast. Note that E is actually the mean squared error (MSE) of the output from the model estimate. For a more thorough discussion of the mechanics of training a neural network, the reader is referred to Appendix B.

Unfortunately, as White (1989b) pointed out, backpropagation is inefficient. To compensate for this inefficiency, White suggested the application of a nonlinear optimization method to the network after backpropagation has been performed. This optimization method is intended to fine tune the network derived by backpropagation.

Consequently, the nonlinear optimization method can also be used prior to backpropagation. The same method can be used to fine tune the initial network that backpropagation starts with. This approach should shorten training significantly if the nonlinear optimization method requires fewer computations than those used by backpropagation. This study will show the results when a particular nonlinear optimization method, called the downhill simplex method, is used before and after backpropagation.

3.3.1 The downhill simplex method

Before any training can be performed on the network, the network needs to be given a starting set of weights for the connections and biases for the nodes. The choice of starting values can be crucial. If a "good" set of weights and biases is chosen, the amount of training can be significantly reduced. However, in spite of its importance, the initial set of weights and biases is usually chosen randomly.

One way to obtain a "good" set of weights and biases for backpropagation to start with is to apply a nonlinear optimization procedure on the randomly chosen initial set. This optimization procedure should be simpler, faster and more efficient than backpropagation. The "improved" set of weights and biases would then be closer to the set that minimizes the cost function than the randomly chosen set. This "improved" set would require, at worst, the same number of iterations for backpropagation to train the network.

The downhill simplex method, due to Nelder and Mead (1965), was chosen to perform the initial weight improvement. The simplex method contracts, expands or reflects the polyhedron represented by the set of weights and biases so as to yield a minimum value of the cost function. The method is quick and simple since it requires

only function evaluations, not derivatives. Depending on the size of the network, the simplex method is at least one level of magnitude faster than backpropagation. For a more detailed discussion of the downhill simplex method, the reader is referred to Nelder and Mead (1965) and to Press et al. (1988). The C language code that implements the simplex method presented in Press et al. (1988) has been adapted for use in this study.

After the simplex method has improved the initial set of weights and biases of the neural network, backpropagation is applied. Once backpropagation has converged to a set of weight estimates, the simplex method can be applied a second time to fine tune the final set of weights and biases. If backpropagation converged to a local minimum or if it did not converge at all, the simplex method will be able to obtain a better set of parameters. If the backpropagation method obtained an optimal set of weights and biases, then the simplex procedure will not need to perform any fine-tuning. The resulting estimator from this two-step procedure has been shown to be asymptotically equivalent to "pure" nonlinear least squares estimators (White, 1989a). In addition, more powerful hypotheses can be proposed and tested on the network.

As a supplement to backpropagation, the downhill simplex provides a quick and easy way of offsetting the inefficiency of backpropagation. In the front end, the downhill simplex will be used to improve the randomly chosen initial set of weights and biases needed by backpropagation. In the back end, it will be used to fine-tune the final estimates produced by backpropagation. This relatively simple addition can be shown to significantly shorten the training and improve the accuracy of neural networks.

3.3.2 Automating the adjustment of the learning rate

Even with the advantages provided by the simplex method, backpropagation remains a long, computationally intensive process. One factor that contributes to the length of the

training time is the number of interruptions made by the user during training. These interruptions are necessary in order to update the values of some of the parameters, most notably the learning rate.

To make backpropagation more efficient, White (1989b) suggests that the learning rate, denoted by τ, be adapted dynamically. He warns that unless the learning rate eventually declines to zero, backpropagation will fail to settle down to the optimal values of the parameters. In most programs, the learning rate is modified by interrupting training and prompting the user for the new value. This, of course, slows down training significantly. A better approach would be to let the program change the learning rate automatically based on how τ has increased or decreased the cost function being minimized.

One such scheme is offered by Hertz et al. (1991). Under this scheme, the learning rate is changed by an amount given by $\Delta\tau$, where

$$\Delta\tau = \begin{cases} +a & \text{if } \Delta E < 0 \text{ consistently;} \\ -b\tau & \text{if } \Delta E > 0; \\ 0 & \text{otherwise.} \end{cases}$$

In the above expression, ΔE is the change in the cost function, and a and b are appropriate constants. ΔE is based on several steps and may be a weighted moving average. The idea is to increase τ by a constant amount if the last k steps have decreased the cost. If the cost has increased, then the step overshot the cost minimum and τ should be decreased geometrically. When ΔE reaches zero, the training is terminated.

One modification to the above rule is to increase τ by a geometric amount instead of a constant amount. This is because τ may assume the value where $a = b\tau$. When this happens, the above rule will alternately increase and decrease τ by a resulting in τ oscillating indefinitely. To avoid this condition, the rule is changed so that τ is

increased by $a\tau$ if $\Delta E < 0$. As long as $0 < a < b < 1$, τ will be prevented from oscillating indefinitely.

For practical purposes, training will be terminated once ΔE reaches a specified nonzero tolerance level c. This tolerance level will also be used to determine the bounds to test whether τ should be increased or decreased. In this study, this tolerance level will be fixed at 0.01%. The gradient is considered oscillating if the percentage increase in the errors is less than 0.01%. When the percentage decrease in errors is greater than 0.01%, significant improvements are being made. Otherwise, training may be terminated.

In this study, a will be fixed at 0.3, and b will be fixed at 0.5. This means that the update rule for τ will be

$$\Delta\tau = \begin{cases} +0.3\tau & \text{if } \Delta E < -0.0001 \text{ consistently;} \\ -0.5\tau & \text{if } \Delta E > +0.0001; \\ 0 & \text{otherwise.} \end{cases}$$

The starting value of τ will be 2.0 and should be large enough to enable the gradient to escape shallow valleys and focus more on the search for a global minimum. When the gradient is consistently decreasing the errors of the estimate, the learning rate will be increased by 30% to speed up convergence. On the other hand, when the gradient seems to be oscillating back and forth across a minimum, the learning rate will be reduced by half in order to get closer to the minimum. When the change in the cost is less than 0.01%, training may be terminated or some other parameter may be readjusted. One of the parameters that may be changed would be the coefficient that controls the size of the network.

3.3.3 Finding the minimal network configuration

The modifications aimed at making backpropagation more efficient result in shorter training times and estimates that better fit the training data. However, this hardly ensures

good generalization beyond the training data. This means that the model estimate may fit the estimation sample well but will poorly predict the test sample. This occurs when there are too many free parameters in the network. Then, the network learns to fit the noise in the data but becomes very poor in interpolation and extrapolation. Thus, an unnecessarily large number of weight parameters in a network results in poor generalization. This condition is called overfitting. It is therefore desirable to find algorithms that not only optimize the weights in a network, but also optimize the network itself (Hertz et al., 1991, p.156).

The problem of overfitting can be approached two ways. First, overfitting can be avoided if the estimation sample is large compared to the number of weights in the network. A rule of thumb often cited is that the number of training patterns should be at least double the number of weights. This will require the weights to concentrate on the significant features of the data. However, for those situations where data is difficult to obtain, this constraint is quite restrictive (Weigend et al., 1990).

A similar method applies when the number of weights in the network is of the order of the size of the estimation sample; in this case, the network is said to be oversized. One strategy is to stop the training of an oversized network as soon as the network has extracted all the useful information in the data and is beginning to fit the sampling noise (Weigend et al., 1990). This strategy requires that the estimation sample be split into two sets: an approximation set used for determining the values of the weights and biases, and a validation set used for monitoring the performance of the network. As long as the performance on the validation set improves, training continues. When improvement stops, training is terminated. This will prevent the network from using many of its degrees of freedom in fitting the noise.

The second strategy aims to remove or incapacitate those nodes and branches that are found to be nonessential in modeling the sample. Two techniques, pruning and weight-elimination, address the problem of overfitting while searching for the minimal network architecture. The minimal network is defined to be the smallest network able to fit the training data. The minimal network is expected to generalize the best since it has extracted the smallest amount of noise from the data.

In the network in Figure 3.1, each node connects to every other node lying in an upper layer. This structure gives the network the highest number of possible connections while limiting the propagation of the signals to the forward (or upward) direction. That is, a node cannot send its output to a node in the same layer or a lower layer. With a large set of connections, the initial configuration given in Figure 3.1 should be more than sufficient to model most estimation samples. During training, this oversized network will be transformed, hopefully, into a minimal network by removing unnecessary connections and nodes.

Pruning (Sietsma and Dow, 1991) involves the removal of nodes or connections that are redundant or unnecessary. The procedure starts with an oversized network. The oversized network is trained to obtain the best fit. Afterward, the network is searched for nodes and connections that duplicate the functions of other nodes and connections. These redundant nodes and connections are removed accordingly. The corresponding weights and biases are adjusted so that the pruned network provides a performance similar to that of the unpruned network. The network is then retrained. This cycle of pruning, weight adjustment and retraining is repeated until the minimal network is obtained.

Sietsma and Dow (1991) provide a number of indicators to determine which nodes and connections should be removed. They also give the adjustments on the remaining

nodes and biases that will be needed to compensate for the lost parameters. This ensures that the pruned network will provide at least the same level of performance as the unpruned network.

Unfortunately, pruning requires at least one order of magnitude of processing more than the training. This is because the optimization problem is more computationally complex than the learning problem by an order of magnitude (Thodberg, 1991). One way of reducing this complexity is by using weight elimination.

Weight elimination (Weigend et al., 1990) seeks to obtain the minimal network by penalizing the network for non-useful connections. Each connection w_{ij} is given the tendency to decay to zero. This is accomplished by adding a complexity term to the cost function. Thus, the cost function will be composed of the mean squared errors and a complexity term. The new cost function E will be given by

$$\sum (Y_i - O_i)^2 \quad + \quad \delta \quad \sum \frac{w_{ij}^2}{1 + w_{ij}^2}$$

where

Y_i is the target output for the i^{th} point,

O_i is the corresponding network output,

w_{ij} is the weight of the connection from node j to node i, and

δ is the coefficient of decay (also called the coefficient of complexity).

The second term penalizes the network for complexity since the more connections there are in the network, the higher will be the cost. The coefficient of decay δ represents the relative importance between the MSE and network complexity in determining the cost of the network. The larger the value of δ the more the network will be penalized for complexity rather than inaccuracy.

The introduction of the complexity term will change the value of the weight w_{ij} by an amount given by

$$\tau \left\{ \begin{matrix} \text{change due} \\ \text{to the} \\ \text{gradient} \end{matrix} \right\} + \alpha \left\{ \begin{matrix} \text{last} \\ \text{change} \\ \text{made} \end{matrix} \right\} + \frac{(2\tau\delta)\, w_{ij}}{(1 + w_{ij}^2)^2}$$

where τ is the learning rate, α is the momentum value, and δ is the complexity coefficient. Unless the weights are reinforced, most of the connections will be effectively removed when their weights decay to zero. The network is then simplified by pruning off those nodes with identical weights or those with zero weights. Pruning also shortens training, since fewer nodes will mean less processing of inputs during training. Pruning would not be difficult to automate since it can be implemented by a subroutine that searches for and prunes off nodes with identical weights or zero weights.

In this study, the coefficient of decay will start at 0.0, increase up to 0.20 in increments of 0.05, and then decrease to 0.0 in decrements of 0.05. At the initial value of 0.0, backpropagation will obtain a full model estimate(1-4-2-1 network or 2-4-2-1 network) that may overfit the estimation sample. The network is then retrained with a higher coefficient of decay in order to force the network to eliminate unnecessary nodes and connections. These nodes and connection are then removed prior to the next increase in decay. This is repeated until the coefficient reaches 0.20. After this, the coefficient is gradually decreased to 0.0 in order to gradually increase the accuracy of the network. The network estimate obtained at the end of this process therefore represents a model that strives to achieve the highest accuracy with the minimum complexity.

A diagram illustrating the training process is shown in Figure 3.2. After the estimation sample has been chosen and scaled, a neural network with two hidden layers is initialized with a randomly chosen set of weights and biases. The downhill simplex is

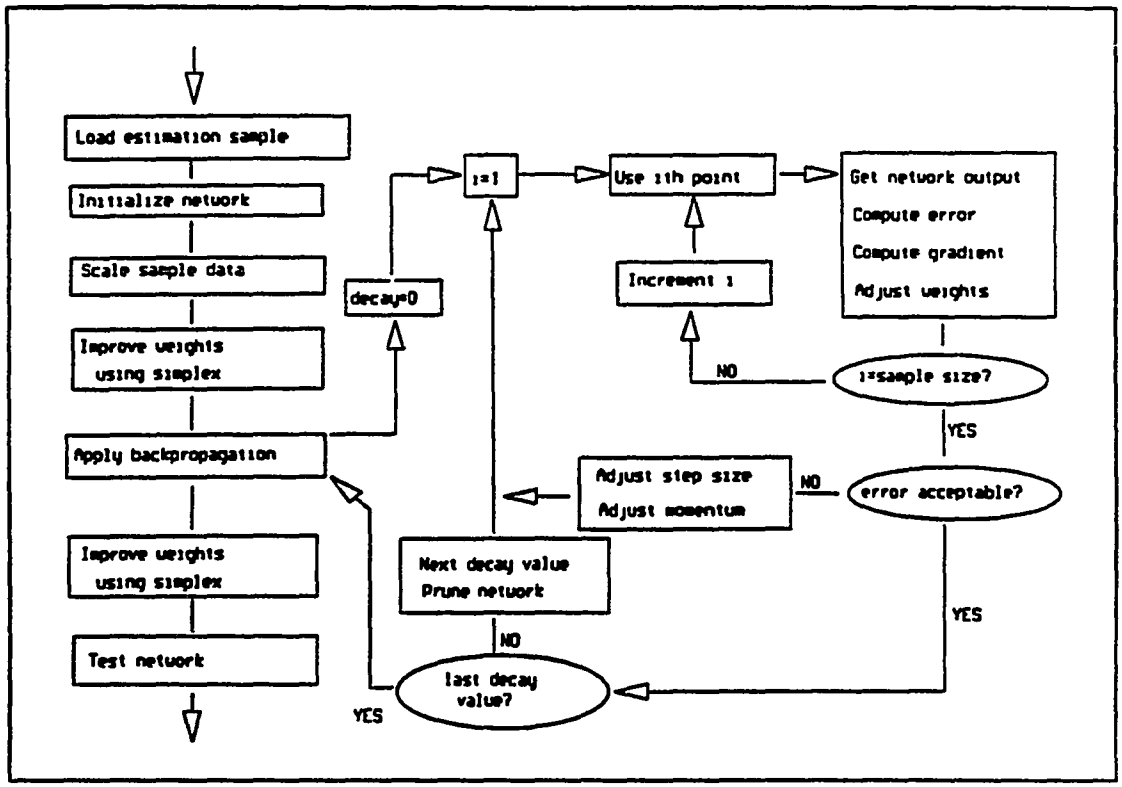
first applied to improve the weights and biases. Afterward, backpropagation is performed. Backpropagation involves the repeated presentation of the estimation sample until a sufficiently accurate and small network is obtained. The network that backpropagation obtains is then improved by using the downhill simplex a second time.

To summarize, the simulation and modeling of each set of characteristics will be accomplished as follows:

1. One hundred estimation samples and three testing samples will be generated to reflect a given set of characteristics.
2. For each of the 100 estimation samples;
    a. A neural network estimate will be obtained using the estimation sample. Statistics will be compiled on the training.
    b. The estimated neural network model will be tested using the three testing samples.
    c. The true model estimate incorporating the ideal transformation will be obtained using the estimation sample. This estimate will then be tested using the three testing samples.
    d. A "nearest-neighbor" model estimate will be obtained using the estimation sample and tested using the three testing samples.
3. The data obtained from one hundred estimation samples will be summarized.
4. The forecasting performance of the three estimates will be compared.
5. The data on the training of the neural net will be analyzed.

Since the initial condition for training a network is crucial in determining the outcome, a description of the initial condition used in this study is given in Appendix D. This description should allow readers to duplicate or verify the results of this study using a neural network program that implements backpropagation and weight elimination. However, the reader is reminded that the statistics on performance accuracy and training presented in the analysis are average values covering 100 model estimates. The performance of any individual model estimate should fall within a reasonable range of these average values.

Figure 3.2
Neural Network Training



3.4 Method of Analysis

The model estimates and forecasts obtained for the different data characteristics will be analyzed using simple statistical analysis. The analysis will consist of two parts, performance evaluation and training assessment. The former will focus on measuring and comparing the performance of the neural network against those the linear and GRNN models. The latter will assess the effects of data characteristics on the training of the neural nets. Relationships between the network and the features of the data being modeled will be established.

3.4.1 Performance evaluation

The accuracy of the forecasts made by the neural network, linear and GRNN models will be measured using the mean absolute percentage error (MAPE) of the model over each of the testing samples. The absolute percentage error of a forecast is given by

$$APE = \frac{|\text{observed value} - \text{forecast value}|}{\text{observed value}} \times 100\%$$

This value is then averaged over the entire testing sample to obtain the MAPE. The neural network performance will be compared with that of the regression estimate to determine how closely the neural network forecast approaches the performance of the best model. The neural network performance will also be compared to that of the GRNN model to determine which of these misspecifications provides better forecasts. A test of hypotheses will be performed on the comparisons to determine if the difference in performance is significant.

For each case that will be considered, 100 samples for model estimation will be used. For each estimation sample, a neural network model, the true regression model and a GRNN model will be estimated. To compare the performance of the neural network model to that of the true regression model in any given test (regular, noiseless or extended), let

- $MAPE(NN_i)$ be the MAPE of the neural network model derived from the i^{th} estimation sample,
- $MAPE(REG_i)$ be the MAPE of the true regression model derived from the i^{th} estimation sample, and
- $d_i = MAPE(NN_i) - MAPE(REG_i)$.

Thus, d_i represents the difference in forecasting accuracy between the neural network estimate and the true regression model. Positive values for d_i indicate that the true regression model is more accurate while negative values will mean otherwise.

Since there are 100 estimation samples, there will be 100 values for the d_i's. Let d_{ave} and s_d be the mean and standard error of this sample of differences. Let D be the unknown true difference between the two estimates. To test whether the true regression estimate is significantly more accurate than the corresponding neural network estimate, the hypotheses will be

$$H_0: \ D > 0 \quad \text{versus} \quad H_1: \ D \leq 0.$$

Since the sample size is large (n > 30), the test statistic z will follow a standard normal distribution where

$$z = d_{ave} / s_d .$$

This value will be referred to as the z-statistic. The critical values for this test will be 1.65 for a significance level of 5% and 2.33 for a 1% level of significance.

In the same manner, the difference in performance between the neural network model and the GRNN model will be given by

$$d_i = \text{MAPE}(NN_i) - \text{MAPE}(GRNN_i)$$

where $\text{MAPE}(GRNN_i)$ is the MAPE of the GRNN model derived from the i^{th} estimation sample.

Negative values for d_i will mean that the neural network model made smaller errors than the GRNN model. To test whether the forecasts made by the neural network estimate are more accurate than those made by the GRNN estimate, the hypotheses will be

$$H_0: \ D < 0 \quad \text{versus} \quad H_1: \ D \geq 0.$$

Again, the sample size is large (n > 30) so that the test statistic z will follow a standard normal distribution where

$$z = d_{ave} / s_d .$$

Since the attention is on the negative differences, the critical values for this test will be -1.65 for a significance level of 5% and -2.33 for a 1% level of significance.

These tests of hypotheses will be performed on the results of the forecasts for the regular, noiseless and extended test samples for each of the 84 cases covered in this study. The z-statistic, along with the average MAPEs of the models being compared will be tabulated and analyzed in the next two chapters.

3.4.2 Training assessment

In assessing the training of the neural network, focus will be put on the length of the training and the size of the resulting network. The length of training will be assessed from the number of iterations or epochs required for the backpropagation method to obtain the network estimate. The number of iterations used by the simplex method before and after backpropagation will also be considered.

One measure of the complexity of the network is the number of parameters that the network requires. The bigger the network, the more nodes and branches will be present, and therefore, the more weights and biases will be estimated. Since all network estimation starts with the same 1-4-2-1 network (2-4-2-1 for the bivariate case), 28 values for the weights and biases are needed at the start of training (35 for the 2-4-2-1 network). With pruning and weight elimination, this number will be reduced while keeping the accuracy of the model as high as possible. The number of parameters in the final network will indicate how complex a network needs to be to model a given set of data characteristics.

The results of the comparisons between the different models and the evaluation of the training process will be presented in the next three chapters. The discussion will start with the presentation of results of the main experiment. The results from this experiment

center on the different functional forms and the level of random noise. The analysis of performance will always begin with comparisons between the neural network and the linear regression model. Then, differences between neural networks and the GRNN model will be discussed.

CHAPTER 4

DISCUSSION OF RESULTS: THE MAIN EXPERIMENT

This chapter will present the results of the main experiment where the focus is on the functional form of the model and the level of random noise in the estimation sample. The main experiment covers half of the 84 cases considered in this study. The 42 cases are divided into three sections. Section 4.1 will discuss the modeling of data generated by five simple functional forms. The results of tests comparing the performance of the neural network model against that of four other models will be examined. Section 4.2 will present the analysis for data generated using more complex univariate functional forms while Section 4.3 will explore simple bivariate functional forms.

