UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Neural networks for time series analysis

DISSERTATION

BY

K du Plessis

8635498

**Submitted in fulfillment of part of the
Requirements for the degree of**

Master of Mathematical Statistics

in the

Department of Statistics

of the

Faculty of Science

at the

University of Pretoria

Promoter: Dr H. Boraine

Co-Promoter: Dr J.E.W. Holm

January 2000

Acknowledgements

1. Dr Hermi Boraine, my promoter, for her guidance, help and endless patience.

2. Dr Johan Holm, my co-promoter, for his advice and encouragement especially in the field of neural networks.

3. My husband and four children for their love and endurance.

4. To my Creator for without Him nothing is possible.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Summary

Neural networks for time series analysis

by

K du Plessis

Promoter: Dr H Boraine
Co-promoter: Dr JEW Holm

Department of Statistics

Submitted in fulfillment of part of the requirements for the degree of Master of

Mathematical Statistics

The analysis of a time series is a problem well known to statisticians. Neural networks form the basis of an entirely non-linear approach to the analysis of time series. It has been widely used in pattern recognition, classification and prediction. Recently, reviews from a statistical perspective were done by Cheng and Titterington (1994) and Ripley (1993).

One of the most important properties of a neural network is its ability to learn. In neural network methodology, the data set is divided in three different sets, namely a *training set*, a *cross-validation set*, and a *test set*. The training set is used for training the network with the various available learning (optimisation) algorithms. Different algorithms will perform best on different problems. The advantages and limitations of different algorithms in respect of all training problems are discussed.

In this dissertation the method of neural networks and that of *ARIMA* models are discussed. The procedures of identification, estimation and evaluation of both models are investigated. Many of the standard techniques in statistics can be compared with neural network methodology, especially in applications with large data sets.

To illustrate the adaptability of neural networks the problem of forecasting is considered. A few alternative theoretical approaches to obtain forecasts for non-linear time series models are discussed. It is shown that bootstrap methods can be used to calculate predictions, standard errors and prediction limits for the forecasts.

Neural network methodology is applied to predict an electricity consumption time series.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Samevatting

Neurale netwerke vir tydreeksanalise

deur

K du Plessis

Studieleier: Dr H Boraine
Medestudieleier: Dr J.E.W Holm

Departement Statistiek

Voorgelê ter vervulling van 'n deel van die vereistes
vir die graad Magister in Wiskundige Statistiek

Die ontleding van tydreekse is 'n bekende probleem vir statistici. Neurale netwerke vorm die basis van 'n geheel en al nieliniêre benadering tot die analise van 'n tydreeks. Dit word in 'n wye spektrum van probleemgebiede gebruik, byvoorbeeld patroonerkenning asook klassifikasie en vooruitskatting. Slegs onlangs is neurale netwerke deur navorsers soos Cheng en Titterington (1994) en Ripley (1993) vanuit 'n statistiese oogpunt beskou.

Een van die belangrikste eienskappe van neurale netwerke is die vermoë om te leer. Neurale netwerkmetodologie behels dat die datastel verdeel word in drie dele, naamlik: 'n leerstel, 'n kruisvalidasiestel en 'n toetsstel. Die leerstel word gebruik om die netwerk af te rig deur gebruik te maak van verskeie leeralgoritmes. Sekere algoritmes presteer beter as ander, na gelang van die tipe probleemsituasie. Die voor- en nadele van leeralgoritmes word bespreek.

In hierdie verhandeling word die metodiek van neurale netwerke en dié van *ARMA* modelle bespreek. Die prosedures van identifikasie, beraming en evaluasie van beide modelle word bespreek. Baie van die standaardtegnieke in statistiek word vergelyk met neurale netwerkmetodologie en spesifiek met die toepassing van groot datastelle.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Om die aanpasbaarheid van neurale netwerke te illustreer, word die probleem van vooruitskatting in oënskou geneem. 'n Paar alternatiewe teoretiese beskouings om vooruitskattinings te maak vir nieliniêre tydreekse, word bespreek. Daar word aangetoon dat *bootstrap*-metodes gebruik kan word om vooruitskattings, standaardfoute en intervalle vir vooruitgeskatte waardes te bereken.

Neurale netwerk metodologie word gebruik om elektrisiteitsverbruik vooruit te skat.