4.1 Simple Functional Forms

Table 4.1 presents the MAPEs of the NN model, the MAPEs for the true regression model and the z-statistic of the differences when estimating data generated by the five simple functional forms described in the methodology chapter. The MAPE values are shown for each combination of noise level and test sample. The parenthesized value under the MAPE of the neural network model represents the standard error of the MAPE. The performance comparison for each functional form consists of three rows and three groups of columns. Each row corresponds to the level of noise present in the sample used to derive the model estimate. Remember that every sample used for model estimation contained either high noise, medium noise or low noise. On the other hand, the columns indicate the performance of the model during testing. All model estimates are tested on regular data, noiseless data and extended data.

Table 4.1
MAPES for the Simple Functional Forms
Neural Network (NN) vs. True Regression Model (Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
SQUARED	High	78.10 (2.68)	47.28	11.5	66.88 (3.28)	15.68	15.6	70.15 (3.51)	23.81	13.2
	Med	54.73 (2.71)	26.16	10.6	46.92 (3.41)	8.39	11.2	49.31 (3.56)	12.54	10.1
	Low	12.93 (0.68)	11.20	2.6	5.12 (0.80)	2.58	3.2	7.20 (0.74)	3.66	4.7
LINEAR	High	13.64 (0.06)	13.16	9.0	7.42 (0.19)	7.16	1.7	9.85 (0.26)	9.77	0.4
	Med	10.43 (0.04)	10.11	11.2	4.06 (0.14)	3.81	2.2	5.45 (0.17)	5.12	2.4
	Low	5.41 (0.03)	5.01	18.3	1.71 (0.07)	1.16	8.5	2.60 (0.10)	1.46	13.2
SQRT	High	6.05 (0.03)	5.91	6.1	3.34 (0.08)	3.22	1.9	4.45 (0.10)	4.25	2.5
	Med	4.70 (0.02)	4.58	9.9	1.81 (0.06)	1.70	2.2	2.54 (0.07)	2.21	5.9
	Low	2.53 (0.01)	2.29	24.9	0.87 (0.03)	0.53	10.6	1.36 (0.03)	0.65	19.3
LOG	High	9.31 (0.03)	9.05	11.4	4.90 (0.15)	4.55	3.4	7.51 (0.21)	6.75	5.7
	Med	6.82 (0.02)	2.34	60.7	2.64 (0.10)	2.39	3.6	4.38 (0.13)	3.46	11.0
	Low	3.49 (0.02)	0.78	80.8	1.36 (0.05)	0.75	13.2	2.75 (0.07)	1.00	25.4
RECIP	High	147.07 (5.49)	284.95	-20.7	92.42 (1.52)	62.77	13.6	95.23 (1.56)	78.12	6.9
	Med	26.43 (0.42)	24.26	4.9	17.18 (0.75)	6.70	15.0	20.28 (0.85)	8.17	15.4
	Low	5.68 (0.09)	5.15	5.8	1.98 (0.12)	1.11	6.6	2.84 (0.13)	1.28	10.1

4.1.1 NN model versus true regression model

According to Table 4.1, the true regression model gave significantly more accurate estimates than the neural network model in almost all cases. This was to be expected of the correct regression model. Surprisingly, the NN model was able to keep within 3% of the MAPEs of the true regression model for those cases where the estimation samples had low noise.

Among the simple functional forms, the LOG, SQRT and LINEAR forms were best approximated by the neural network. From Table 4.1, the MAPE values of the NN estimate for these functional forms were within 2% of the MAPE values for the true model in almost all of the cases. Only the approximation of regular LOG data produced differences in MAPEs exceeding 2%. For the SQRT and LINEAR forms, the differences in MAPEs were small enough that in three of the six cases involving high noise ($z = 1.9$, 1.7, 0.4), the differences were insignificant.

In only one case did the neural model provide more accurate estimates than the true regression model. This was the model obtained from the RECIP data with high noise and tested using regular samples ($z = -20.7$). Even under medium and low noise, the NN models continued to obtain MAPEs for the regular data that were close to the MAPEs of the true model. However, when it came to estimating the noiseless or extended RECIP data, the NN model did not do well for high and medium noise. Only under conditions of low noise did the NN model come within 2% of the true MAPE.

The tests where the NN model had the least forecasting accuracy involved the SQUARED data. When the estimation data contained a high level of noise, the errors were large -- almost three times that of the true regression model. Only at the low noise level did the NN MAPEs come to within 4% of the true MAPEs.

4.1.2 NN model versus GRNN model

In Table 4.2, the neural network model shows the same advantage over the GRNN model that the true model displayed over the neural network model in Table 4.1. Since the NN model has already been shown to be almost as good as the true model in approximating the LOG, SQRT and LINEAR data, the neural net model had significantly smaller MAPEs for these forms compared to the GRNN model. Consequently, Table 4.2 shows negative z values for these forms.

In addition, while the NN MAPEs seemed to improve with decreasing noise for LOG, SQRT and LINEAR functional forms, the GRNN MAPEs remained at the same level. This is consistent with the fact that the GRNN estimates are weighted averages. For an average, positive and negative errors cancel out. The MAPEs of the GRNN did not only come out unchanged among the three noise levels but also came out the same for the regular and noiseless test data. Because of this, the difference in MAPEs between the NN and GRNN models increased as the noise level decreased. This is indicated by z-statistics becoming more negative with less noise.

Since the NN models did not estimate as accurately with the RECIP and SQUARED data as the models did with the LOG, SQRT and LINEAR data, their performance advantage over the GRNN models was expected to diminish. Thus, the MAPEs that the GRNN model received for the RECIP and SQUARED tests were lower than the NN MAPEs in most of the cases involving medium and high noise. It was only in predicting regular RECIP data where NN models were significantly more accurate ($z = -24.6$ and $z = -22.0$). The neural network model's performance improved only after noise was decreased to the lowest level. At this noise level, the neural network model obtained MAPEs that were 3 to 15 times smaller than those of the GRNN model.

Table 4.2
MAPES for the Simple Functional Forms
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat
SQUARED	High	78.10 (2.68)	57.62	7.6	66.88 (3.28)	36.37	9.3	70.15 (3.51)	51.34	5.3
	Med	54.73 (2.71)	43.28	4.2	46.92 (3.41)	36.36	3.1	49.31 (3.56)	51.37	-0.6
	Low	12.93 (0.68)	37.86	-36.7	5.12 (0.80)	36.36	-39.1	7.20 (0.74)	51.35	-59.4
LINEAR	High	13.64 (0.06)	16.64	-51.0	7.42 (0.19)	16.64	-49.2	9.85 (0.26)	22.29	-47.7
	Med	10.43 (0.04)	16.64	-173.2	4.06 (0.14)	16.64	-91.9	5.45 (0.17)	22.29	-96.6
	Low	5.41 (0.03)	16.63	-352.7	1.71 (0.07)	16.63	-204.2	2.60 (0.10)	22.28	-195.4
SQRT	High	6.05 (0.03)	8.08	-83.6	3.34 (0.08)	8.31	-60.1	4.45 (0.10)	10.78	-60.9
	Med	4.70 (0.02)	8.20	-194.5	1.81 (0.06)	8.31	-108.3	2.54 (0.07)	10.78	-111.1
	Low	2.53 (0.01)	8.29	-282.0	0.87 (0.03)	8.31	-217.9	1.36 (0.03)	10.78	-247.0
LOG	High	9.31 (0.03)	10.79	-56.6	4.90 (0.15)	10.46	-37.2	7.51 (0.21)	15.49	-37.6
	Med	6.82 (0.02)	10.80	-139.6	2.64 (0.10)	10.47	-78.7	4.38 (0.13)	15.49	-81.5
	Low	3.49 (0.02)	10.92	-212.5	1.36 (0.05)	10.47	-167.9	2.75 (0.07)	15.49	-165.9
RECIP	High	147.07 (5.49)	379.78	-24.6	92.42 (1.52)	26.52	34.5	95.23 (1.56)	29.24	34.8
	Med	26.43 (0.42)	37.22	-22.0	17.18 (0.75)	16.30	1.1	20.28 (0.85)	20.37	-0.1
	Low	5.68 (0.09)	16.16	-82.7	1.98 (0.12)	15.94	-93.4	2.84 (0.13)	20.13	-98.1

For the RECIP and SQUARED data, the GRNN showed improvements in MAPE with decreasing noise only when estimating regular data. This is in contrast to the almost static movement in MAPEs in all cases of the LOG, SQRT and LINEAR tests. The

improvement in MAPEs became evident only during the change from high to medium noise. No significant change in MAPEs occurred with the decrease from medium to low noise.

When estimating noiseless and extended data for the RECIP functional form, the GRNN model produced MAPEs that showed little improvement with decreasing noise. For the same type of tests involving the SQUARED functional form, the GRNN model showed no improvement at all when the noise level was decreased.

4.1.3 NN model versus misspecified regression model

Another set of misspecified models can be obtained by using the "ladder-like" series formed by the simple functional forms. As explained in Chapter 3, the RECIP, LOG, SQRT, LINEAR and SQUARED functional forms represent an ascending "ladder of reexpression" based on the power used in the functional form. For each functional form, the adjacent functional form in the ladder represents a "best" misspecification since the adjacent form is closest to function being approximated. For example, in approximating the SQUARED functional form, the LINEAR model can be considered the best misspecification among the four remaining functions since it is closest to the SQUARED functional form in the ladder.

Two sets of "best" misspecifications can be obtained from the ladder of reexpressions. The first set involves the functional forms positioned (on the ladder) immediately above the function being approximated. Consequently, second set includes those functional forms located (on the ladder) directly below the function being approximated. The former will be called the "upper misspecifications" and the latter will be referred to as the "lower" misspecifications." Table 4.3 gives the "upper" misspecifications for the five simple functional forms while Table 4.4 describes the

Table 4.3
Upper Misspecifications for the Simple Functional Forms

TRUE FUNCTIONAL FORM		UPPER MISSPECIFICATION	
Label	Model	Label	Model
SQUARED	$Y = X^2 + \varepsilon$	CUBIC	$Y = X^3 + \varepsilon$
LINEAR	$Y = X + \varepsilon$	SQUARED	$Y = X^2 + \varepsilon$
SQRT	$Y = \sqrt{X} + \varepsilon$	LINEAR	$Y = X + \varepsilon$
LOG	$Y = \ln(x) + \varepsilon$	SQRT	$Y = \sqrt{X} + \varepsilon$
RECIP	$Y = 1/X + \varepsilon$	LOG	$Y = \ln(X) + \varepsilon$

"lower" misspecifications. In Table 4.3, the CUBIC functional form was added to serve as the upper misspecification for the SQUARED model. Similarly, the INVSQ functional form in Table 4.4 was included to model the RECIP data.

Regression model estimates were obtained using the upper or lower misspecification as the model estimate. The test of hypotheses performed to compare the performance of the neural network model with that of the misspecified regression model was similar to that performed to compare the neural network model with the true regression model. Thus, the difference in performance d_i is given by

$$d_i = \text{MAPE}(NN_i) - \text{MAPE}(MIS_i)$$

where $\text{MAPE}(MIS_i)$ is the MAPE of the (upper or lower) misspecified regression model derived from the i^{th} estimation sample.

Again, d_i represents the difference in accuracy between the neural network estimate and the misspecified regression model. Positive values for d_i indicate that the

Table 4.4
Lower Misspecifications for the Simple Functional Forms

TRUE FUNCTIONAL FORM		LOWER MISSPECIFICATION	
Label	Model	Label	Model
SQUARED	$Y = X^2 + \varepsilon$	LINEAR	$Y = X + \varepsilon$
LINEAR	$Y = X + \varepsilon$	SQRT	$Y = \sqrt{X} + \varepsilon$
SQRT	$Y = \sqrt{X} + \varepsilon$	LOG	$Y = \ln(x) + \varepsilon$
LOG	$Y = \ln(x) + \varepsilon$	RECIP	$Y = 1/X + \varepsilon$
RECIP	$Y = 1/X + \varepsilon$	INVSQ	$Y = 1/X^2 + \varepsilon$

misspecified regression model was more accurate than the neural network model. The null and alternative hypotheses tested were

$$H_0: D > 0 \quad \text{versus} \quad H_1: D \leq 0.$$

The critical values for this test are 1.65 for a significance level of 5% and 2.33 for a 1% level of significance.

Table 4.5 shows the MAPEs of the model estimates for the upper misspecifications and the z-statistics of their differences with the NN models. For the LINEAR data, the NN model had significantly lower MAPEs for the noiseless and extended samples under all noise levels, and also for the regular samples under low noise. For the rest of the functional forms, the results were mixed.

In estimating the RECIP data, the NN model had significantly lower MAPEs than the INVSQ model under conditions of low noise, and for regular data, under conditions of high noise. For the SQUARED data, only the extended test under low noise showed significantly lower ($z = -4.3$) MAPEs for the NN model. Otherwise, the upper

Table 4.5
MAPES for the Simple Functional Forms
Neural Network (NN) vs. Upper Misspecification (UMis)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	UMis (%)	Z Stat	NN (%)	UMis (%)	Z Stat	NN (%)	UMis (%)	Z Stat
SQUARED	High	78.10 (2.68)	47.02	11.6	66.88 (3.28)	16.89	15.2	70.15 (3.51)	27.17	12.3
	Med	54.73 (2.71)	26.56	10.4	46.92 (3.41)	10.40	10.6	49.31 (3.56)	17.57	8.8
	Low	12.93 (0.68)	12.79	0.2	5.12 (0.80)	5.72	-0.7	7.20 (0.74)	10.55	-4.3
LINEAR	High	13.64 (0.06)	13.01	11.2	7.42 (0.19)	7.51	-0.6	9.85 (0.26)	10.71	-4.2
	Med	10.43 (0.04)	10.07	12.7	4.06 (0.14)	4.42	-3.1	5.45 (0.17)	6.54	-7.6
	Low	5.41 (0.03)	5.61	-8.3	1.71 (0.07)	2.44	-11.6	2.60 (0.10)	3.83	-14.3
SQRT	High	6.05 (0.03)	5.88	7.4	3.34 (0.08)	3.25	1.4	4.45 (0.10)	4.41	0.5
	Med	4.70 (0.02)	4.57	9.6	1.81 (0.06)	1.77	1.0	2.54 (0.07)	2.43	1.9
	Low	2.53 (0.01)	2.42	11.2	0.87 (0.03)	0.72	14.7	1.36 (0.03)	1.06	9.2
LOG	High	9.31 (0.03)	9.13	7.7	4.90 (0.15)	4.61	2.9	7.51 (0.21)	7.18	2.5
	Med	6.82 (0.02)	2.40	57.8	2.64 (0.10)	2.46	2.5	4.38 (0.13)	4.02	4.5
	Low	3.49 (0.02)	1.06	85.0	1.36 (0.05)	1.04	8.0	2.75 (0.07)	1.89	15.2
RECIP	High	147.07 (5.49)	309.69	-25.2	92.42 (1.52)	70.22	10.8	95.23 (1.56)	86.40	3.8
	Med	26.43 (0.42)	24.36	4.8	17.18 (0.75)	8.73	12.0	20.28 (0.85)	10.56	12.4
	Low	5.68 (0.09)	6.25	-6.9	1.98 (0.12)	2.74	-6.5	2.84 (0.13)	3.52	-5.2

misspecification gave forecasts that were as good as, if not better than, the NN model. The SQRT data had the same results. For the LOG data, the upper misspecification had significantly higher accuracy than the neural network in all conditions.

A similar set of results is shown in Table 4.6 where the MAPEs of the lower misspecifications are given. For the RECIP data, the NN models again showed equivalent if not better performance than the misspecified regression model when estimating under conditions of low noise, or when estimating regular test data.

For the LOG data, the results were mixed. The NN model had equal or better MAPEs for the noiseless test data. The model had better accuracy when estimating regular data under conditions of high noise or when estimating extended data under low noise. For the remaining cases, the lower misspecification had significantly better performance. Overall, the MAPEs show that the NN model and the lower misspecification had equivalent success with the LOG data. Recall that the upper misspecifications were significantly more accurate than the NN model for this same set of data.

For the SQRT, LINEAR and SQUARED data, the z-statistics in Table 4.6 show that the lower misspecifications had at least as good a performance as the NN model. When estimating noiseless data, the two models had differences that were mostly insignificant. When estimating regular samples, the lower misspecifications clearly displayed the better performance. The only case where the NN model had significantly smaller errors than the lower misspecification was when estimating extended data under low noise ($z = -3.3$).

In summary, the neural network model provides good estimates for data that were generated using the LINEAR, SQRT and LOG functional forms. The MAPEs of the neural network model were found to be within 2% of the MAPE of the true regression model in most of the cases. The accuracy of the NN models was measured and found to be approximately the same as those of the best misspecified regression models, and definitely better than those of the corresponding GRNN estimates.

Table 4.6
MAPES for the Simple Functional Forms
Neural Network (NN) vs. Lower Misspecification (LMis)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	LMis (%)	Z Stat	NN (%)	LMis (%)	Z Stat	NN (%)	LMis (%)	Z Stat
SQUARED	High	78.10 (2.68)	47.91	11.3	66.88 (3.28)	15.31	15.7	70.15 (3.51)	20.55	14.0
	Med	54.73 (2.71)	26.10	10.6	46.92 (3.41)	8.17	11.3	49.31 (3.56)	10.77	10.7
	Low	12.93 (0.68)	11.33	2.4	5.12 (0.80)	5.04	0.1	7.20 (0.74)	9.63	-3.3
LINEAR	High	13.64 (0.06)	13.25	7.1	7.42 (0.19)	7.15	1.7	9.85 (0.26)	9.38	2.3
	Med	10.43 (0.04)	10.16	9.1	4.06 (0.14)	3.84	1.7	5.45 (0.17)	4.72	5.1
	Low	5.41 (0.03)	5.00	17.3	1.71 (0.07)	1.53	2.3	2.60 (0.10)	2.13	4.8
SQRT	High	6.05 (0.03)	5.95	4.3	3.34 (0.08)	3.26	1.2	4.45 (0.10)	4.14	3.6
	Med	4.70 (0.02)	4.60	6.8	1.81 (0.06)	1.80	0.2	2.54 (0.07)	2.17	5.7
	Low	2.53 (0.01)	2.30	20.2	0.87 (0.03)	0.79	2.1	1.36 (0.03)	1.06	7.4
LOG	High	9.31 (0.03)	8.93	15.6	4.90 (0.15)	4.69	1.7	7.51 (0.21)	6.11	8.7
	Med	6.82 (0.02)	2.80	51.6	2.64 (0.10)	2.84	-2.1	4.38 (0.13)	4.04	2.8
	Low	3.49 (0.02)	1.73	29.9	1.36 (0.05)	1.83	-7.2	2.75 (0.07)	3.50	-9.2
RECIP	High	147.07 (5.49)	273.16	-16.8	92.42 (1.52)	54.28	16.3	95.23 (1.56)	69.35	9.5
	Med	26.43 (0.42)	25.24	2.5	17.18 (0.75)	5.73	15.7	20.28 (0.85)	7.42	15.5
	Low	5.68 (0.09)	5.77	-1.0	1.98 (0.12)	3.44	-10.4	2.84 (0.13)	4.51	-10.4

In contrast, the NN models had a good amount of success estimating RECIP and SQUARED data only under conditions of low noise. With high or medium noise, the NN model rarely came close to the performance of the true regression model. Under

these conditions, the neural network was significantly outperformed by the misspecified regression models. Even the GRNN model had MAPEs that were mostly as good as, if not better than, those of the NN model. The neural network's difficulty estimating SQUARED data would carry on to the more complex functional forms.

4.2 More Complex Functional Forms

Table 4.7 presents the comparison of the performance of the neural network and the true regression model with respect to five more complex univariate functional forms. In four of the five forms, the neural network showed errors that were almost as low as those of the true model. For the MICHAELIS, EXRIS, and LOGISTIC forms, the z-statistics showed that the true model had significantly lower MAPEs than the NN model. However, the MAPEs of the NN model stayed within 1% of the regression MAPEs. For the MICHAELIS data, the NN model had MAPEs ranging from 6.48% to 0.94%. For the EXRIS data, the range improved with MAPEs from 6.10% to 0.88%. For the LOGISTIC data, both the NN model and the true model had large MAPEs under conditions of high noise. However, these MAPEs dropped significantly to values under 1% with medium and low noise.

For the XLNX data, the NN network received errors that were not significantly different from those of the true model under conditions of high and medium noise. With low noise, the difference in MAPEs became highly significant. However, an examination of the MAPE values shows that the NN model exceeded the true model by less than 3% under all noise levels. Again, the NN model kept close to the performance of the true model.

Table 4.7
MAPES for the More Complex Functional Forms
Neural Network (NN) vs. True Regression Model (Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
XSQX	High	1041.59 (24.41)	175.95	35.2	2346.08 (52.41)	371.77	39.1	2519.40 (56.40)	529.44	36.9
	Med	38.51 (2.40)	19.79	7.8	30.13 (2.99)	7.00	7.8	31.59 (2.96)	9.81	7.4
	Low	13.70 (1.38)	9.03	3.4	7.61 (1.54)	2.12	3.6	9.16 (1.53)	2.83	4.2
XLNX	High	24.68 (0.62)	22.74	2.8	14.18 (0.87)	11.89	2.2	21.48 (1.06)	19.20	1.8
	Med	19.10 (1.03)	17.04	1.6	8.49 (1.16)	6.17	1.6	12.28 (1.22)	10.05	1.7
	Low	8.94 (0.05)	8.17	25.2	3.06 (0.12)	2.01	7.1	5.88 (0.21)	3.03	11.9
MICHAELIS	High	6.48 (0.03)	6.32	6.3	3.71 (0.10)	3.43	3.6	5.12 (0.12)	4.69	4.9
	Med	5.04 (0.02)	4.89	8.6	2.01 (0.07)	1.82	3.4	2.92 (0.09)	2.44	7.6
	Low	2.72 (0.01)	2.45	25.2	0.94 (0.03)	0.57	11.6	1.58 (0.04)	0.72	21.1
EXRIS	High	6.10 (0.03)	5.95	6.4	3.36 (0.10)	3.22	2.1	4.45 (0.13)	4.27	2.1
	Med	4.72 (0.02)	4.61	7.5	1.84 (0.07)	1.69	3.2	2.48 (0.09)	2.20	5.4
	Low	2.51 (0.01)	2.30	21.9	0.88 (0.03)	0.53	12.0	1.31 (0.04)	0.65	19.0
LOGISTIC	High	12.83 (0.06)	12.34	9.6	7.30 (0.22)	6.17	8.1	8.90 (0.27)	7.89	6.1
	Med	0.88 (0.00)	0.87	4.4	0.35 (0.01)	0.31	3.8	0.47 (0.02)	0.39	5.4
	Low	0.48 (0.00)	0.44	19.6	0.17 (0.01)	0.10	12.6	0.24 (0.01)	0.12	18.5

Just as in the SQUARED data, the NN model exhibited difficulty modeling the XSQX data. This may be due to the presence of a quadratic term in the SQUARED and XSQX functional forms. Under conditions of high noise, the MAPEs of the NN model

were in the thousands of percentage, five to six times the MAPEs of the true model. The NN model had double the errors estimating the noiseless and extended data than it had with the regular data. With medium noise, the MAPEs of the NN model dropped down to around 35%, and with low noise, the MAPEs improved to around 9%. Even at medium and low noise, the MAPEs of the NN model were still 2 to 4 times those of the true model.

With such poor estimates for the XSQX data, the neural network model was significantly outperformed by the GRNN model under conditions of high noise. Table 4.8 shows the performance comparison between neural networks and the GRNN model for more complex functional forms. The MAPEs of the NN model were 9 to 25 times larger than those of the GRNN model. However, when the noise level decreased to medium level, the error difference became mostly insignificant. Then with the noise at low level, the NN model became significantly more accurate than the GRNN model.

The GRNN model also gave better estimates for the LOGISTIC data under conditions of high noise. This occurred for the tests involving the noiseless and extended samples (z= 19.3, 20.9). For the rest of the LOGISTIC data, as well as the XLNX, MICHAELIS and EXRIS data, the NN model gave significantly better estimates than the GRNN model.

The changing levels of noise had no effect on the performance of the GRNN model on three of the forms, but had a dramatic effect on the remaining two. For the XLNX, MICHAELIS and EXRIS data, the MAPEs showed virtually no change among the three levels of noise. There was also little difference between the MAPEs obtained from estimating regular data and those from estimating noiseless data. In contrast, there was a substantial decrease in MAPE for the XSQX and LOGISTIC data when the noise level was dropped from high to medium. The reduction ranged from 30% to 80% of the

Table 4.8
MAPES for the More Complex Functional Forms
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat
XSQX	High	1041.59 (24.41)	114.70	37.6	2346.08 (52.41)	92.14	42.9	2519.40 (56.40)	100.25	42.7
	Med	38.51 (2.40)	34.56	1.7	30.13 (2.99)	32.06	-0.6	31.59 (2.96)	41.21	-3.3
	Low	13.70 (1.38)	32.66	-13.7	7.61 (1.54)	32.05	-15.8	9.16 (1.53)	41.21	-20.9
XLNX	High	24.68 (0.62)	29.10	-9.5	14.18 (0.87)	28.51	-20.1	21.48 (1.06)	45.25	-25.1
	Med	19.10 (1.03)	29.04	-9.7	8.49 (1.16)	28.51	-17.5	12.28 (1.22)	45.25	-27.6
	Low	8.94 (0.05)	28.73	-301.7	3.06 (0.12)	28.50	-205.2	5.88 (0.21)	45.25	-178.5
MICHAELIS	High	6.48 (0.03)	7.82	-43.8	3.71 (0.10)	7.92	-45.1	5.12 (0.12)	10.92	-48.1
	Med	5.04 (0.02)	7.86	-130.7	2.01 (0.07)	7.92	-86.5	2.92 (0.09)	10.92	-94.4
	Low	2.72 (0.01)	7.90	-291.8	0.94 (0.03)	7.92	-211.2	1.58 (0.04)	10.92	-219.0
EXRIS	High	6.10 (0.03)	15.04	-150.6	3.36 (0.10)	15.40	-106.3	4.45 (0.13)	16.35	-87.0
	Med	4.72 (0.02)	15.15	-173.8	1.84 (0.07)	15.39	-139.5	2.48 (0.09)	16.34	-133.1
	Low	2.51 (0.01)	15.31	-186.9	0.88 (0.03)	15.39	-198.3	1.31 (0.04)	16.34	-213.1
LOGISTIC	High	12.83 (0.06)	15.44	-50.6	7.30 (0.22)	3.03	19.3	8.90 (0.27)	3.28	20.9
	Med	0.88 (0.00)	2.14	-124.0	0.35 (0.01)	2.09	-115.3	0.47 (0.02)	2.40	-116.1
	Low	0.48 (0.00)	2.10	-213.2	0.17 (0.01)	2.09	-225.7	0.24 (0.01)	2.41	-274.8

original MAPE. The largest reductions were observed with the estimates on the regular test data. Below the medium noise level, the GRNN exhibited no change in its MAPEs.

4.3 Simple Bivariate Functional Forms

The difference in movement between the GRNN errors for regular data and those for noiseless and extended data are again exhibited in the modeling of the simple bivariate functional forms. Table 4.9 compares the performance of the GRNN model and the NN model for data with two independent variables.

In the table, the MAPEs of the GRNN for the noiseless and extended test samples were not affected by the changes in the noise level. This applies to all four functional forms. In these cases, the NN model had significantly higher accuracy than the GRNN model in almost all of the cases since the NN model is able to take advantage of the decrease in noise. The only case where the GRNN had significantly more accurate estimates than the NN model was for the X1*X2 data under conditions of high noise.

Just as in the XSQX and LOGISTIC functional forms, the GRNN showed improvement in MAPEs only when estimating regular data. However, unlike the univariate forms, most of the improvements in GRNN MAPEs were small but still significant. The only dramatic improvement occurred with the change in the X1*X2 data from high to medium noise. Again, unlike the univariate case, the improvements in MAPEs continued beyond medium data. There was steady improvement from high noise to medium noise and from medium noise to low noise. Even with the steady improvement in estimating regular data, the GRNN model remained significantly less accurate than the neural network model.

Just like the GRNN model, the neural network model exhibited considerable accuracy when estimating regular data. Under conditions of high noise, the performance of the neural network was at least as accurate as the true model. The z-statistics in Table 4.10 bear this out. Table 4.10 gives the comparative performance of the neural network model and the true regression model in modeling the four bivariate functional forms.

Table 4.9
MAPES for the Bivariate Functional Forms
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data NN (%)	Regular Test Data GRNN (%)	Regular Test Data Z Stat	Noiseless Test Data NN (%)	Noiseless Test Data GRNN (%)	Noiseless Test Data Z Stat	Extended Test Data NN (%)	Extended Test Data GRNN (%)	Extended Test Data Z Stat
X1+X2	High	15.01 (0.07)	20.79	-58.5	8.00 (0.22)	15.37	-31.5	10.08 (0.28)	18.72	-28.8
	Med	10.76 (0.07)	18.30	-67.9	4.82 (0.21)	15.35	-45.1	6.11 (0.27)	18.70	-42.5
	Low	5.05 (0.03)	16.26	-29.6	1.85 (0.06)	15.35	-29.4	2.66 (0.08)	18.70	-29.5
X1LOGX2	High	14.61 (0.08)	17.26	-27.6	7.89 (0.24)	15.48	-28.6	10.22 (0.32)	19.71	-26.9
	Med	10.69 (0.05)	16.43	-95.5	4.60 (0.15)	15.48	-71.2	6.08 (0.19)	19.70	-73.3
	Low	5.38 (0.05)	15.74	-197.2	2.28 (0.08)	15.46	-171.2	3.44 (0.11)	19.70	-148.6
X1*X2	High	113.06 (6.10)	104.07	1.5	56.81 (2.32)	27.84	12.6	57.75 (2.27)	33.94	10.5
	Med	41.29 (1.78)	47.68	-3.6	23.75 (2.50)	27.82	-1.6	25.47 (2.49)	33.92	-3.4
	Low	12.25 (0.80)	31.59	-22.6	5.76 (0.83)	27.77	-24.4	7.60 (0.79)	33.87	-29.7
X1/X2	High	13.66 (0.06)	18.11	-60.4	7.59 (0.24)	14.82	-29.8	9.68 (0.31)	17.82	-26.2
	Med	9.84 (0.06)	16.56	-93.6	4.19 (0.15)	14.81	-69.4	5.46 (0.18)	17.81	-66.5
	Low	4.83 (0.03)	15.29	-53.8	1.93 (0.05)	14.80	-54	2.91 (0.07)	17.80	-54.1

When estimating noiseless and extended data, the true model obtained significantly lower MAPEs than the NN model. The only exception occurs with the X1LOGX2 data under high noise, where the difference in MAPEs was not significant. Even then, the neural network model kept to within 3% of the accuracy of the true model for the X1+X2, X1LOGX2 and X1/X2 data.

Table 4.10
MAPES for the Bivariate Functional Forms
Neural Network (NN) vs. True Regression Model (Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
X1+X2	High	15.01 (0.07)	16.43	-14.9	8.00 (0.22)	5.48	8.5	10.08 (0.28)	6.96	8.2
	Med	10.76 (0.07)	10.93	-1.9	4.82 (0.21)	3.03	7.0	6.11 (0.27)	3.79	7.1
	Low	5.05 (0.03)	4.84	5.2	1.85 (0.06)	1.01	8.7	2.66 (0.08)	1.21	11.3
X1LOGX2	High	14.61 (0.08)	14.65	-0.4	7.89 (0.24)	7.41	1.6	10.22 (0.32)	9.66	1.4
	Med	10.69 (0.05)	10.56	2.3	4.60 (0.15)	3.94	3.2	6.08 (0.19)	5.09	3.7
	Low	5.38 (0.05)	4.85	9.8	2.28 (0.08)	1.18	10.9	3.44 (0.11)	1.48	14.6
X1*X2	High	113.06 (6.10)	130.25	-2.7	56.81 (2.32)	10.08	20.2	57.75 (2.27)	13.46	19.6
	Med	41.29 (1.78)	33.18	4.6	23.75 (2.50)	5.58	7.2	25.47 (2.49)	7.33	7.2
	Low	12.25 (0.80)	10.06	2.5	5.76 (0.83)	1.85	4.5	7.60 (0.79)	2.31	6.5
X1/X2	High	13.66 (0.06)	14.50	-11.5	7.59 (0.24)	6.09	5.0	9.68 (0.31)	7.67	5.3
	Med	9.84 (0.06)	10.08	-3.6	4.19 (0.15)	3.30	4.5	5.46 (0.18)	4.10	5.5
	Low	4.83 (0.03)	4.56	7.1	1.93 (0.05)	1.03	11.7	2.91 (0.07)	1.23	16.6

For the X1*X2 data, both the true model and the NN model posted serious inaccuracies at the high noise level. For the regular data, the NN model even had a significant edge with $z = -2.7$. However, for the rest of the comparisons, the true model had significantly better MAPEs. The MAPEs of the NN model were 3 to 5 times larger than the MAPEs of the true model for the noiseless and extended test samples.

Looking back at the whole experiment, we can conclude that the neural network model generally provides estimates that are nearly as accurate as those of the true model. This applies to most of the functional forms under all levels of noise. This capability applies whether the neural network is used to estimate regular data, noiseless data or extended data. For these functional forms, the performance of the neural network is comparable to that of the best misspecifed regression models, and definitely superior to the performance of the GRNN model.

For some functional forms including RECIP, XLNX and X1*X2, the neural network made huge errors when making estimates under conditions of high noise. The errors were in the hundreds or thousands of percentage error. In these cases, the GRNN model usually provided the more acceptable estimate. However, when the noise is limited to the medium and low levels, the accuracy of the neural network improves tremendously, making the NN model more attractive than the GRNN model.

For the functional forms with a quadratic term, SQUARED and XSQX, the neural network showed considerable difficulty in making accurate estimates. This occurred in all three levels of noise. Even though substantial improvements in MAPEs are again achieved by the neural network with a decrease in the level of noise, the MAPEs obtained by the NN model continue to be several times larger than that of the true regression model. Whether the same difficulty is exhibited by the neural network whenever confronted with a functional form with a quadratic term, or one that is concave upward, remains to be explored.

The results showed that the performance of the GRNN model was unaffected by changes in the noise level. This means that the GRNN model usually provides estimates under conditions of high noise that are as accurate as those it would have made under conditions of low noise. This makes the GRNN model attractive for noisy data.

However, this also means that no advantage is obtained when the noise level is reduced, making the GRNN undesirable for pure data.

Other properties of the neural network model and the GRNN model will be presented when the results of the three side experiments are discussed in Chapter 5. The side experiments comprise the remaining half of this study. The same analytic approach will be applied and similar tests of hypotheses will be performed.

CHAPTER 5

DISCUSSION OF RESULTS: THE SIDE EXPERIMENTS

The second half of the study investigates the relationship between estimation error, level of random noise and other sources of uncertainty. Three side experiments were performed to look into these sources. In these experiments, the data generated was based on a linear model. Section 5.1 discusses the results of the experiment investigating the effect of the size of the estimation sample on the performance of the NN and GRNN estiamtes. Section 5.2 shows the results of two analyses involving outliers. Finally, Section 5.3 peeks into the effects of multicollineartity on estimation accuracy.

5.1 Side Experiment 1: Sample Size

In the main experiment, all estimation samples used to obtain the neural network and GRNN models contained 60 points. Table 5.1 shows the MAPEs of the neural network estimates and the true regression estimates when three different sizes of estimation samples are used. The process of estimation is similar to the one used for estimating the LINEAR functional form. The only difference was that in some of the cases, estimation samples consisting of 15 points (LIN15) were used while in other cases the samples were of size 30 (LIN30). As was done in the main experiments, three levels of random noise were incorporated into the samples.The results of these estimations, plus the results of the LINEAR case, are presented in Tables 5.1 and 5.2.

From the results given in Table 5.1, the MAPEs obtained by the neural network estimates were significantly higher than those of the true model when estimating regular data, or when small estimation samples (LIN15) were used. The differences became mostly insignificant when the medium (n=30) and large (n=60) sample sizes were used and when estimates were made for the noise-free (noiseless and extended) data. In

Table 5.1
MAPES for the Different Sample Sizes
Neural Network (NN) vs. True Regression Model (Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
LIN15	High	19.17 (0.17)	18.78	3.1	11.31 (0.49)	10.34	2.8	12.82 (0.58)	11.98	2.2
	Med	14.76 (0.13)	14.15	5.2	7.99 (0.35)	6.25	6.2	9.02 (0.40)	7.17	6.1
	Low	7.93 (0.16)	6.70	8.7	4.50 (0.23)	2.36	11.0	5.30 (0.27)	2.63	11.7
LIN30	High	16.36 (0.09)	16.35	0.1	8.42 (0.29)	8.40	0.1	9.07 (0.32)	9.17	-0.4
	Med	12.48 (0.07)	12.32	2.6	5.12 (0.21)	4.70	2.2	5.44 (0.22)	5.07	1.9
	Low	6.33 (0.04)	5.98	9.0	2.66 (0.10)	1.68	9.3	2.81 (0.10)	1.77	9.4
LINEAR	High	13.64 (0.06)	13.16	9.0	7.42 (0.19)	7.16	1.7	9.85 (0.26)	9.77	0.4
	Med	10.43 (0.04)	10.11	11.2	4.06 (0.14)	3.81	2.2	5.45 (0.17)	5.12	2.4
	Low	5.41 (0.03)	5.01	18.3	1.71 (0.07)	1.16	8.5	2.60 (0.10)	1.46	13.2

addition, all of these insignificant differences occurred under conditions of high and medium noise.

The differences were also smaller for estimates of noiseless and extended data than for estimates of regular data. Even though the NN model had higher MAPEs than the true model, the difference between corresponding models was usually 1% or less. This means that the NN model always gave an approximation that was as accurate as the true model.

For a given noise level, the difference in accuracies decreased when the small sample size was replaced with the medium sample size. However, the difference

increased again when large samples were used. This means that with small samples, both models will have high MAPEs because of the small amount of information contained in the sample. When the sample size increased, the NN benefitted more from the increase in information since the NN model possessed the larger errors in the first place. Then as the amount of information needed by the neural network is saturated, the learning slows down. What this says is that the neural network will continue to be more accurate with a large sample size. However, its performance will not come any closer to that of the true model as when the sample size was 30.

Since the performance of the NN model was consistently close to that of the true model, the NN model should outperform the GRNN model. Table 5.2 shows that this is the case since all z-statistics comparing the difference in errors between the NN model and the GRNN model are highly significant. The MAPEs of the GRNN appear to decrease as the sample size increases, just like the NN model. However the MAPEs of the NN model decreased at a faster rate. This means that the NN model makes more efficient use of the additional information than the GRNN model. This in turn leads to significant differences between the two models. At the highest levels of information (low noise, large sample size) the MAPEs of the GRNN model were ten times those obtained by the NN model.

Just as in the main experiment, the GRNN model was unaffected by changes in the level of random noise. For a given sample size, the three noise levels produced no difference in MAPEs for the GRNN model. The GRNN model also produced the same error estimating regular data as it had estimating noiseless data. Estimates for the extended case produced slightly higher MAPEs.

Table 5.2
MAPES for the Different Sample Sizes
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat
LIN15	High	19.17 (0.17)	22.61	-20.2	11.31 (0.49)	22.62	-23.3	12.82 (0.58)	25.38	-21.7
	Med	14.76 (0.13)	22.61	-57.6	7.99 (0.35)	22.62	-41.7	9.02 (0.40)	25.38	-40.8
	Low	7.93 (0.16)	22.61	-85.5	4.50 (0.23)	22.62	-75.7	5.30 (0.27)	25.38	-72.9
LIN30	High	16.36 (0.09)	19.40	-36.0	8.42 (0.29)	19.40	-37.0	9.07 (0.32)	21.66	-38.8
	Med	12.48 (0.07)	19.40	-88.1	5.12 (0.21)	19.40	-67.6	5.44 (0.22)	21.66	-72.9
	Low	6.33 (0.04)	19.40	-241.8	2.66 (0.10)	19.40	-176.1	2.81 (0.10)	21.65	-183.0
LINEAR	High	13.64 (0.06)	16.64	-51.0	7.42 (0.19)	16.64	-49.2	9.85 (0.26)	22.29	-47.7
	Med	10.43 (0.04)	16.64	-173.2	4.06 (0.14)	16.64	-91.9	5.45 (0.17)	22.29	-96.6
	Low	5.41 (0.03)	16.63	-352.7	1.71 (0.07)	16.63	-204.2	2.60 (0.10)	22.28	-195.4

This side experiment shows that any increase in sample size improves the NN estimate and reduces error. Although the same conclusion applies to the GRNN model, the NN model makes more efficient use of the additional information than the GRNN model since the difference in performance increases.

This experiment showed that when there is a limited number of points to describe a certain distribution, the NN estimate will produce large MAPEs that can still be close to the optimal. With each additional point that is included in the sample, the NN model obtains information that allows it to improve its estimates and reduce its errors. This means that the NN method is able to be take advantage of additional information, better than the GRNN model.

The next experiment looks at the effect of misinformation on the performance of a neural network. The misinformation will be in the form of outliers. Outliers exert a pulling effect on most models. Whether the same effect will be observed for neural network models as well as GRNN models will be discussed in the next section.

5.2 Side Experiment 2: Outliers

Tables 5.3 and 5.4 show the effect of the number and magnitude of outliers on the accuracy of the NN and GRNN estimates. In this experiment, each estimation sample contained one or two outliers. The magnitude of the outliers was either two standard deviations above the expected value (a 2σ-outlier) or four standard deviations above the expected value (a 4σ-outliers).

In Table 5.3, the number of outliers appear to have less effect on the NN model estimates than on the true regression estimates when the outliers are small (2σ). The z-statistic of the difference in MAPEs became smaller as the number of 2σ outliers increased from one to two.

However, the presence of a large outlier (4σ) have a more unsettling effect on the NN model since its MAPEs and the z-statistics show a significant increase over those with small outliers. For the neural network, one large outlier in the estimation sample is less desirable than 2 small outliers, since the large outlier produces a significant increase in error. And when the number of large outliers is increased, the MAPEs increase considerably. In addition, the NN estimates for regular data produces significantly higher errors than estimates made for noiseless and extended data.

Table 5.3
MAPES for Magnitude and Number of Outliers
Neural Network (NN) vs. True Regression Model (Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
LINEAR	High	13.64 (0.06)	13.16	9.0	7.42 (0.19)	7.16	1.7	9.85 (0.26)	9.77	0.4
	Med	10.43 (0.04)	10.11	11.2	4.06 (0.14)	3.81	2.2	5.45 (0.17)	5.12	2.4
	Low	5.41 (0.03)	5.01	18.3	1.71 (0.07)	1.16	8.5	2.60 (0.10)	1.46	13.2
LIN1O2σ	High	13.79 (0.09)	13.20	7.5	7.14 (0.24)	7.17	-0.1	9.44 (0.32)	9.78	-1.6
	Med	10.48 (0.05)	10.16	9.0	4.03 (0.16)	3.85	1.3	5.37 (0.21)	5.14	1.3
	Low	5.39 (0.03)	5.07	11.5	1.68 (0.08)	1.39	3.8	2.48 (0.11)	1.72	7.6
LIN2O2σ	High	13.90 (0.11)	13.25	7.3	7.27 (0.26)	7.29	-0.1	9.49 (0.33)	9.89	-1.7
	Med	10.58 (0.06)	10.21	8.6	4.21 (0.17)	4.06	1.0	5.49 (0.22)	5.33	0.8
	Low	5.42 (0.04)	5.18	6.7	1.81 (0.09)	1.81	-0.1	2.62 (0.12)	2.16	3.8
LIN1O4σ	High	15.56 (0.77)	13.26	3.0	9.50 (0.92)	7.29	2.4	12.32 (1.12)	9.90	2.2
	Med	12.15 (0.89)	10.23	2.2	6.26 (0.98)	4.08	2.2	7.94 (1.03)	5.37	2.4
	Low	6.75 (0.68)	5.24	2.2	3.30 (0.70)	1.95	1.9	4.32 (0.82)	2.37	2.3
LIN2O4σ	High	18.76 (1.52)	13.45	3.5	13.32 (1.68)	7.70	3.3	16.35 (1.77)	10.26	3.4
	Med	14.76 (1.36)	10.44	3.2	9.41 (1.46)	4.76	3.1	11.59 (1.52)	6.02	3.5
	Low	9.64 (1.36)	5.65	3.0	6.43 (1.41)	3.12	2.3	7.54 (1.47)	3.56	2.7

Since the GRNN is unaffected by changes in the level of random noise, it would also be expected to remain immune to the influence of outliers. Table 5.4 shows this phenomenon. The value of the MAPEs for the GRNN in all cases remained basically the

Table 5.4
MAPES for Magnitude and Number of Outliers
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat
LINEAR	High	13.64 (0.06)	16.64	-51.0	7.42 (0.19)	16.64	-49.2	9.85 (0.26)	22.29	-47.7
	Med	10.43 (0.04)	16.64	-173.2	4.06 (0.14)	16.64	-91.9	5.45 (0.17)	22.29	-96.6
	Low	5.41 (0.03)	16.63	-352.7	1.71 (0.07)	16.63	-204.2	2.60 (0.10)	22.28	-195.4
LIN1O2σ	High	13.79 (0.09)	16.66	-31.9	7.14 (0.24)	16.66	-40.1	9.44 (0.32)	22.30	-40.4
	Med	10.48 (0.05)	16.66	-127.1	4.03 (0.16)	16.66	-81.5	5.37 (0.21)	22.30	-81.1
	Low	5.39 (0.03)	16.68	-292.8	1.68 (0.08)	16.67	-189.6	2.48 (0.11)	22.32	-185.6
LIN2O2σ	High	13.90 (0.11)	16.70	-27.3	7.27 (0.26)	16.70	-37.7	9.49 (0.33)	22.33	-38.6
	Med	10.58 (0.06)	16.70	-106.5	4.21 (0.17)	16.70	-76.0	5.49 (0.22)	22.33	-75.8
	Low	5.42 (0.04)	16.70	-90.6	1.81 (0.09)	16.70	-84.0	2.62 (0.12)	22.33	-83.5
LIN1O4σ	High	15.56 (0.77)	16.70	-1.5	9.50 (0.92)	16.70	-7.8	12.32 (1.12)	22.33	-8.9
	Med	12.15 (0.89)	16.70	-5.1	6.26 (0.98)	16.70	-10.6	7.94 (1.03)	22.33	-14.0
	Low	6.75 (0.68)	16.70	-14.6	3.30 (0.70)	16.70	-19.1	4.32 (0.82)	22.33	-21.8
LIN2O4σ	High	18.76 (1.52)	16.85	1.3	13.32 (1.68)	16.85	-2.1	16.35 (1.77)	22.46	-3.4
	Med	14.76 (1.36)	16.85	-1.5	9.41 (1.46)	16.84	-5.1	11.59 (1.52)	22.46	-7.2
	Low	9.64 (1.36)	16.84	-5.3	6.43 (1.41)	16.84	-7.4	7.54 (1.47)	22.45	-10.1

same with the increase in the number of outliers. When the magnitude of the outliers increased, the effect was minimal. So while the NN performance deteriorated, the GRNN performance remained largely unaffected. The increased presence of outliers resulted in smaller and less significant differences between the MAPEs of the NN annd GRNN models. With one or two large (4σ) outliers, the GRNN model even made more accurate forecasts for the regular data than the NN model. Actually, identical MAPEs were registered when estimating regular and noiseless data. Tables 5.5 and 5.6 show the effect on the NN model and the GRNN model of combining the expanded information provided by an increase in sample size and the misdirection introduced by outliers.

When the sample size is small, the presence of a single outlier is stronger leading to larger MAPEs for the NN estimate as well as the true regression estimate. This can be observed if the MAPEs from the samples with no outliers (Table 5.1) are compared with those containing outliers (Table 5.5). Since both the NN model and true regression model are working with less information and more uncertainty, the MAPEs are larger but the difference in MAPEs was smaller. In most of the cases with small and medium sample size, the difference in MAPEs was not significant.

The presence of the outlier became more pronounced when estimating noise-free (noiseless and extended) data. When the sample, with a single outlier, is small, the NN model produces virtually the same performance as the true model. However, when estimating regular data with a large sample size, the true model is able to produce significantly lower MAPEs than the NN model. Thus, the true model is able to rise above the noise introduced by the outlier in estimating regular data. For the noise-free test data, the outliers again produce uncertainty in the estimates of both the NN model and the true model. Thus, the noise-free data show insignificant differences between the performance of the NN model and the performance of thetrue model.

Table 5.5
MAPES for Outliers and Sample Size
Neural Network (NN) vs. True Regression Model (Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
LIN15N2σ	High	20.08 (0.51)	19.09	2.1	12.54 (0.75)	10.86	2.4	14.15 (0.86)	12.55	2.1
	Med	15.36 (0.38)	14.53	2.3	8.58 (0.57)	7.12	2.7	9.60 (0.65)	8.08	2.6
	Low	7.98 (0.20)	7.48	2.4	4.59 (0.28)	4.25	1.1	5.33 (0.34)	4.67	1.8
LIN30N2σ	High	16.63 (0.32)	16.40	0.7	9.24 (0.50)	8.75	0.9	9.98 (0.53)	9.54	0.8
	Med	12.67 (0.14)	12.43	1.8	5.47 (0.26)	5.20	1.0	5.81 (0.27)	5.58	0.8
	Low	6.40 (0.06)	6.24	2.4	2.80 (0.13)	2.58	1.5	2.98 (0.14)	2.70	1.8
LIN1O2σ	High	13.79 (0.09)	13.20	7.5	7.14 (0.24)	7.17	-0.1	9.44 (0.32)	9.78	-1.6
	Med	10.48 (0.05)	10.16	9.0	4.03 (0.16)	3.85	1.3	5.37 (0.21)	5.14	1.3
	Low	5.39 (0.03)	5.07	11.5	1.68 (0.08)	1.39	3.8	2.48 (0.11)	1.72	7.6

Just like in Table 5.1, the MAPEs gradually decreased as the sample size increased. The effect of the outlier is drowned out by the greater number of correct points in the sample making the presence of the single outlier unnoticeable. This is why at the largest sample size, the MAPEs of the NN model and the true model when one outlier is present (Table 5.5) were equivalent to the MAPEs obtained in Table 5.1, when no outlier existed.

The MAPEs obtained for the GRNN model when a single outlier is present (Table 5.6) are almost identical to those when no outlier is present (Table 5.2). This means that the single outlier produced no effect on the GRNN model. Since the NN model was more adversely affected by the single outlier, the difference in MAPEs

between the NN model and the GRNN model are now smaller. However, these differences were still highly significant.

Again, as in Table 5.2, the MAPEs for the GRNN model showed no effect from the changing noise levels. Thus, the GRNN model is virtually unaffected by both noise and outliers.

Just as outliers add noise to a sample, multicollinear variables introduce another form of distraction when searching for the distribution underlying an estimation sample. The third side experiment investigates the effect of multicollinearity on the estimates produced by neural network and GRNN models.

Table 5.6
MAPES for Outliers and Sample Size
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat
LIN15N2σ	High	20.08 (0.51)	22.70	-5.2	12.54 (0.75)	22.70	-13.7	14.15 (0.86)	25.51	-13.4
	Med	15.36 (0.38)	22.70	-19.8	8.58 (0.57)	22.70	-24.9	9.60 (0.65)	25.51	-24.5
	Low	7.98 (0.20)	22.71	-72.4	4.59 (0.28)	22.71	-63.2	5.33 (0.34)	25.50	-59.1
LIN30N2σ	High	16.63 (0.32)	19.43	-8.7	9.24 (0.50)	19.42	-20.1	9.98 (0.53)	21.67	-21.6
	Med	12.67 (0.14)	19.43	-45.9	5.47 (0.26)	19.42	-52.5	5.81 (0.27)	21.67	-58.3
	Low	6.40 (0.06)	19.42	-183.2	2.80 (0.13)	19.42	-127.4	2.98 (0.14)	21.66	-133.6
LIN1O2σ	High	13.79 (0.09)	16.66	-31.9	7.14 (0.24)	16.66	-40.1	9.44 (0.32)	22.30	-40.4
	Med	10.48 (0.05)	16.66	-127.1	4.03 (0.16)	16.66	-81.5	5.37 (0.21)	22.30	-81.1
	Low	5.39 (0.03)	16.68	-292.8	1.68 (0.08)	16.67	-189.6	2.48 (0.11)	22.32	-185.6

Table 5.7
MAPES for Multicollinearity
Neural Network (NN) vs. True Regression Model(Reg)

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat	NN (%)	Reg (%)	Z Stat
X1+X2	High	15.01 (0.07)	16.43	-14.9	8.00 (0.22)	5.48	8.5	10.08 (0.28)	6.96	8.2
	Med	10.76 (0.07)	10.93	-1.9	4.82 (0.21)	3.03	7.0	6.11 (0.27)	3.79	7.1
	Low	5.05 (0.03)	4.84	5.2	1.85 (0.06)	1.01	8.7	2.66 (0.08)	1.21	11.3
MX2LOW	High	20.60 (0.40)	19.04	3.8	11.95 (0.57)	11.56	0.7	15.03 (0.61)	15.21	-0.3
	Med	14.62 (0.07)	14.47	2.0	6.89 (0.21)	8.73	-8.3	8.78 (0.27)	11.40	-8.9
	Low	7.15 (0.07)	9.34	-27.2	2.85 (0.14)	8.00	-35.2	3.95 (0.17)	10.34	-35.0
MX2MED	High	22.18 (0.92)	19.23	3.2	13.37 (1.17)	10.63	2.4	16.67 (1.37)	13.95	2.0
	Med	14.61 (0.08)	13.99	7.7	6.60 (0.19)	7.05	-2.3	8.48 (0.27)	9.16	-2.5
	Low	7.27 (0.07)	7.75	-6.3	2.98 (0.14)	5.69	-19.1	4.18 (0.17)	7.35	-18.0
MX2HIGH	High	21.62 (0.65)	20.06	2.4	12.07 (0.87)	10.12	2.2	14.90 (0.86)	13.37	1.8
	Med	14.62 (0.07)	14.30	3.6	6.10 (0.21)	5.68	1.4	7.85 (0.27)	7.42	1.1
	Low	7.21 (0.05)	6.65	10.5	2.99 (0.08)	3.00	-0.0	4.30 (0.11)	3.85	3.8

5.3 Side Experiment 3: Multicollinearity

Tables 5.7 and 5.8 show MAPEs obtained for a bivariate model when the two independent variables are collinear. The results of modeling X1+X2 are also included to allow for comparison with non-collinear models.

According to Table 5.7, the NN models exhibit no significant changes in the MAPEs of the estimates given increasing levels of multicollinearity. This applies to the

regular data, and noiseless data as well as to the extended data. The differences in MAPEs can be attributed solely to the changes in the level of random noise. On the other hand, the true regression model exhibits decreasing MAPEs for increasing levels of multicollinearity. The result is that at low levels of multicollinearity, the NN models provide significantly more accurate estimates than the true model. This occurs specially under conditions of low noise. However, as the level of multicollinearity increases, the MAPEs of the true model improves, and the advantage swings in the direction of the true model.

From Table 5.8, the effect of multicollinearity on the performance of the GRNN model is also minimal. The MAPEs of the GRNN estimate only exhibit changes when tested using the noiseless data. The significant change occurs when the level of multicollinearity increase from low to high. With this change, the MAPEs of the GRNN models decreased from about 19% to between 5% to 10%. Since the MAPEs for the NN model and the GRNN models were largely unaffected by the level of multicollinearity, the differences remained virtually unchanged.

Table 5.8
MAPES for Multicollinearity
Neural Network (NN) vs. GRNN

Model	Noise Level	Regular Test Data			Noiseless Test Data			Extended Test Data		
		NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat	NN (%)	GRNN (%)	Z Stat
X1+X2	High	15.01 (0.07)	20.79	-58.5	8.00 (0.22)	15.37	-31.5	10.08 (0.28)	18.72	-28.8
	Med	10.76 (0.07)	18.30	-67.9	4.82 (0.21)	15.35	-45.1	6.11 (0.27)	18.70	-42.5
	Low	5.05 (0.03)	16.26	-29.6	1.85 (0.06)	15.35	-29.4	2.66 (0.08)	18.70	-29.5
MX2LOW	High	20.60 (0.40)	21.21	-1.5	11.95 (0.57)	18.99	-12.3	15.03 (0.61)	24.24	-15.0
	Med	14.62 (0.07)	20.06	-73.5	6.89 (0.21)	18.98	-59.5	8.78 (0.27)	24.23	-56.7
	Low	7.15 (0.07)	19.20	-166.1	2.85 (0.14)	18.96	-119.1	3.95 (0.17)	24.21	-118.6
MX2MED	High	22.18 (0.92)	21.39	0.9	13.37 (1.17)	10.63	-4.8	16.67 (1.37)	24.14	-5.4
	Med	14.61 (0.08)	20.15	-68.2	6.60 (0.19)	7.05	-64.7	8.48 (0.27)	24.14	-57.5
	Low	7.27 (0.07)	19.21	-158.1	2.98 (0.14)	5.69	-116.1	4.18 (0.17)	24.12	-116.9
MX2HIGH	High	21.62 (0.65)	22.22	-0.9	12.07 (0.87)	10.12	-8.4	14.90 (0.86)	24.79	-11.5
	Med	14.62 (0.07)	20.73	-64.3	6.10 (0.21)	5.68	-52.6	7.85 (0.27)	24.77	-52.6
	Low	7.21 (0.05)	19.66	-233.6	2.99 (0.08)	3.00	-188.3	4.30 (0.11)	24.77	-187.3

CHAPTER 6

ANALYSIS OF TRAINING STATISTICS

The preceding two chapters discussed the results of the study that shed light on the relationship between the characteristics of data and the performance and accuracy of the neural network estimate. The results show that in many of the cases considered by the study, the neural network provided estimates that were almost as accurate as the true regression model.

Since the derivation of the neural network estimates involves considerable time and computation during training, the effect of the characteristics of data on the training process is also an important concern in modeling. Besides the performance of the network during testing, the ease or difficulty of training a network and the structure of the network estimate are linked to the characteristics of the data being modeled. This chapter will present some of the statistics gathered during training and explore how these statistics relate to the characteristics of data. Section 6.1 will describe the statistics that will be used and how these statistics can characterize data. Section 6.2 then provides an analysis of the training statistics for the simple functional forms. Section 6.3 will do the same for the more complex functional forms and Section 6.4 will cover the bivariate functional forms. Sections 6.5, 6.6 and 6.7 then discuss the effect of the size of the estimation sample, the presence of outliers, and the level of multicollinearity, respectively, on the training process.

6.1 Epoch Statistics and Network Statistics

The process of obtaining a neural network estimate starts with an estimation sample with specified characteristics in the data. Using this estimation sample, a 1-4-2-1 network is trained, pruned and tested. This process is repeated for 100 estimation samples. From

the 100 repetitions, statistics can be gathered to provide a general picture of how training progresses for a given set of characteristics of data.

The statistics that have been obtained can be divided into two groups, epoch statistics and network statistics. The epoch statistics include the average number of epochs used for the initial simplex method, the average number used for backpropagation, and the average number used for the final simplex application. This group also includes the MAPE of the trained network on the estimation sample.

The number of epochs used by the initial simplex will indicate how much preprocessing can be performed on the data prior to backpropagation. If a larger number of epochs were performed by the initial simplex compared to backpropagation, then a substantial portion of the backpropagation load was assumed by the initial simplex saving considerable training time. The number of epochs used by backpropagation will indicate how difficult it was to model the data and determine the minimal network at the same time. The number of epochs used by the final simplex will indicate the amount of additional fine tuning needed to bring the backpropagation estimate closer to the optimal model. Finally, the MAPE of the network estimate on the estimation sample will reflect how successful the training process was in modeling the estimation data.

The structure of the network obtained by the training process indicates the complexity of the model estimate. From the 100 model estimates, network statistics are obtained that include the average number of branches, the average number of processing nodes, and the average number of nonzero parameters in the 100 estimates. Note that the number of processing nodes does not include the input nodes. The average number of branches and processing nodes will then give an idea of how an "average trained network" will look for a given set of characteristics.

Table 6.1
Possible Structures for a Univariate Neural Network

Network Structure	Number of Branches	Number of Processing Nodes
1-4-2-1	21	7
1-3-2-1	17	6
1-4-1-1	15	6
1-2-2-1	13	5
1-3-1-1	12	5
1-4-1	9	5
1-2-1-1	9	4
1-3-1	7	4
1-1-1-1	6	3
1-2-1	5	3
1-1-1	3	2

The structure of the "average network" can be obtained by comparing the average number of branches and processing nodes of the trained network with those of the structures given in Tables 6.1 and 6.2. Table 6.1 provides information on networks with a single input node while Table 6.2 gives the numbers for the bivariate case. Since all training starts with a 1-4-2-1 network (2-4-2-1 in the bivariate case), the size and complexity of the final network will indicate how successful the training process was in pruning the network while maximizing accuracy. Averaging the final networks for each of the 100 estimation samples gives a general picture of what network structure should be required for a given set of characteristics of data. A researcher can then use the "average network" as the target model to be obtained through training.

Although the number of branches and processing nodes determine the structure of the network, these two values do not necessarily determine the number of parameters in the model. This is because the weights of some of the branches may be zero. Thus the number of parameters is actually the total number of nonzero weights and biases in the network. The number of parameters will indicate the complexity of the model obtained from the estimation data.

6.2 Simple Functional Forms

The number of epochs used in obtaining the neural network estimates for the five simple functional forms are given in Table 6.3. From the table, the SQRT form shows the least

Table 6.2
Possible Structures for a Bivariate Neural Network

Network Structure	Number of Branches	Number of Processing Nodes
2-4-2-1	28	7
2-3-2-1	23	6
2-4-1-1	21	6
2-2-2-1	18	5
2-3-1-1	17	5
2-4-1	14	5
2-2-1-1	13	4
2-3-1	11	4
2-1-1-1	9	3
2-2-1	8	3
2-1-1	5	2

Table 6.3
Epoch Statistics for the Simple Functional Forms

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
SQUARED	High	618.61	1612.20	145.67	80.72
	Med	743.36	1664.40	224.05	51.74
	Low	1002.97	1660.80	400.67	12.30
LINEAR	High	593.62	1377.60	545.55	13.60
	Med	578.24	1424.70	488.30	10.21
	Low	942.66	1753.50	355.91	5.18
SQRT	High	596.34	1299.90	106.99	6.16
	Med	552.14	1334.40	85.00	4.69
	Low	683.24	1353.30	102.68	2.42
LOG	High	627.89	1442.10	109.02	8.88
	Med	745.89	1539.30	138.14	6.71
	Low	792.80	1495.80	153.91	3.53
RECIP	High	1239.71	2823.30	1011.38	96.03
	Med	793.93	2244.90	312.07	26.89
	Low	1126.77	1630.80	123.98	5.73

number of epochs used in the three stages of training. It also provides the lowest MAPEs of the network model when fitting the estimation sample. This indicates the relative ease with which a neural network can model the SQRT data. After the SQRT data, the LOG data provided the shortest training time and the lowest MAPEs. The LOG data is then followed by the LINEAR form. The neural network's success in training with these three forms coincides with the high level of performance that neural network produced for these data in Chapter 4.

At the opposite end, Table 6.3 shows that the neural network had the most difficulty and the least success training for the RECIP and SQUARED data. The RECIP data required the largest number of epochs and therefore the longest training time. In addition, while the number of epochs required by backpropagation were increasing with

decreasing noise for the other forms, the number for the RECIP form was decreasing. Even with the long training, the neural network model failed to provide a satisfactory fit to the estimation sample. Only under conditions of low noise did the neural network obtain small MAPEs for the RECIP data.

The SQUARED model showed the least success with respect to fitting the estimation sample. The amount of time used for training was generally smaller than those required by the RECIP data. This means that training did not take long to converge to a set of weights. Unfortunately, this set of weights produced a terrible fit of the estimation sample. This condition suggests that the failure in training was caused by entrapment in a local minima rather than lack of convergence. The researcher is then advised to use large initial values for the learning rate as one way of overcoming local minima when training with SQUARED data.

The average number of nodes and branches of the trained networks are given in Table 6.4. These averages were compared with the number of nodes and branches in the different network configurations listed in Table 6.1. The configuration corresponding to the pair of nodes and branches that appears closest to the average values is then used as the "average" network.

Among the simple functional forms, the SQRT form and the LOG form produced the simplest configuration at the end of training. Consequently, these two also had the smallest number of parameters in the model. This means that backpropagation pruned off more parameters in these models than it did for the other types of data. Surprisingly, the LINEAR form obtained models that were more complicated models than the SQUARED or RECIP forms. This can be interpreted as a sign that the error surface of the LINEAR form consists of slowly sloping curves in large dimensions. This would

account for the slow convergence exhibited by backpropagation to obtain a model with a large number of parameters.

6.3 More Complex Functional Forms

For the five complex functional forms, the epoch statistics are given in Table 6.5 and the network statistics are presented in Table 6.6. Under conditions of low and medium noise, the NN model for the LOGISTIC form required the smallest number of epochs for backpropagation and provided the smallest MAPEs. The LOGISTIC estimate is followed by the estimates for the EXRIS data and MICHAELIS data. The latter two forms also provided the shortest training times under conditions of low noise. On the other hand, the XLNX form required the longest training time while the estimate for the

Table 6.4
Average Number of Elements in a Trained Network
for the Simple Functional Forms

Model	Noise Level	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
SQUARED	High	7.67	3.36	9.49	1-3-1
	Med	10.76	4.21	12.65	1-3-1-1
	Low	11.55	4.41	14.50	1-3-1-1
LINEAR	High	19.38	6.55	25.43	1-4-2-1
	Med	18.54	6.32	22.99	1-4-2-1
	Low	13.34	4.95	12.08	1-4-1-1
SQRT	High	3.48	2.14	5.31	1-2-1
	Med	3.03	2.00	4.99	1-1-1
	Low	3.09	2.03	5.12	1-1-1
LOG	High	3.58	2.16	5.44	1-2-1
	Med	4.02	2.3	5.74	1-2-1
	Low	4.36	2.39	5.84	1-2-1
RECIP	High	14.84	5.32	18.94	1-4-1-1
	Med	7.53	3.33	8.70	1-2-1-1
	Low	3.74	2.22	5.52	1-2-1

Table 6.5
Epoch Statistics for the More Complex Functional Forms

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
XSQX	High	241.71	525.90	12.43	1267.74
	Med	1082.66	1565.40	372.80	13.18
	Low	800.32	1571.10	253.24	35.95
XLNX	High	932.49	2121.69	662.66	23.78
	Med	786.18	1905.60	465.18	18.08
	Low	1085.22	1661.40	368.97	8.52
MICHAELIS	High	692.62	1134.60	91.20	6.70
	Med	744.83	1142.40	103.42	5.07
	Low	782.09	1137.30	98.94	2.64
EXRIS	High	585.97	1457.10	457.69	6.24
	Med	643.07	1518.90	466.93	4.68
	Low	646.86	1659.90	265.08	2.40
LOGISTIC	High	708.92	1524.00	331.89	13.00
	Med	830.82	736.50	88.45	0.93
	Low	1075.63	846.30	90.29	0.47

XSQX form produced the worst fit of the estimation sample. Again, these results were consistent with the performance of the estimates for these functional forms discussed in Chapter 4.

For the XSQX data, the estimates had the highest errors under conditions of high noise. The training for the XSQX data was relatively short requiring a very small number of epochs for backpropagation (525.90). However, the MAPEs of the resulting network had an average MAPE of 1267%. Again, this means that problems with local minima exist for the XSQX data, similar to those faced with the SQUARED data. The high amount of noise prevents any type of learning to take place for the neural network. When the level of noise was reduced, the neural network was able to use more epochs in training to obtain better estimates. The resulting fits of 13.18% and 35.95% look more

Table 6.6
Average Number of Elements in a Trained Network
for the More Complex Functional Forms

Model	Noise Level	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
XSQX	High	19.71	6.67	22.70	1-4-2-1
	Med	11.42	4.46	12.41	1-3-1-1
	Low	9.63	3.89	10.97	1-2-2-1
XLNX	High	17.38	6.04	19.66	1-3-2-1
	Med	16.38	5.73	20.07	1-3-2-1
	Low	12.44	4.67	16.32	1-2-2-1
MICHAELIS	High	3.09	2.03	5.11	1-1-1
	Med	3.24	2.08	5.30	1-1-1
	Low	3.06	2.02	5.08	1-1-1
EXRIS	High	15.39	5.48	19.20	1-4-1-1
	Med	15.38	5.50	19.15	1-4-1-1
	Low	9.13	3.82	10.71	1-2-2-1
LOGISTIC	High	11.03	4.26	13.62	1-3-1-1
	Med	3.00	2.00	5.00	1-1-1
	Low	3.00	2.00	5.00	1-1-1

acceptable, although these MAPEs still remain significantly higher that the rest of the functional forms.

With respect to model complexity, the estimates in Table 6.6 obtained for the MICHAELIS data produced the simplest model, followed by the network for the LOGISTIC data. For these models, the short training time, the closeness of the fit and the simplicity of the model estimate point to an error surface with deep slopes and few local minima. In contrast, the long training period, high MAPEs, and complicated network produced by the estimate for the XLNX model suggest an error surface with gentle slopes conducive to slow convergence.

The EXRIS data is similar to the MICHAELIS data in that a small number of epochs were required to obtain model estimates that provided a good fit to the estimation data. However, the EXRIS data required models that were as complicated as those used by the XSQX data. This means that the pruning process found most of the parameters important in modeling the data. This suggests that the error surface for the EXRIS data exhibits the same kind of fast convergence possessed by the error surface of the MICHAELIS data but in more dimensions. Thus, the error surface of the EXRIS data involved 20 dimensions while the error surface of the MICHAELIS data only involved 6.

6.4 Bivariate Functional Forms

Among the bivariate functional forms in Table 6.7, the estimates for the X1*X2 model show significantly higher MAPEs than the rest of the group. This occurred even though all four functional forms used relatively the same number of epochs for backpropagation. This discrepancy may be traced to the amount of fine tuning that the downhill simplex achieved after backpropagation. For the X1*X2 data, the number of epochs used by the second application of the simplex was only about 1/2 to 1/3 of the number required for the other bivariate forms. This means that less fine tuning was performed by the second simplex on the X1*X2 data than those performed on the other forms.

For Table 6.8, the average network was obtained by comparing the average number of nodes and branches with the configurations given in Table 6.2. From Table 6.8, the network statistics show that the estimates for the X1*X2 data require the smallest number of parameters and the simplest configuration. The other three forms showed results that were similar to each other and significantly higher than the one obtained for the X1*X2 data.

Table 6.7
Epoch Statistics for the Bivariate Functional Forms

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
X1+X2	High	1327.53	1588.49	848.49	11.76
	Med	1521.98	1448.49	1031.82	8.81
	Low	2307.93	1388.00	681.33	4.40
X1LOGX2	High	1452.71	1485.31	915.96	11.82
	Med	1689.24	1491.30	1033.41	8.86
	Low	2215.98	1689.90	893.79	4.56
X1*X2	High	951.91	1486.50	218.90	56.99
	Med	1510.20	1791.30	352.65	28.68
	Low	2169.33	1664.08	373.94	9.67
X1/X2	High	1404.43	1486.20	944.88	11.15
	Med	1376.41	1485.60	904.59	8.36
	Low	2197.02	1402.27	794.73	4.24

Since the X1*X2 estimate was reduced to the simplest configuration, it offered the least opportunity for fine tuning since there were fewer parameters to adjust. This in turn resulted in a less accurate model. In essence, the training sacrificed accuracy in order to reduce network complexity. If this result is not acceptable to the user, then training may be repeated using a smaller coefficient of decay. The smaller coefficient will reduce the weight of the complexity term in the cost function, allowing more complicated models to be formed.

6.5 Sample Size

Tables 6.9 and 6.10 summarize the training statistics for the side experiment investigating the effect of different sizes of estimation samples. The results show that the size of the estimation sample did not affect the number of epochs used. This means that approximately the same number of presentations of the estimation sample was used

Table 6.8
Average Number of Elements in a Trained Network
for the Bivariate Functional Forms

Model	Noise Level	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
X1+X2	High	24.13	6.19	28.71	2-4-2-1
	Med	25.34	6.45	29.03	2-4-2-1
	Low	22.84	5.93	24.20	2-3-2-1
X1LOGX2	High	23.14	5.99	28.41	2-3-2-1
	Med	23.30	6.02	28.28	2-3-2-1
	Low	21.44	5.65	22.06	2-4-1-1
X1*X2	High	14.25	4.13	15.70	2-4-1
	Med	14.63	4.19	15.19	2-4-1
	Low	12.62	3.74	13.85	2-2-1-1
X1/X2	High	23.29	6.02	27.66	2-3-2-1
	Med	23.84	6.12	27.87	2-3-2-1
	Low	21.88	5.75	23.09	2-3-2-1

during training. However, the MAPEs obtained for the estimation sample showed improvements with increasing sample size. It should be clarified that, since the same number of epochs was used, the length of training (in real time) would be longer for a larger estimation sample. This is simply because more points need to be presented during training.

According to Table 6.10, the complexity of the network model was moderately affected by changes in sample size. Most of these changes involved the addition or deletion of one parameter in the model, depending on the level of random noise. At high noise, an increase in sample size brought about a slight increase in the number of parameters in the model. At low noise, the same increase in sample size brought about a slight decrease in the complexity of the model. This can be viewed in terms of how the neural network sees the points in the sample.

Table 6.9
Epoch Statistics for the Different Sample Sizes

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
LIN15	High	867.14	1347.90	629.60	12.15
	Med	879.46	1441.50	642.18	9.12
	Low	939.70	1596.30	399.53	4.79
LIN30	High	713.25	1436.40	635.82	13.28
	Med	672.18	1392.90	579.70	9.92
	Low	877.69	1645.50	353.19	5.07
LINEAR	High	593.62	1377.60	545.55	13.60
	Med	578.24	1424.70	488.30	10.21
	Low	942.66	1753.50	355.91	5.18

Table 6.10
Average Number of Elements in a Trained Network
for the Different Sample Sizes

Model	Noise	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
LIN15	High	17.69	6.09	23.11	1-3-2-1
	Med	18.72	6.39	23.22	1-4-2-1
	Low	12.87	4.77	14.50	1-2-2-1
LIN30	High	18.70	6.37	24.21	1-4-2-1
	Med	18.47	6.32	22.70	1-4-2-1
	Low	12.08	4.58	12.75	1-3-1-1
LINEAR	High	19.38	6.55	25.43	1-4-2-1
	Med	18.54	6.32	22.99	1-4-2-1
	Low	13.34	4.95	12.08	1-2-2-1

At high noise, the neural network saw each point as introducing more noise than information. Thus, an increase in sample size was seen as an increase in noise so that additional parameters were used to understand the noise. At low noise, each point was seen as representing more information than noise so that fewer parameters were used to

model larger samples. At medium noise, the network may have regarded each point as having equivalent noise and information so that no changes were made in the number of parameters in the model. The reader is reminded that although the size of the sample only moderately affects the complexity of the model, the size of the estimation sample significantly affects the accuracy of the model as shown in Section 5.1.

6.6 Outliers

In contrast to the sample size, the number and magnitude of outliers had a direct effect on the length of training and on the complexity of the model. Table 6.11 shows that the length of training increased as more outliers or larger outliers were introduced into the estimation sample. The network required more epochs to be able to learn the behavior of

Table 6.11
Epoch Statistics for the Different Number of Outliers

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
LINEAR	High	593.62	1377.60	545.55	13.60
	Med	578.24	1424.70	488.30	10.21
	Low	942.66	1753.50	355.91	5.18
LIN1O2σ	High	589.35	1563.30	477.37	14.45
	Med	624.19	1443.90	502.12	10.97
	Low	815.68	1681.50	434.69	6.03
LIN2O2σ	High	643.39	1477.80	530.43	15.09
	Med	671.54	1409.70	522.24	11.60
	Low	978.43	1552.42	544.25	6.77
LIN1O4σ	High	780.77	1798.80	544.42	19.29
	Med	786.49	1649.40	651.01	16.35
	Low	772.61	1731.00	600.61	11.42
LIN2O4σ	High	954.91	2007.60	757.58	24.65
	Med	962.10	1822.50	780.07	21.33
	Low	975.88	1746.60	728.62	16.87

Table 6.12
Average Number of Elements in a Trained Network
for the Different Number of Outliers

Model	Noise Level	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
LINEAR	High	19.38	6.55	25.43	1-4-2-1
	Med	18.54	6.32	22.99	1-4-2-1
	Low	13.34	4.95	12.08	1-2-2-1
LIN1O2σ	High	16.49	5.77	21.68	1-3-2-1
	Med	17.64	6.10	21.69	1-3-2-1
	Low	16.09	5.67	16.53	1-3-2-1
LIN2O2σ	High	17.92	6.16	23.63	1-4-2-1
	Med	18.89	6.42	24.63	1-4-2-1
	Low	18.03	6.22	21.33	1-4-2-1
LIN1O4σ	High	15.43	5.49	20.13	1-4-1-1
	Med	17.99	6.20	22.22	1-4-2-1
	Low	17.70	6.10	21.25	1-3-2-1
LIN2O4σ	High	16.95	5.92	21.15	1-3-2-1
	Med	17.42	6.04	22.16	1-3-2-1
	Low	17.92	6.17	22.75	1-3-2-1

the outlier. This in turn resulted in a model that tried to fit the outlier rather than fitting other more legitimate points. The resulting model thus exhibited higher errors in fitting the estimation sample.

Since outliers introduced more noise into the estimation sample above the random noise already present, the resulting model used more parameters to account for the extra noise. The network structures given in Table 6.12 support this reasoning. Under conditions of low noise, the introduction of more outliers or bigger outliers resulted in the number of parameters increasing from 12 to 16 to 21. Since the level of random noise was low, the increase in parameters can be attributed to the noise introduced by the outlier. When the level of random noise was high, the effect of outliers was not as

pronounced since the extra parameters may have been used to model the random noise or the noise introduced by the outliers.

A similar conclusion can be drawn from Tables 6.13 and 6.14. In Table 6.13, the number of epochs used in backpropagation appears to increase with increasing sample size while the number of epochs used in the final simplex appears to decrease with increasing sample size. The total number of epochs used to obtain a neural network model thus appeared to remain the same for the different sample sizes. However, the MAPEs of the estimates appeared to decrease slightly with increasing sample size.

The above observations are consistent with our earlier observation that the sample size does not have a significant effect on the number of epochs used in training. Apparently, the size of the outlier used in the sample was not sufficient to produce any effect on the number of epochs. Concerning the MAPEs on the estimation sample, the small sample size exhibited larger errors since the effect of an outlier was stronger on smaller samples, provided the level of random noise was low. When the level of noise

Table 6.13
Epoch Statistics for the Outliers and Sample Sizes

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
LIN15N2σ	High	923.62	1462.80	797.75	14.47
	Med	915.89	1463.10	709.49	11.41
	Low	1169.48	1516.80	580.09	7.37
LIN30N2σ	High	782.29	1430.40	595.21	14.85
	Med	726.40	1419.90	556.42	11.53
	Low	1020.49	1691.70	509.58	6.74
LIN1O2σ	High	589.35	1563.30	477.37	14.45
	Med	624.19	1443.90	502.12	10.97
	Low	815.68	1681.50	434.69	6.03

Table 6.14
Average Number of Elements in a Trained Network
for the Outliers and Sample Sizes

Model	Noise Level	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
LIN15N2σ	High	19.10	6.49	24.84	1-4-2-1
	Med	19.38	6.56	24.53	1-4-2-1
	Low	17.37	6.04	20.59	1-3-2-1
LIN30N2σ	High	18.21	6.24	23.38	1-4-2-1
	Med	19.47	6.60	24.67	1-4-2-1
	Low	17.13	5.87	19.66	1-3-2-1
LIN1O2σ	High	16.49	5.77	21.68	1-3-2-1
	Med	17.64	6.10	21.69	1-3-2-1
	Low	16.09	5.67	16.53	1-3-2-1

was high, the outlier was again rendered ineffective.

The effect of an outlier on the complexity of a model depends on whether the noise introduced by an outlier can rise above the size of the sample and the level of random noise. In Table 6.14, the number of parameters present in the model showed little change with increasing sample size when the level of noise was high. When the level of random noise was low, the effect of the outlier was again manifested in the larger number of parameters produced by the smaller estimation samples. These additional parameters will be used to model the noise introduced by the outlier as the outlier exerts a more pronounced effect on smaller samples.

6.7 Multicollinearity

Tables 6.15 and 6.16 show the training statistics for those cases that involved multicollinearity. Table 6.15 shows that the level of multicollinearity did not have any effect on the number of epochs used in training nor on the MAPE of the model estimate.

Table 6.15
Epoch Statistics for the Multicollinearity Experiment

Model	Noise Level	Training Epochs Used			Training MAPE (%)
		Front Simplex	Back-Propagation	Back Simplex	
MX2LOW	High	2346.19	1873.20	1098.65	17.06
	Med	2524.70	1584.60	1071.84	12.64
	Low	3153.75	1570.50	1054.46	6.34
MX2MED	High	2277.00	1868.40	920.77	18.29
	Med	2592.26	1576.80	925.18	12.60
	Low	3237.76	1583.10	983.12	6.38
MX2HIGH	High	1866.16	1868.40	983.25	17.90
	Med	2210.64	1510.00	952.14	12.59
	Low	3209.77	1460.10	768.17	6.37

The changes in the statistics were due entirely to the changes in the level of random noise. Table 6.16 presents the same picture. The level of multicollinearity did not affect the number of parameters in the models nor the structure of the average network. At given levels of noise, these statistics exhibit no change at different levels of multicollinearity.

Table 6.16
Average Number of Elements in a Trained Network
for the Multicollinearity Experiment

Model	Noise Level	Average Number in Network			"Average" Network
		Branches	Nodes	Parameters	
MX2LOW	High	21.18	5.53	20.27	2-4-1-1
	Med	23.81	6.11	22.06	2-3-2-1
	Low	19.95	5.32	21.31	2-4-1-1
MX2MED	High	19.95	5.28	17.20	2-4-1-1
	Med	25.23	6.43	25.48	2-4-2-1
	Low	19.64	5.25	21.83	2-4-1-1
MX2HIGH	High	20.93	5.48	21.09	2-4-1-1
	Med	24.24	6.20	24.21	2-4-2-1
	Low	19.94	5.29	21.96	2-4-1-1

CHAPTER 7

CONCLUSIONS, GUIDELINES AND FUTURE EXTENSIONS

The previous three chapters presented the results of the study and discussed the interpretation of these results. The knowledge that has been gained from these results can now be consolidated. This chapter integrates the various aspects of training and testing to form generalizations on the performance of the neural network model and the GRNN model, and their relationship with the different characteristics of data that were selected for the study. In Section 7.1, the general conclusions that were drawn from the results of the study will be presented. Section 7.2 proposes a set of guidelines for training neural networks that will be of value to other researchers in the field. Section 7.3 discusses some areas for future research.

7.1 Conclusions

This study used simulation to generate data that conform to a given set of characteristics. With the knowledge of a sample's underlying distribution, the study used the true regression model as a benchmark for measuring the performance other models. Since the true model is known, any other model was a misspecification. The idea was to determine how close a misspecification can come to the performance of the true model.

The neural network model and the GRNN model are two such misspecifications. The performance of these two models was compared with the true model and against each other and the differences were tested using statistical analysis. The results of the tests show that the true model was significantly more accurate than both the NN model and the GRNN model. This, of course, was expected. However, in a large majority of the cases, the neural network model produced estimates that were almost as accurate as those of the true model. The MAPE values attained by the neural network model

generally stayed within 2% of the error values produced by the true model. In addition, the neural network model produced estimates that were comparable to those of the upper and lower misspecifications of the true model. This means that the neural network model ranks among the "best" misspecifications of the true model. Thus, the neural network model provides reasonably accurate approximations of the true model.

Given its fine performance, the neural network model consistently outperformed the GRNN model. Most of the test statistics obtained for the difference between the neural network model and the GRNN model were highly significant in favor of the neural network. Only in a few cases, mostly under conditions of high random noise, did the GRNN model provide more accurate estimates than the neural network model.

The performance of the neural network model was found to be affected significantly by random noise. This observation limits the claim that neural networks see through noise and distortion. The accuracy of the model deteriorates as the level of random noise increases. Aside from decreasing the accuracy of the neural network model, higher levels of noise have been found to produce longer training times and more complicated models. These results were produced by neural networks that do not overfit the data.

The effect of outliers on the training and performance of neural networks is similar to that of random noise since outliers represent another form of noise. The effect of an outlier depends on whether the noise introduced by the outlier can rise above the random noise already present in the data, and the information provided by the sample size. This means that an outlier will have its most pronounced effect under conditions of low random noise and small estimation samples. When the sample size is large, or when the level of random noise is high, the effect of an outlier becomes negligible. In addition, neural networks appear to be more sensitive to the magnitude of an outlier than to the

number of outliers. For example, in the second side experiment, the neural network model produced a slightly larger error when a single large (4σ) outlier was introduced than when two small (2σ) outliers were present.

The results suggest that better estimates for high noise data would be provided by the GRNN model. The levels of error for the GRNN were scarcely affected, if at all, by changes in the noise level. This implies that the best GRNN model can be obtained under conditions of high random noise, as well as under low random noise. In addition, this makes the GRNN model an attractive alternative when the estimation sample exhibits high noise.

Not surprisingly, the performance of the GRNN model was only mildly affected by the presence of outliers. The GRNN produced the same level of errors in forecasting even when a different number and magnitude of outliers were introduced. In addition, the GRNN produced the same level of errors in forecasting regular data as in forecasting noiseless data when outliers were present. The only changes that were effected on the MAPEs of the GRNN model by an outlier were in conjunction with changing sample size.

As expected, increased sample size proved to have a positive effect on the forecasting performance of both the neural network and the GRNN model. For these two models, a larger sample size represented additional information on the characteristics of the data. This information was then used to improve the accuracy of the estimate. However, the neural network model appears to use the additional information more efficiently than the GRNN model. This was based on the observation that the neural network achieves bigger reductions in MAPEs than the GRNN model for the same increase in estimation sample size.

In contrast to sample size, multicollinearity showed little or no evidence of having any effect on the forecasting performance of the neural network or the GRNN network. Consequently, multicollinearity had no effect on the training or complexity of a neural network estimate.

The neural network was most successful in estimating the LOG, SQRT, LINEAR, MICHAELIS, EXRIS and LOGISTIC models. For the LOG, SQRT, MICHAELIS and LOGISTIC forms, the neural network model obtained the highest accuracy, the shortest training times, and the least complicated models. The success of the neural network on these forms suggests that the error surface of these forms may consist of a steep valley involving a few dimensions.

The LINEAR and EXRIS functional forms also provided the neural network with highly accurate estimates. However, these models required longer training and more parameters. In these models, the error surface appeared to possess gentle slopes in a large number of dimensions. This would account for the slow convergence and the complicated models observed in training.

The functional forms for which the neural network derived the least accurate estimates were SQUARED and XSQX. For the SQUARED and XSQX models, the interpretation indicates the existence of several local minima that hindered the neural network from discovering better estimates. This would account for the short training time required to obtain the estimates.

7.2 Guidelines

With the knowledge and experience gained from this study, the author offers the following guidelines to assist other researchers in applying neural networks to model data.

1. Use as large an estimation sample as possible. The size of the sample would be subject to the availability of data as well as the availability of resources for processing the data. Note that larger estimation samples require longer training time.

2. If the estimation sample exhibits a high level of random noise, obtain the NN estimate and the GRNN estimate and choose the estimate with the smaller error.

3. If the estimation sample contains potential outliers that can adversely affect the estimate, obtain the NN estimate and the GRNN estimate and choose the estimate with the smaller error.

4. Consider using the downhill simplex method to improve the initial set of weights. This can significantly shorten the time that backpropagation requires to converge to an appropriate set of weights.

5. Use an oversized network for the initial configuration and apply pruning to obtain the minimal configuration. Consider the "average networks" presented in Chapter 6 as the target configuration for training.

6. Perform pruning, not only to reduce the complexity of the model, but also to shorten training time.

7. Use large values to initialize the learning rate. This will allow the gradient to escape local minima.

8. During training, adjust the learning rate as dynamically as possible. A dynamically adjusted learning rate speeds up training, and obtains more accurate estimates

9. Use the simplex method after backpropagation to fine tune the set of weights that backpropagation has derived.

10. Improvements in accuracy may be gained by substantially increasing the complexity of the model. This can be accomplished by reducing the value of the coefficient of decay.

11. Use scaling to transform the data into an image that resembles the activation function. For example, if the neural network provides inaccurate estimates for a sample of data that appears concave upward, like the SQUARED data, use reverse scaling to present the data to the neural network as concave downward, like the SQRT data.

7.3 Future Extensions

This study investigated only a small portion of the problem of using neural networks for function approximation. The field is wide open for future research.

A logical sequel to this study would be to apply neural networks to simulated data from more complicated functions. These functions may contain more independent variables, more complicated expressions and more constraints than those considered in this study. Different types of variables can be used. This would include continuous variables as well as binary, integer, qualitative and other types. The functions may contain discontinuities, may be piecewise linear or have some other characteristic not present in more common functions.

Another direction of research would be to apply neural networks directly to real data that have no known distributions. These data appear in economics, biostatistics, the

social sciences and other fields. The value of neural networks would ultimately be evaluated based on how well the method estimates real data.

A host of other problems on neural network modeling remains to be explored. A short list would include questions on variable subset selection, alternative optimization methods, enhancements on the gradient technique, parameter interpretation, and alternative network structures. It is hoped that the results of this study would be of value to other researchers in this very dynamic field.

APPENDIX A

A REVIEW OF NEURAL NETWORKS

A.1 What is a Neural Network?

Neural networks are input-output models based on how the brain is thought to process information. Neural networks are sometimes called artificial neural networks to distinguish them from their biological counterpart.

The biological neural network is a representation of the human nervous system. The human nervous system is built of cells, called neurons, operating in parallel at any given moment. The number of neurons is estimated to be between 10^{11} and 10^{14} each with about 10,000 connections to other neurons. The signals that a neuron receives are conducted to the cell body and summed. Some inputs tend to excite a cell, other inhibit the cell's firing. When the cumulated excitation level exceeds a certain threshold or bias, the cell fires, sending a signal down the connections to other neurons (Wasserman, 1989, p.12). Although the biological neuron is a more complex computing device than this model illustrates, most neural networks model only these simple characteristics. Apart from these characteristics, artificial neural networks have very little in common with biological neural networks.

Hecht-Nielsen (1988) offers the following general, yet rigorous, definition of an (artificial) neural network:

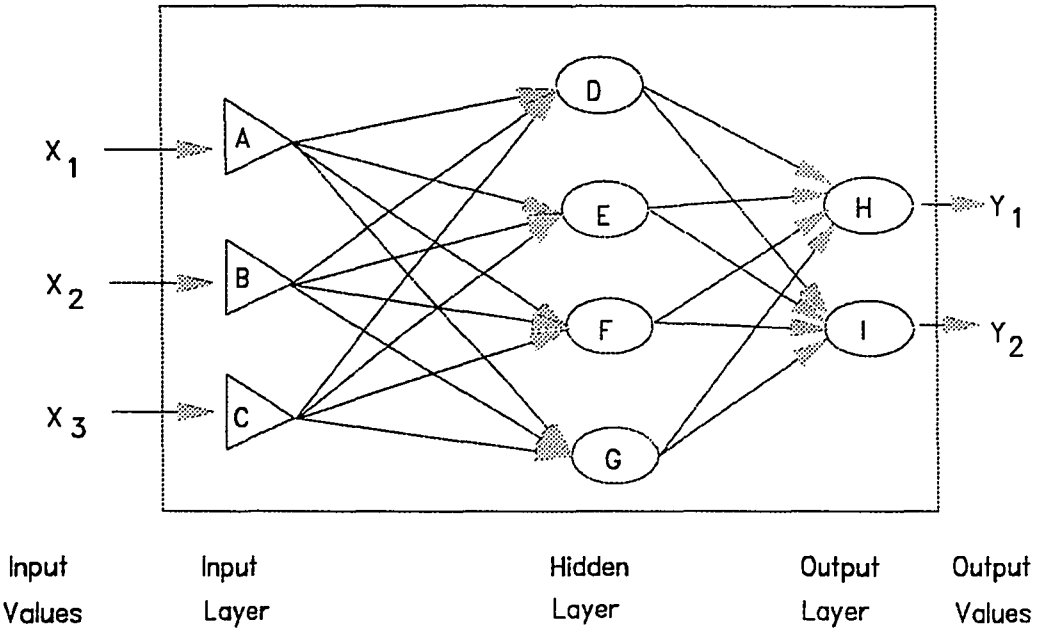
> A neural network is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and carry out localized information processing operations) interconnected together with unidirectional signal channels called connections. Each processing element has a single output connection which branches ("fans out") into as many collateral connections as desired (each carrying the same signal -- the processing element output signal). The processing element output signal can be any mathematical type desired. All of the processing that goes on within each processing element must be

completely local; i.e., it must depend only upon the current values of the input signal arriving at the processing element via impinging connections and upon values stored in the processing element's local memory.

Figure A.1 gives a picture of a neural network as described in Hecht-Nielsen's definition. Just as in other networks, a neural network is composed of nodes and branches. The nodes are grouped into layers with branches connecting the nodes in one layer to some or all of the nodes in the next layer.

A neural network has one input layer, one output layer, and zero, one or more middle or hidden layers. The neural network in Figure A.1 is composed of 9 nodes and 20 connections. Nodes A, B, C comprise the input layer. The input layer receives the signals that come from the network's environment. These signals are passed on to the next layer (nodes D, E, F, and G) for processing, which in turn passes on their own signals to the next layer. The last or output layer, represented by nodes H and I, then

Figure A.1
A 3-4-2 Neural Network



composes the neural network's response or output signals and transmits these to the environment.

The "knowledge" possessed by a network is stored or represented in the pattern of variable interconnection weights between the nodes (Caudill, 1990, p.7). These weights are obtained by training the network to associate a set of input values with the desired output values. The network adapts the weights to reflect the changes as new information is added to the network. This adaptation capability provides the premise for the neural network's ability to generalize or learn the characteristics of a set of data based on a series of examples from that set. Furthermore, the neural network is claimed to be able to provide information from incomplete, noisy, or partially incorrect input data.

If the connections from the output of any layer extend only in the direction of higher-number layers, then the network is said to be nonrecurrent or feedforward. Networks that allow feedback connections, connections that extend from the output of one layer to the input of the same or previous layers, are said to be recurrent. This study will consider only feedforward networks.

A.2 The Topology of a Neural Network

One of the initial decisions that any user of neural networks will have to make concerns the topology or geometry of the network. The topology consists of the arrangements of the nodes in layers, the number of nodes in each layer, and what connections exist between the nodes of one layer and those of the next layer.

The literature does not agree on the manner of counting the number of layers in a neural network. In this paper, an m-layer neural network will have:

- layer 0 as the input layer,
- layer m as the output layer and

- layers 1 to m-1 as the hidden layers.

Since the input and output layers are always present, we can also refer to this m-layered network as an (m-1)-hidden layer network. The input layer is not being counted along with the other layers since the nodes in the input layer do not perform any processing. The input nodes simply receive the signals from the environment and forward these values to the specified nodes in the next layer. With this manner of counting, an m-layer network will have

m = number of layers in the neural network
= number of layers of processing elements
= number of layers of weights

Another way of describing the topology of an m-layer network is by giving the number of nodes in each layer starting with the input layer. Thus, the 2-layer neural network in Figure A.1 can also be declared as a 3-4-2 neural network.

At present, there is no established method of determining the appropriate number of hidden layers nor the correct number of nodes in each layer of an efficient neural network. The number of layers is generally dependent on the depth at which the features lie hidden within the data. The first hidden layer is believed to distinguish the most prominent (highest-level) features of the data. The lower the level of a feature, the more hidden layers may be needed to generalize this feature. Although the same low-level feature may be detected by a single hidden layer with a sufficient number of nodes, the large number of nodes usually involved renders this proposition less practical than a neural network with multiple hidden layers.

On the other hand, the use of too many hidden layers makes a neural network unnecessarily complicated and extremely difficult to train and analyze. The presence of any low-level feature may not compensate for the overall increase in complexity.

Because of this, most of the neural networks put into application or testing utilize only one or two hidden layers. The number of hidden layers used seldom exceeds four.

The number of nodes to be installed in any layer is another problem that is receiving a lot of attention. Generally, two hidden nodes are required for each bump or peak in the function that it is trying to approximate. Of course, if the nature of the true distribution is unknown, the process of determining the number of nodes and layers is usually reduced to trial and error.

Here are some guidelines to remember when determining the number of nodes in a neural network. The number of nodes cannot be too few. Otherwise, the neural network will not have the capacity to generalize the data. Its estimate will result in huge errors. On the other hand, the number of nodes cannot be too large. When there are more parameters in the model than there are data for estimation, the neural network will model the errors and memorize the data for estimation. This is analogous to having a polynomial regression model with the same number of parameters as there are points in the data. The polynomial model can be made to fit the estimation data exactly but it will provide inaccurate estimates for future data.

One rule of thumb is that the data for estimation should contain at least twice the number of points as there are parameters in the network. The more points are available to the nodes in a network, the more information needs to be processed to model the data.

A.3 The Node as Processing Element

In a neural network, most if not all of the computing is done in the nodes. The nodes are also called processing elements (PE), neurons, neurodes, or threshold logic units (TLU). Each node has four important components:

(1) the input connections through which the unit receives signals from other units and which give weights to these signals,

(2) a summation function that combines the various input signals into a single NET signal,

(3) an activation function that converts the NET signal into an OUTput signal, and

(4) output connection that carries the output signal to other units in the system.

Figure A.2 illustrates a processing element. The input signal from all sending nodes, represented by the vector $[X_1, X_2, ...X_n]$ is received via the input connections. The input signals are weighted according to the strength of the connection W_i, and added to a threshold value B to produce the NET input. The net input is then passed through an activation function F() to obtain the output signal OUT. In this study, the activation function F() has been chosen to be the logistic function. That is,

$$OUT = F(NET) = 1/ (1 + \exp(-NET)).$$

A.3.1 The activation function

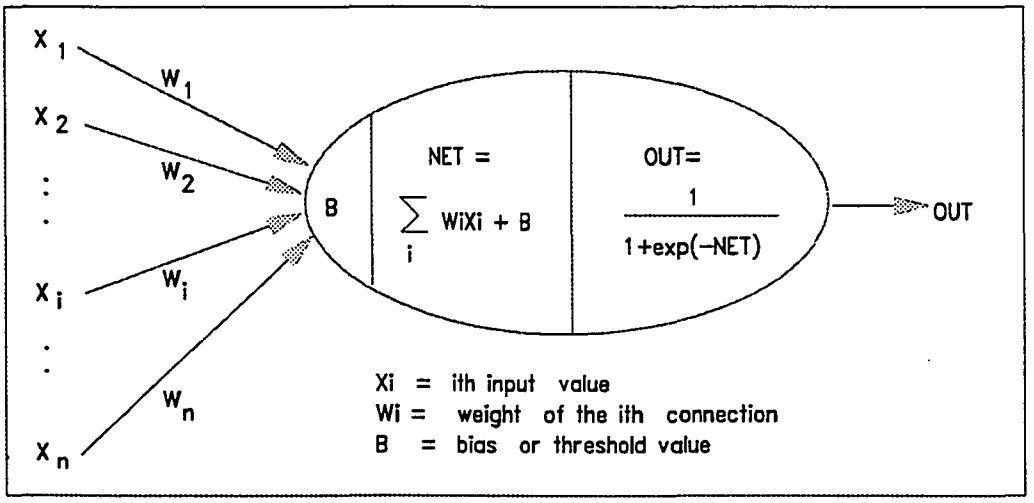
The purpose of an activation function (also called nodal function, threshold function, or squashing function) is to map a node's input domain into some prespecified output range. The choice of activation is very important since the correct choice will result in fewer hidden nodes required to generalize the data, and also fewer data points required to train the network. The two most common types of activation function used are the linear function and sigmoid function.

The linear function has the equation

$$F(X) = k X$$

where k is a real-valued constant. The use of the linear activation function is usually limited to the output node. This is because any multilayer network with solely linear activation functions is equivalent to a single-layer network whose weight matrix is

Figure A.2
A Processing Element (PE)



equivalent to the product of the matrices of the individual layers (Wasserman, 1989, p.19). Unfortunately, single layer linear networks are only applicable to those problems that are linearly separable. A nonlinear activation function is needed to extend the capability beyond linearly separable problems.

Several types of nonlinear function have been implemented in neural networks. Specht (1990) used the exponential function in his probabilistic neural networks (PNN). His research showed that PNNs produce computing decision boundaries that approach the Bayes optimal. In addition, the training of PNNs is simpler and faster.

Gallant and White (1988) used a "cosine squasher" to show that single hidden layer networks can construct a Fourier series approximation to a given function. They implemented the cosine squasher in all the hidden units but used the linear function in the output node. Actually, Stinchcombe and White (1989) proved that neural networks with a single hidden layer, using any general continuous nonlinear function as activation

function and having linear functions at the output layer, are capable of accurately approximating any arbitrary function.

A.3.2 The sigmoid function

Perhaps, the most commonly used nonlinear activation function is the sigmoid function. The sigmoid activation function is a bounded, monotonic, non-decreasing function that provides a nonlinear response.

Several sigmoidal functions have been investigated. The hyperbolic tangent function

$$F(X) = \tanh(X)$$

has been shown to extrapolate well in symbol processing problems (Lapedes and Farber, 1988). Its output lies in the range (-1,1).

The sigmoid function that will be utilized in this study is the logistic function. This function is given by

$$F(X) = 1 / (1 + \exp(-X))$$

and has a range (0, 1). This function has been used extensively in statistics, chemistry and sociology (Simpson, 1990, p.9). The function features an S-shaped curve that approaches 0 and 1 asymptotically. Figure A.3 gives a picture of the logistic function.

Even though the logistic function has a limited range, it can be used to approximate functions with ranges other than (0, 1). This can be accomplished in two ways. First, the response values in the data may be scaled into the interval (0,1). This is done by using a simple mapping formula. The scaled value of any response Y can be obtained by computing

$$\text{scaled } Y = (Y - \min_Y)/(\max_Y - \min_Y)$$

With this formula, the minimum Y value will be scaled to the value 0 while the maximum value will be assigned 1. All the other response values will fall between 0 and 1.

The second approach involves changing the coefficients in the logistic function so that it maps into the desired range. For example, Stornetta and Huberman (1987) found significant improvements in the rate of learning and uniformity of learning by simply making the logistic function symmetric about 0. For this activation function, the desired range is (-1/2, +1/2). This range is achieved by using the logistic function

$$F(X) = [\ 1 / (1 + \exp(-X))\ ]\ - 1/2\ .$$

Besides the range, other characteristics of the logistic function such as the steepness of the curve, the direction of the curve, and the location of the inflection point can be modified to fit any set of data.

A.3.3 Graphical interpretation of the weight and bias

When a processing element uses the logistic activation function, the weight of the connection and the bias in the node attain simple geometric significance. Consider a node with a single input line with weight w, and a bias value of b. For an input value of X,

$$NET = wX + b \qquad \text{and}$$

$$OUT = F(NET) = 1 / (\ 1 + \exp(-(wX+b))\ ) \qquad \text{(A.1)}$$

Since, the inflection point occurs when OUT = 1/2, substituting the value in equation A.1 and solving for X indicates that the inflection point occurs at X = -b/w. Thus, the bias and the weight determine the location of the center of the curve. Figure A.3 illustrates this property.

Differentiating equation A.1 and substituting X = -b/w, it can be shown that the line tangent to the inflection point has a slope of w/4. Aside from contributing to the location of the curve, the weight w determines how steep or flat the curve is, and where the curve faces. If w > 0, the logistic function will be an increasing function. The larger the absolute value of w, the steeper the curve will be.

Figure A.3
The Logistic Function

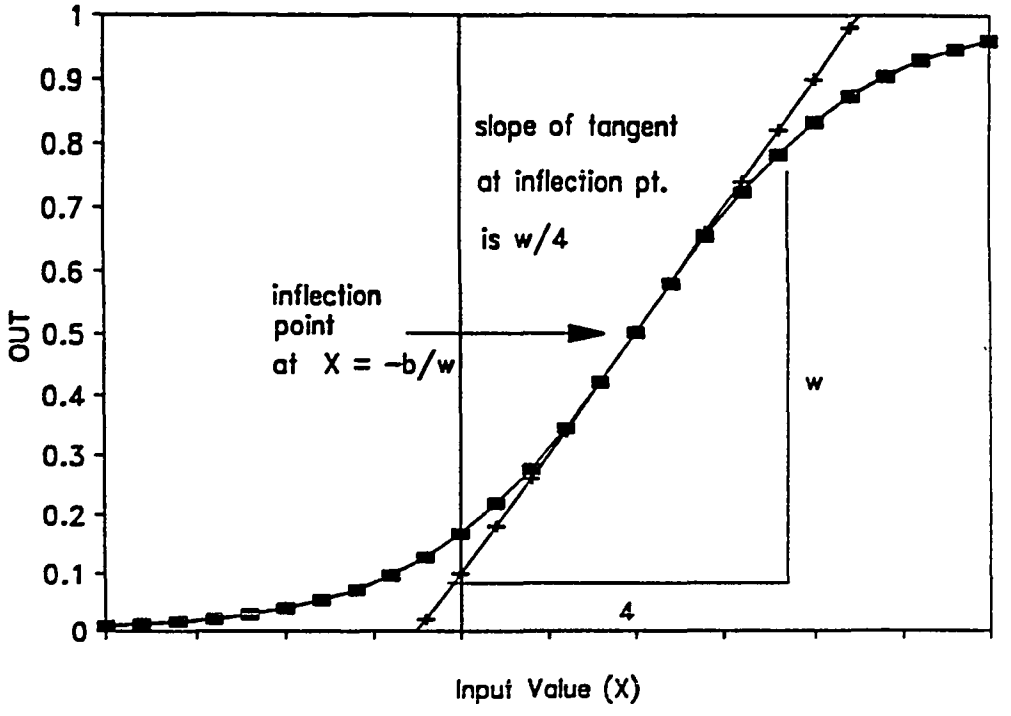
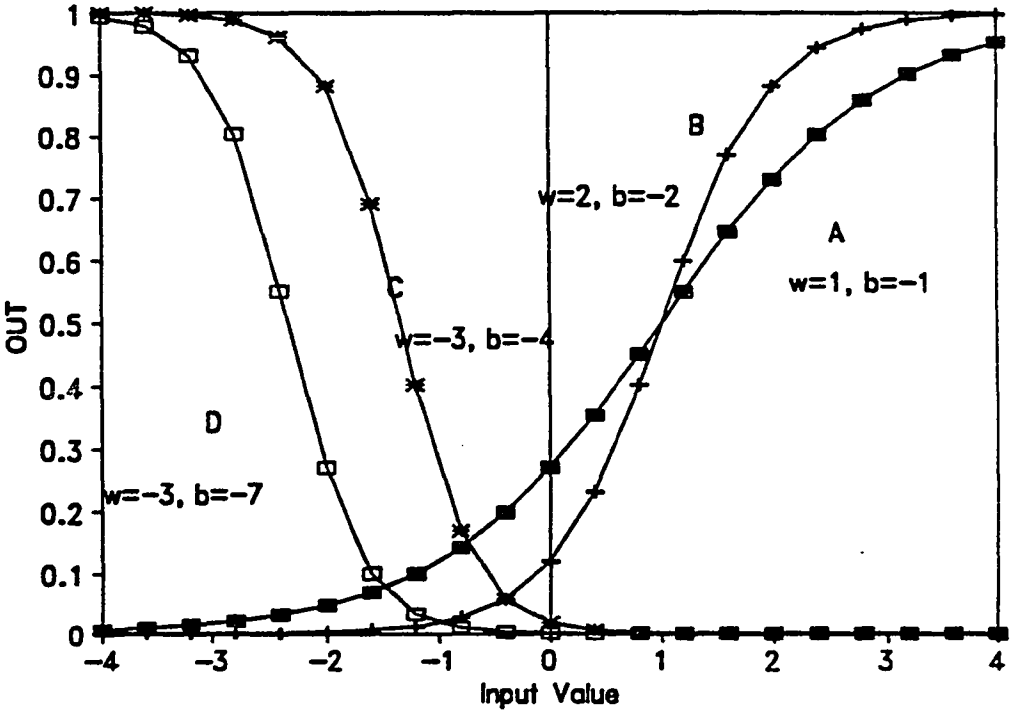


Figure A.4
Weight and Bias for the Logistic Function



To illustrate further the significance of the weights and biases, Figure A.4 shows four logistic curves using different values for the weights w and biases b. Curves A and B both represent positive weights so their ordinates increase to the right. On the other hand, curves C and D have negative weights so their ordinates increase in the opposite direction. Curve A has a lower weight than B so it is less steep. Curves C and D have the same weight values so they have identical steepness. A and B have the same values for -b/w so their points of inflection coincide. Since the value of -b/w for curve D is less than that of curve C, the point of inflection of curve D is located to the left of that of C.

Consider the 1-2-1 neural network shown in Figure A.5. In this network, nodes A and C receive the input values. Nodes A and C then send their output to node B which,

in turn, produces the network output. For an input x, the outputs of nodes A and C are computed as:

$$out(A) = 1 / (1 + exp(-(-1 + 1x))) \text{ and}$$

$$out(C) = 1 / (1 + exp(-(-4 + -3x))).$$

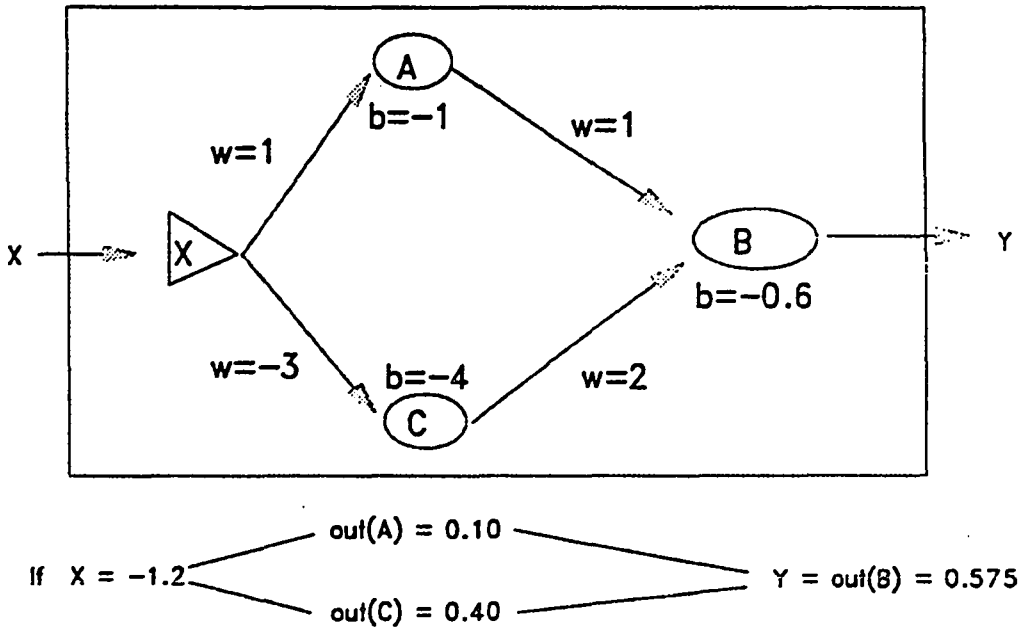
Node B combines the outputs of nodes A and C to produce the network output. This is given by

$$out(B) = 1 / (1 + exp(-(-0.6 + 1*out(A) + 2*out(C)))).$$

For example, if x = -1.2, the output of node A would be 0.10 and the output of node C would be 0.40. Node B then combines these outputs to produce 0.575.

The graph of the output of nodes A, B and C over a range of input values is shown in Figure A.6. Note that the curves for nodes A and C are identical to curves A and C

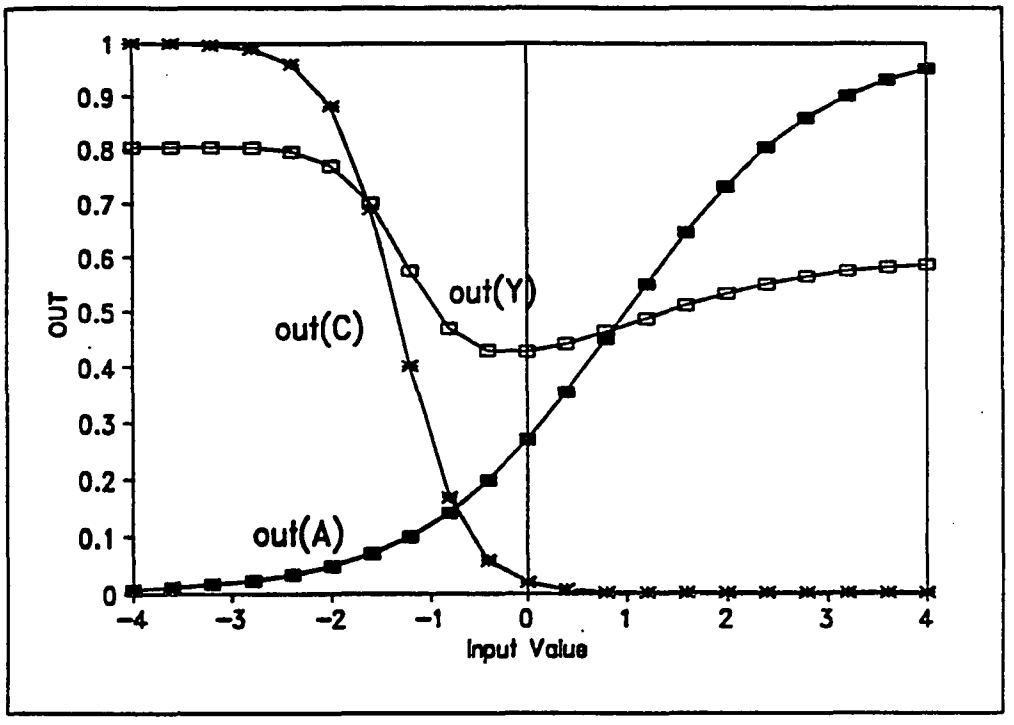
Figure A.5
A 1-2-1 Neural Network



given in Figure A.4. The curve for node B represents one way of combining the curves for nodes A and C so that a "bump" or "valley" can be produced. If the values of the weights and biases in the network are changed, the graph of the network output will also change. Just as a single sigmoid can be used to approximate a monotonic function, then two sigmoids can be combined to produce a curve that possesses a minimum or maximum point. With more sigmoids, more complicated functions may be approximated.

In essence, a neural network is a function embedded in a set of nodes and branches. A neural network can produce a curve that fits a given set of points by combining the different sigmoids possessed by the nodes in the network. The sigmoids are shifted, flattened, turned around, or extended when the weights and bias values are modified. What is desired is a set of values for the weights and biases that combines the sigmoids in a way that minimizes the errors of the fit. The process of obtaining this optimal set of weights and biases is called the training of the network. A review of the mechanics of training a network is given in Appendix B.

Figure A.6
Combining Two Logistic Curves



APPENDIX B

TRAINING A NEURAL NETWORK

Training is the iterative process of deriving the weights of all connections and biases in a neural network. The weights and biases are initially given random values. The process of training involves the presentation of a set of input vectors to the neural network, obtaining the network output for each of these vectors, and then computing the error or difference between the network output and the actual or target vector. The network weights and biases are then adjusted so that this error is minimized. The changes in the weights and biases are computed according to some predetermined training algorithm, usually based on the gradient of the error.

B.1 Supervised Learning versus Unsupervised Learning

Training algorithms can be classified into two categories, supervised learning and unsupervised learning. Supervised learning requires that a target vector representing the desired output be provided along with each input vector. The combination of the input vector and target vector is called the training pair. If **X** is the input vector and **Y** is the target vector, then the training pair is (**X**, **Y**). The set of all training pairs is called the training sample. One input vector is first applied to get the output of the network. The difference between the network output and the actual output is the error. The errors are fed back to the network to modify the weights and biases according to an algorithm that aims to minimize error. The process is repeated using the second training pair, and then the third pair, and so on until the entire training sample has been presented. The presentation of the whole training sample constitutes one epoch of training. Usually, training is repeated over several epochs until the parameters stabilize or until an acceptable error level is attained.

Unsupervised training algorithms do not require target vectors. The training data consists solely of input vectors. The algorithm modifies the network parameters so that the network produces similar outputs when given identical inputs. The training process extracts the characteristics of the training data so that similar inputs are grouped in the same classes (Wasserman, 1989, p.23). This kind of training is also referred to as self-organization.

B.2 The Backpropagation Method

The supervised training algorithm that will be implemented in this study is called backpropagation. Backpropagation consists of a forward pass and a backward pass. When an input pattern is presented, the forward pass will generate the output (OUT) of the network. The difference between the network output and the target output is then propagated backwards to modify the weights of the connections.

The weight adjustment process involves the computation of the gradient of the error function with respect to each weight and bias. For a complete discussion of how this gradient is used to adjust the weights and biases, the reader is referred to Rumelhart, Hinton and Williams (1986) or Hecht-Nielsen (1989). The key element in adjusting the weights in any layer is to realize that the error of each node is a proportionally weighted sum of the errors in the preceding (higher) layer. The following discussion will show how backpropagation is used to adjust the weights of the connections. A similar process occurs when the bias values of the nodes are adjusted.

The term backpropagation stems from the act of propagating the errors backward from the higher layers to the lower layers. The weight adjustments are computed one layer at a time in a top-to-bottom direction. The weight adjustment for the connection from node j to node k, Δw_{jk}, is given by

$$\Delta w_{jk} = \tau * OUTj * d_k$$

where

τ is the learning rate coefficient or step size,

OUT_j is the output of the node j, and

d_k is the gradient adjustment for node k.

The new weight value will be $w_{jk} + \Delta w_{jk}$.

Two types of gradient adjustments are performed. The first type pertains to the nodes in the output layer while the second pertains to the nodes in the hidden layers. The first type adjusts the weights of the all connections leading to nodes in the topmost or output layer. After the first type has been accomplished, then the second type adjusts the weights of all connections in the middle or hidden layers.

For any node o in the output layer, the gradient adjustment is given by

$$d_o = OUT * (1\text{-}OUT) * (TARGET - OUT)$$

where

OUT is the network output, and

TARGET is the target value.

Thus, the weight change Δw_{ho} for the connection from any node h to node o in the output layer is obtained from

$$\Delta w_{ho} = \tau * OUT_h * OUT * (1\text{-}OUT) * (TARGET - OUT).$$

For any node h in a hidden layer, the gradient adjustment is given by

$$d_h = OUT_h * (1\text{-}OUT_h) * \Sigma(w_{hi} * d_i)$$

where

OUT_h is the output of node h,

w_{hi} is the current weight of the connection from node h to any node i, and

d_i is the gradient adjustment for node i.

Similarly, the weight adjustment for the connection from node h in a hidden layer to node i in a higher layer is given by

$$\Delta w_{hi} = \tau * OUT_h * OUT_i * (1-OUT_i) * \Sigma(w_{ij} * d_j)$$

where

τ is the learning rate or step size,

OUT_h is the output of node h,

OUT_i is the output of node i,

w_{ij} is the current weight of the connection from node i to any node j, and

d_j is the gradient adjustment for node j.

To illustrate the backpropagation process, consider the neural network shown in Figure B.1. This neural network was used initially in Figure A.5. Suppose the training pair to be used is (-1.2, 2.64). This means that the input vector is -1.2 while the target or actual vector is 2.64. During the forward pass, the outputs of the three nodes are obtained. From Figure A.5, for an input of -1.2, node A will output 0.10, node C will have 0.40 and the network output will be 0.575.

The backward pass starts with the output node and computes the gradient adjustment for node B. This is found to be

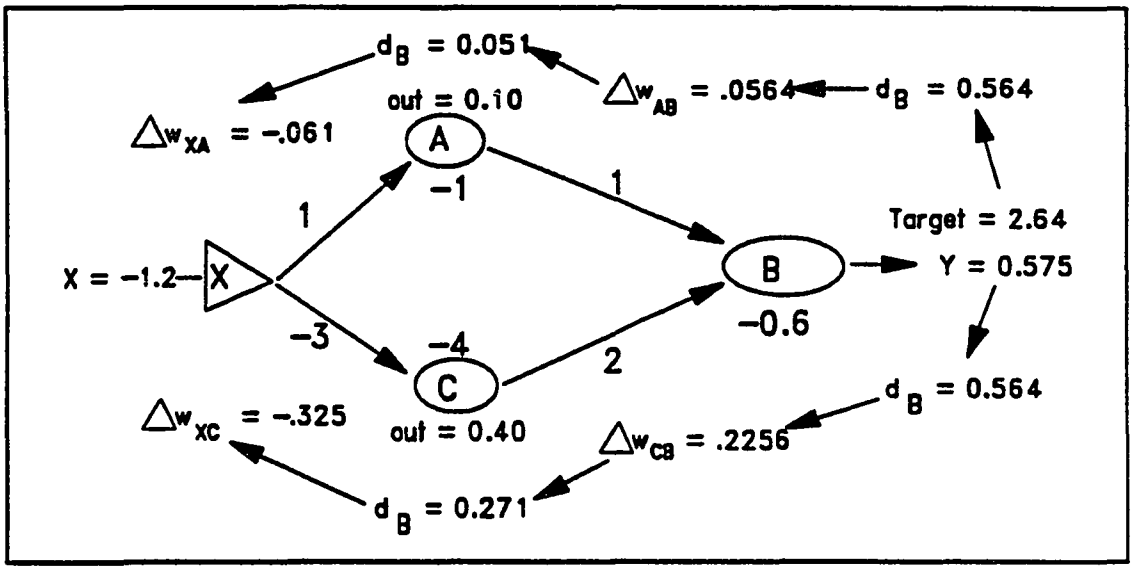
$$d_B = .575 * .475 * (2.64-.575) = 0.564.$$

Using a step size of 1.0 (τ =1), the weight changes for w_{AB} (w_{AB} = 1) and w_{CB} (w_{CB} = 2) are given by

$$\Delta w_{AB} = 1.0 * 0.10 * .564 = .0564 \quad \text{and}$$

$$\Delta w_{CB} = 1.0 * 0.40 * .564 = .2256.$$

Figure B.1
Backpropagation using (-1.20, 2.64)



For the rest of the weights, the second type of gradient adjustment will be used. The gradient adjustment for node A is

$$d_A = .10 * .90 * (1 * 0.564) = 0.051$$

while that for node C is

$$d_C = .40 * .60 * (2 * 0.564) = 0.271.$$

Finally, the weight adjustments for w_{XA} and w_{XC} are obtained from

$$\Delta w_{XA} = 1.0 * -1.2 * 0.051 = -0.061 \quad \text{and}$$

$$\Delta w_{XC} = 1.0 * -1.2 * 0.271 = -0.325.$$

B.3 Weight Updates by Pattern or by Epoch

The above weight adjustments are computed for every training pair in the sample. However, the weights may be updated in two ways. Usually, the weight updates are

performed right after a training pair has been presented. When this is done, training is said to be performed by pattern. From Figure B.1, the updated values of the weights in the network are:

$$w_{AB} = 1 + (.0564) = 1.0564,$$
$$w_{CB} = 2 + (.2256) = 2.2256,$$
$$w_{XA} = 1 + (-.061) = 0.939, \text{ and}$$
$$w_{XC} = -3 + (-.325) = -3.325$$

On the other hand, the weight adjustments may be accumulated over the entire sample after which the total change is applied on the weights. This method of training a network is called training by epoch.

Training by epoch is usually more expensive than training by pattern since it requires additional storage. In cases where the training data continuously change, training may not converge since it may never see the same input vector twice. Unfortunately, training by pattern does not result in a unique neural network (Minsky, 1991). The sequence in which the data is presented affects the adjustment of the weights. This can be solved by presenting the data in a random permutation during every training cycle.

B.4 Problems with Gradient Descent Methods

Since backpropagation is a gradient-descent method, it suffers from the same problems that trouble all gradient-descent procedures. The first problem is that a gradient-descent method performs poorly for surfaces that contain ravines -- places which curve more sharply in some directions than others (Sutton, 1986). In the presence of ravines, large changes in the weights can result in large errors which in turn result in large changes in

weights. Because of this cycling phenomenon, the training process can come to a virtual standstill.

A second problem involves the presence of local minima. Backpropagation follows the direction that minimizes error of the network output and the target vector. This direction may lead to a local minimum, instead of a global minimum. Once backpropagation has reached a minimum, whether local or global, the training stops.

The learning rate or step size needs to be adjusted dynamically in line with the progress in training. The learning rate represents the number of steps in the direction of the gradient that will be taken to adjust the weight. If the learning rate is fixed at a large value, more steps are taken towards the minimum point leading to a faster convergence. However, the gradient will most likely overshoot the minimum resulting in gradient that oscillates back and forth across the minimum point. The response should then be to decrease the learning rate so that the gradient approaches the minimum slowly. In addition, the weight updates should be made automatically. If the adjustments to the learning rate are done manually, the training process would have to be interrupted several times resulting in longer training time.

Finally, gradient descent methods encourage interference between learning with different patterns. When new patterns are introduced, those units that have been found to be most useful are also most likely to be changed (Sutton, 1986). This condition is called temporal instability. When this occurs, the network gains no benefit since the network learns a new pattern while forgetting some previous pattern.

B.5 Improvements on the Backpropagation Method

Numerous improvements and variations have been proposed on the backpropagation method that are designed to solve the problems enumerated in the previous section.

B.5.1 Getting good initial weights and biases

Since the final state of the network depends upon the initial choice of weights, a good set of initial weights and biases can go a long way in speeding up training and avoiding many of the pitfalls present in the error surface. It is suggested that the weights be initialized to small random numbers so as to avoid being saturated by large values of weights. The initial weights and biases also need to be unequal for the network to exhibit learning.

The usual procedure is to select the weights and biases randomly. Under these conditions, gradient descent is not much better than a random search since in general the local gradient will not be a good predictor of the appropriate direction (Sanger, 1989). Nguyen and Widrow (1990) proposed a method for assigning weights so that each connection will be responsible for some subinterval of the output range. Their results show that such a method can dramatically decrease training time.

B.5.2 Search optimization

Many researchers have devised extensions to the basic backpropagation algorithm with the primary purpose of finding the optimal search path to improve. Cater (1987) used 10 to 30 times the size of the calculated gradient to increase the learning rate. Line search optimization techniques have been proposed by Jones et al. (1990) to determine a better learning rate, and by Watrous (1987) to make the optimal size change in each training iteration.

Parker (1987) used second derivatives to produce a more accurate estimate of the appropriate weight adjustment. He called his method second-order backpropagation. Sandon and Uhr (1988) proposed a "local interaction" heuristic for escaping local minima during training. The use of conjugate gradient techniques (Battiti, 1990) as well

Newton's method (Watrous, 1987; Le Cunn, 1988; Parker, 1987) have also been investigated.

B.5.3 Momentum

Rumelhart, Hinton et al. (1986) proposed a method of improving the training time, getting over local minima, and enhancing stability by introducing a "momentum" term into the weight adjustment. The weight adjustment will contain the change based in the gradient plus a fraction of the last weight change used. That is,

$$\Delta w_{ij} = \tau * OUT_i * d_j + \alpha * (\text{last weight adjustment})$$

where α is called the momentum coefficient. The addition of the momentum term makes the weight adjustment "remember" the adjustment made in the previous iteration. This has the effect of making the network follow the bottom of narrow gullies in the error surface rather than crossing rapidly from side to side (Wasserman, 1989, p.54). This study will implement weight adjustments that will include the momentum term.

APPENDIX C

GENERATING A DATA POINT

To generate a data point (x_0, y_0) for the linear functional form

$$Y = X + \varepsilon$$

with a specified level of R^2, we use the method due to Scheuer and Stoller (Law and Kelton, 1982, p.269). The method produces a pair of variables that are normally distributed with mean μ and standard deviation σ.

The first step is to obtain two random numbers U_1 and U_2 (that is, U_1 and U_2 are U(0,1)) obtained using the random number generator of any programming language. For this study, an improved random number generator using the code given in Press et al. (1988, p.207) was implemented. Then using either the Box and Muller method or the polar method (Law and Kelton, 1982, p.259), two standard normal random variables Z_1 and Z_2 are obtained from U_1 and U_2. This means, Z_1 and Z_2 are IID N(0, 1).

Then, if we let

$$x_0 = \mu + Z_1$$

$$y_0 = \mu + \sqrt{R^2} * Z_1 + \sqrt{1-R^2} * Z_2$$

$$\varepsilon_0 = y_0 - x_0$$

We get x_0 and y_0 which are normally distributed with mean μ and variance σ^2, and have covariance $((1 / R^2) - 1)$.

For this study, data will be generated so that $\mu = 5$ and $\sigma = 1.0$. In addition, estimation samples are generated to represent three levels of noise. For data with high noise, R^2=30%. Those with medium noise have R^2 = 60%, and for low noise, R^2 = 90%. Some of the testing data will be noiseless (R^2 = 100%), and some will be noiseless and covers a wider range of values (R^2 = 100%, σ = 1.25).

To obtain a data point for the nonlinear functional form

$$Y = f(X) + \varepsilon$$

we only need to adjust the error to conform to the desired R^2. For example, to produce data points for the model

$$Y = \sqrt{X} + \varepsilon$$

we first obtain the values x_0, y_0 and ε_0 that conform to the linear model as described above. Then, compute

$$y_0' = \sqrt{x_0} + 0.21 * \varepsilon_0$$

(x_0, y_0') will be the desired data point. The error inflator value of 0.21 had been determined empirically for the square root model and may differ for the other functional forms.

APPENDIX D

INITIAL SETTINGS OF THE NETWORK

To enable other researchers to verify the results of this study, the initial settings of the neural network used in this study are described. These settings follow closely the terminology used in the BP program given in Rumelhart and McClelland (1988). Unless otherwise specified in the following declarations, the default values in the BP program were used.

The initial structure consists of a 1-4-2-1 network (2-4-2-1 in the bivariate case) with each node connected to all nodes in the upper layers. The corresponding network file for the BP program contains the following declaration:

```
definitions:
nunits 8
ninputs 1
noutputs 1
end
network:
%r 1  7 0 1
%r 5  3 1 4
%r 7  1 5 2
end
biases:
%r 1 7
end
```

Other training parameters that have corresponding variables in the Rumelhart and McClelland BP program are set as follows (the BP program variable is presented in brackets):

- learning grain [mode/lgrain] = pattern
- learning rate [param/lrate] = 2.0
- momentum [param/momentum] = 0.5
- presentation = strain

- tolerance level [ecrit] = 0.0001 (This setting is used for the MAPE value not MSE).
- number of epochs per cycle of training [nepochs] = 10

Note that the learning rate and momentum are updated automatically in this study but will have to be manually changed in the BP program.

Other parameters used in this study include a coefficient of decay that starts at 0.0, increases to 2.0 in increments of 0.05, and then decreases to 0.0 in increments of 0.05. For the simplex method (Press et.al, 1988), the maximum number of iterations is 5000. The ALPHA parameter is set to 1.0, the BETA parameter is set to 0.5, and the GAMMA parameter is set to 2.0.

APPENDIX E

CPU TIMES FOR TRAINING

The following table gives representative values of the minimum and maximum computer times for training the neural network used in this study. The values are in milliseconds (msec.) and represent the CPU time used to perform one epoch of training for the downhill simplex and the backpropagation. Since the CPU time for one epoch of training is affected by the number of input nodes and the size of the sample for training or estimation, the CPU times for the different instances of these parameters used in this study are given.

Table E.1
CPU Times per Epoch of Training

Number of Input Nodes	Training Sample Size	Front Simplex		Backpropagation		Back Simplex	
		Min (msec)	Max (msec)	Min (msec)	Max (msec)	Min (msec)	Max (msec)
1	15	2.23	2.29	9.52	10.37	0.76	3.00
1	30	4.20	4.33	15.63	19.43	1.49	4.76
1	60	8.07	9.46	37.84	40.43	8.57	11.26
2	60	8.99	10.36	41.11	474.70	3.45	10.00

APPENDIX F

PSEUDOCODE OF THE BACKPROPAGATION PROCEDURE

/* Constants used in the procedure

DECAYINC = value used to increase or decrease the coefficient of decay. The default value is .05.

FALSE = 0.

LRATEDIV = value used to divide the learning rate. The default value is 2.

LRATEMUL = value used to multiply the learning rate. The default value is 1.3.

MAXDECAY = maximum value for the coefficient of decay. The default value is 0.20.

MAXTMAPE = maximum acceptable MAPE for training. If the MAPE obtained exceeds MAXTMAPE, training will be attempted again using a new set of random weights. The default value is 0.20.

MAXTRIALS = maximum number of times that training will be attempted. The default value is 3.

MINMOM = minimum nonzero momentum value. When momentum drops below MINMOM, momentum is set to zero. The default is 0.1.

MOMDIV = value used to divide momentum. The default is 4.

TEMPFILE = temporary file used to store the weights during training.

TRUE = 1.

*/

```
/* Beginning of backpropagation procedure */

 if (number_of_patterns< 1)
  print (" WARNING! No training patterns available.");
  exit;

 set all counters to 0;
 set all sums to 0;
 set dirdecay to 1; /* increase decay up to 2.0 */
 set orig_momentum to default_mom;
 set orig_lrate to default_lrate;
 set orig_decay to default_decay;
 set lastrun to FALSE;

 index patterns in PERMUTATION or SEQUENTIAL order;

 while ( ntrials<1  OR (mape>MAXTMAPE AND ntrials<MAXTRIALS)
        OR lastrun ))
  {
   smape to 0.0;
   ncomps to 0;
   contmom to TRUE;
   dirdecay to 1; /* increase decay up to MAXDECAY */
   reset standard error of weights;

   if (ntrials>0)
   {
    set momentum to orig_momentum;
    set lrate to orig_lrate;
    setdecay to orig_decay;

    if (lastrun)
       set lastrun to FALSE;
         set parameters to those of the best trial;
       read weights from TEMPFILE;
    else
       set weights randomly;

    reset_statistics;
   };

   perform simplex to improve weights;
```

```
while (decay >= 0)
{
   repeat (nepochs  times)
    {
     for each pattern(i)
     {
       forward_propagate pattern[i];
        compute weight_delta values for outer layers;
        compute weight_delta values for hidden layers;
        change weight values;
     };
       increment nepochs;
     increment total_epochs;
    };

  if ( momentum <= 0.0)
  {
     compute delta_mape;

     if (abs(delta_mape) <=  error_criterion)
     {
         if ( decay >= MAXDECAY) set dir_decay to -1;
         decay += dirdecay*DECAYINC;
         compute standard error of weights;
         set redundant weights to zero;
     }
     else if (delta_mape >  ecrit)
       divide lrate by LRATEDIV;
     else
         multiply lrate by LRATEMUL;

     if (decay >= 0)
     {
       compute mape;
       set momentum to orig_momentum;
       reset standard error of weights;
     };

    }
  else
  {
      divide momentum by MOMDIV;
       if (momentum < MINMOM)  set momentum to 0;
   };
};

compute mape;
compute standard error of weights;
increment ntrials;
```

```
   if ((ntrials==1) OR (mape < best_tmape))
   {
      store parameters of best trial;
      store weights in TEMPFILE;
   }
   else if (mape>best_mape) AND (ntrials==MAXTRIALS))
      set lastrun to TRUE;
};

 prune the network;
 perform simplex to improve weights;

/* end of backpropagation procedure */
```

(NOTE: The Multipurpose Backpropagation Package (MBP) used in this study is available from the author. For information, please write to:

Leorey O. Marquez
Department of Decision Sciences
University of Hawaii
2404 Maile Way
Honolulu, HI 96822
USA

or send an e-mail message to leo@uhunix.uhcc.hawaii.edu.)

BIBLIOGRAPHY

Atkinson, A.C. (1985). *Plots, Transformations, and Regression.* Oxford: Oxford Publishing.

Bates, D.M. and D.G. Watts (1988). *Nonlinear Regression Analysis and its Applications.* New York: Wiley.

Battiti, Roberto (1990). Optimization Methods for Back-Propagation: Automatic Parameter Tuning and Faster Convergence. *Proceedings of the International Joint Conference on Neural Networks* **1** (Washington DC, January 1990):593-596.

Bell, T., G. Ribar, and J. Verchio (1989). Neural Nets vs. Logistic Regression. USC Expert Systems Symposium (November 1989). Peat Marwick Research Group.

Box, G. and D. Cox (1964). An Analysis of Transformations. *Journal of the Royal Statistical Society* **B26**:211-243.

Box, G. and P. Tidwell (1962). Transformations of the Independent Variables. *Technometrics* 4:531-550.

Burr, D.J. (1987). Experiments with a Connectionist Text Reader. *Proceedings of the First International Conference on Neural Networks* **4** (San Diego, CA):717-724.

Burr, D. (1986). A Neural Network Digit Recognizer. *Proceedings of the 1986 IEEEE International Conference on Systems, Man and Cybernetics.* 1621-1625.

Carroll, S.M. and B.W. Dickinson (1989). Construction of Neural Nets Using the Radon Transform. *Proceedings of the 1989 International Joint Conference on Neural Networks* **1** (June 1989):607-611.

Cater, John (1987). Successfully Using Peak Learning Rates of 10 (and Greater) in Backpropagation Networks with the Heuristic Learning Algorithm. *Proceedings of the First International Conference on Neural Networks* **2**:645-651.

Caudill, M. and C. Butler (1990). *Naturally Intelligent Systems*, Cambridge, MA: MIT Press.

Chester, Daniel (1990). Why Two Hidden Layers are Better than One. *Proceedings of the International Joint Conference on Neural Networks* **1** (Washington DC, January 1990):265-268.

Cottrell, G., P. Munro, and D. Zipser (1987). Learning Internal Representations from Gray-Scale Images: An Example of Extensional Programming. *Ninth Annual Conference of the Cognitive Science Society* (Seattle 1987), 462-473.

Cybenko, G. (1988). Continuous Valued Neural Networks with Two Hidden Layers are Sufficient. Technical Report, Department of Computer Science, Tufts University.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems* **2** (4):303-314.

Daniel, C. and F. Wood (1980). *Fitting Equations to Data.* 2nd edition. Wiley-Interscience.

Duliba, Katherine (1991). Contrasting Neural Nets with Regression in Predicting Airline Performance. *Proceedings of the 24th Hawaii International Conference on System Sciences* **4** (January 1991):163-170.

Dutta, S. and S. Shekhar (1988). Bond Rating: A Non-Conservative Application of Neural Networks. *Proceedings of the International Conference on Neural Networks* **2** (July 1988):443-450.

Fishwick, Paul, (1989). Neural Network Models in Simulation: A Comparison with Traditional Modeling Approaches. UF-CIS Technical Report TR-89-8 (July 1989), University of Florida.

Forgionne, G.A. (1983). Corporate Management Science Activities: An Update. *Interfaces* **13** (3):20-23.

Friedman, Jerome (1991a). Multivariate Adaptive Regression Splines. *Annals of Statistics* **19** (1):1-141.

Friedman, Jerome (1991b). Adaptive Spline Networks. Technical Report No. 107, Stanford University.

Funahashi, K.I. (1989). On the Approximate Realization of Continuous Mappings By Neural Nets. *Neural Networks* **2** (3):183-192.

Gallant, A.R. (1987). *Nonlinear Statistical Models.* John Wiley and Sons.

Gallant, A.R. and H. White (1988). There Exists a Neural Network That Does Not Make Avoidable Mistakes. *Proceedings of the International Conference on Neural Networks* **1** (July 1988):657-666.

Hartley, H.O. (1961). The Modified Gauss-Newton Method for the Fitting of Nonlinear Regression Functions By Least Squares. *Technometrics* **3**:269-80.

Hecht-Nielsen, R. (1988). Applications of Counterpropagation Networks. *Neural Networks* **1**:131-140.

Hecht-Nielsen, Robert (1989). Theory of the Backpropagation Neural Network. *Proceedings of the IEEE INNS International Joint Conference on Neural Networks,* vol. I, 593-606.

Hertz, J., A. Krogh, and R. Palmer (1991). *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison-Wesley.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks,* **2**, No. 5, 359-366.

Irie, B. and S. Miyake (1988). Capabilities of Three-Layer Perceptrons," *Proceedings of the IEEE Second International Conference on Neural Networks* **1** (San Diego CA): 641-648.

Jones, K., I. Lustig, and A. Kornhauser (1990). Optimization Techniques Applied to Neural Networks: Line Search Implementation for Back Propagation. *Proceedings of the International Joint Conference on Neural Networks* **3** (San Diego CA, June 1990):933-939.

Kennedy, W. and J. Gentle (1980). *Statistical Computing.* Marcel Dekker Publishing.

Lapedes, A. and R. Farber (1988). How Neural Nets Work. In D. Anderson (ed.), *Proc. of the 1987 IEEE Conference on Neural Information Processing Systems - Natural and Synthetic,* 442-456. New York: American Institute of Physics.

Law, A. and W. Kelton (1982). *Simulation Modeling and Analysis.* McGraw-Hill.

Le Cunn, Y. (1988). Using Curvature Information to Improve Backpropagation. *Neural Networks Supplement: INNS Abstracts,* **1**:168.

Ledbetter, W. and J. Cox (1977). Are OR Techniques Being Used. *Industrial Engineering* **9** (2):19-21.

Lippman, R.P. (1987). An Introduction to Computing with Neural Nets. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **2** (4):4-22.

Marquez, L., T. Hill, R. Worthley, and W. Remus (1991). Neural Network Models as an Alternative to Regression. *Proceedings of the 24th Hawaii International Conference on System Sciences* **4** (January 1991):129-135.

McCulloch, W.S. and W. Pitts (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulleting of Mathematical Biophysics* **5**:115-133.

Minsky, Marvin (1991). Article 2823 in comp.ai.neural-nets, message-ID:<4962media-lab.MEDIA.MIT.EDU>, Jan. 21, 1991.

Minsky, M.L. and S.A. Papert (1969). *Perceptrons.* Cambridge: MIT Press.

Mosteller, F. and J. Tukey (1977). *Data Analysis and Regression.* Addison-Wesley Publishing.

Nelder, J. and R. Mead (1965). The Downhill Simplex Method. *Computer Journal* **7**:308-310.

Neter, J., W. Wasserman, and M. Kutner (1989). *Applied Linear Regression Models.* 2nd Edition. Irwin Publishing.

Nguyen, D. and B. Widrow (1990). Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights. *Proceedings of the International Joint Conference on Neural Networks* **3** (San Diego CA, June 1990): 21-26.

Odom, M., and R. Sharda (1990). A Neural Network Model for Bankruptcy Prediction. *Proceedings of the International Joint Conference on Neural Networks* **2** (San Diego CA, June 1990):163-168.

Parker. D. (1985). Learning Logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA.

Parker, David (1987). Second Order Back propagation: Implementing an Optimal O(n) Approximation to Newton's Method as an Artificial Neural Network. *Computer*, in review.

Press, W., B. Flannery, S. Teukolsky, and W. Vetterling (1988). *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge: Cambridge University Press.

Remus, William and Timothy Hill (1990). Neural Network Models of Managerial Judgment. *Proceedings of the 23rd Hawaii International Conference on System Sciences* **4** (January 1990):340-344.

Rosenblatt, F. (1957). The Perceptron: A Perceiving and Recognizing Automation Project (Project PARA). Cornell Aeronautical Laboratory Report, 85-460-1.

Rosenblatt, F. (1962). *Principles of Neurodynamics.* New York: Spartan.

Roy, Jean and Jean-Claude Cosset (1990). Forecasting Country Risk Ratings Using a Neural Network. *Proceedings of the 23rd Hawaii International Conference on System Sciences* **4** (January 1990):327-334.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams (1986). Learning Internal Representations by Back-Propagating Errors. *Nature* **323**:533-536.

Rumelhart, D.E., and J.L. McClelland (1988). *Explorations in Parallel Distributed Processing,* Cambridge: MIT Press.

Sandon, P. and L. Uhr (1988)," A Local Interaction Heuristic for Adaptive Networks," *Proceedings of the IEEE International Conference on Neural Networks* **1**:317-324.

Sanger, Terence (1989). Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network. *Neural Networks* **2**:459-473.

Sejnowski, T. and C. Rosenberg (1987). Parallel Networks that Learn to Pronounce English Text. *Complex Systems* **1**:145-168.

Sharda, P., and R.B. Patil (1990a). Neural Networks as Forecasting Experts: An Empirical Test. *Proceedings of the . International Joint Conference on Neural Networks* **2** (Washington D.C., January 1990):491-494.

Sharda, R., and R. Patil (1990b). Connectionist Approach to Time Series Prediction: An Empirical Test. Working Paper 90-26 11-90 (Nov. 1990), Oklahoma State University.

Sietsma, J. and R. Dow (1991). Creating Artificial Neural Networks That Generalize. *Neural Networks* **4** :67-79.

Simpson, Patrick K. (1990). *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations.* New York: Pergamon Press.

Smith, A. and C. Dagli (1990), Backpropagation Neural Network Approaches to Process Control: Evaluation and Comparison. Working Paper Series #90-22-47, Department of Engineering Management, Univesity of Missouri-Rolla.

Specht, Donald (1990). Probabilistic Neural Networks. *Neural Networks* **3**:109-118.

Specht, Donald (1991). A General Regression Neural Network. *IEEE Transactions on Neural Networks* **2** (6):568-576.

Stinchcombe, M. and H. White (1989). Universal Approximation Using Feedforward Networks with Non-Sigmoid Hidden Layer Activation Functions. *Proceedings of the International Joint Conference on Neural Networks* **1** (June 1989):613-617.

Stornetta, W. and B. Huberman (1987). An Improved Three-Layer Back Propagation Algorithm. *Proceedings of the First IEEE International Conference on Neural Networks* **2** (August 1987):637-644.

Surkan, A.J. and J.C. Singleton (1990). Neural Networks for Bond Rating Improved by Multiple Hidden Layers. *Proceedings of the International Joint Conference on Neural Networks* **2** (San Diego CA, June 1990):157-162.

Sutton, Richard (1986). Two Problems with Backpropagation and other Steepest-Descent Learning Procedures for Networks. *Proceedings of the 8th Annual Conference of the Cognitive Science Society*. 823-831.

Szu, H. (1986). Fast Simulated Annealing. In *Neural Networks for Computing* (Snowbird 1986), ed. J.S. Denker, 420-425. New York: American Institute of Physics.

Thodberg, Hans (1991). Improving Generalization of Neural Networks Through Pruning. *International Journal of Neural Systems* **1** (4):317-326.

Wasserman, Philip (1989). *Neural Computing Theory and Practice*, Van Nostrand Reinhold Pub.

Watrous, Raymond (1987). Learning Algorithms for Connectionist Networks: Applied Gradient Methods for Nonlinear Optimization. *Proceedings of the First International Conference on Neural Networks* **2** (June 1987):619-627.

Weigend, A., Huberman, B. and D. Rumelhart (1990). Predicting the Future: A Connectionist Approach. *International Journal of Neural Systems* **1** (3):193-209.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. Thesis, Harvard University.

White, Halbert (1989a). Learning in Artificial Neural Networks: A Statistical Perspective. *Neural Computation* **1**:425-464.

White, Halbert (1989b). Neural Network Learning and Statistics. *AI Expert*. (December 1989):48-52.

White, Halbert (1989c). Some Asymptotic Results for Learning in Single Hidden-Layer Feedforward Network Models. *Journal of the American Statistical Association* **84** (408):1003-1013.

Yoon, Barbara (1989). Artificial Neural Network Technology. *DARPA*. February 1989.