Notation

The following list provides guidance of the notation adopted

Symbol	Commentary
$Y_1, Y_2, \ldots, Y_N$	Time series with N observations
$Y_t = f(\underline{X}_t, \underline{w}) + \varepsilon_t$	Neural network model for time series
$f(.)$	Non-linear function
$\underline{X}_t$	A set of input variables for example: Y_{t-1}, Y_{t-2}, temperature, days …
$\underline{w}$	A set of parameters of dimension W
$\{\varepsilon_t, t = \ldots -1,0,1,\ldots\}$	White noise process
$E(\underline{w})$	Error function
$E(\underline{w}) = \frac{1}{2}\sum_{j=1}^{N}\{y_j - f(\underline{x}_j;\underline{w})\}^2$	Sum of squares error function
$(H)_{ij} = \frac{\partial^2 E(\underline{w})}{\partial w_i \partial w_j}$	ij-element of Hessian of the error function
$\underline{b} = \frac{\partial E(\underline{w})}{\partial \underline{w}}\Big\|_{\underline{w}_0}$	Gradient of error function
$\underline{\gamma} \equiv (\theta_0, \phi_1, \phi_2, \ldots, \phi_p, \theta_1, \theta_2, \ldots, \theta_q, \sigma_a^2)'$	Vector of parameters for ARIMA model
$E(\underline{\gamma}) = -\ln L(\underline{\gamma})$	Error function used for maximum likelihood estimation

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Contents

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 1

Introduction

Detecting trends and patterns in a time series traditionally involves statistical methods such as clustering and regression analysis. However, these methods are linear and may fail to forecast a non-linear data set. Neural networks form the basis of a different approach to the non-linear analysis of time series. This study investigates the use of a linear time series *ARIMA* model in comparison with a non-linear neural network model for application in forecasting.

Chapter 2 deals with the history and development of neural networks. This is done in the framework of a variety of problems, such as classification, pattern recognition and forecasting. The model of a neural network is illustrated along with different types of applied neural networks. Of all the possible networks, multi-layered feedforward neural networks are used in this dissertation.

A very important property of a neural network is its ability to learn, which is the phase during which parameter estimation takes place. The data set is divided into three different sets, namely a *training set*, *cross-validation set*, and a *test set*. Neural networks are data driven in the sense that they use data to train and build statistical models of a recorded process. The training set is used for training the network with various available learning (optimisation) algorithms. These various available learning algorithms are treated in Chapter 3. Optimisation is the process in which an error function, E, is minimised. The cross-validation set is used to prevent the network from over-training. Over-training occurs when the network is able to represent the data very well but without the ability to generalise. In well-behaved optimisation problems, the value of the error function will have a downward slope in regard to the parameters or weights. The algorithm will terminate when a specified criterion is met, such as a required degree of accuracy. A stopping criterion that is often used in neural network methodology, to prevent over-training, is the value of the error function calculated for the cross-validation set (a data set not used for the estimation of the parameters of the model). If this value shows an upward trend, while the corresponding value for the training set decreases, over-training is indicated. The test

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

set does not form part of the learning process and is not considered in the estimation of the model.

Further in Chapter 3, different optimisation algorithms are discussed. It is emphasised that it is not possible to highlight one specific algorithm because all algorithms have advantages and limitations.

Chapter 4 reviews the *ARMA*(p,q)-process. The different stages of modelling, namely identification, estimation by means of maximum likelihood and evaluation are discussed. This is illustrated by examples where both an *AR*(*2*) model and a neural network model are fitted to two time series, a computer generated *AR(2)* time series and an electricity consumption time series. In Chapter 6, the model used to describe the electricity consumption is extended in order to describe complex seasonal patterns.

Time series models are often used for forecasting. Since neural network models are non-linear, minimum mean squared error forecasts are not of a simple form. In Chapter 5, a procedure is proposed to calculate one- and multi-step forecasts as well as prediction limits for the forecasts based on bootstrap methodology.

In Chapter 6, more complex models are considered to fit electricity consumption. The aim of Chapter 6 is to demonstrate the use of neural networks as a statistical tool for forecasting. Forecasting is done for one to twelve and twentyfour hours ahead. Three of the forecasting procedures, discussed in Chapter 5, are also implemented.

Chapter 7 concludes this work and suggests three extensions to this research, namely the inclusion of bootstrap methods in new software tools, comparative studies and applications.

Chapter 2

Neural Networks

2.1 Introduction

An introduction to neural networks will be given in this chapter. In Section 2.2 the history of Artificial Neural Networks (ANN), the development thereof, the different fields of applications as well as the contributions that have been made between Neural Networks and Statistics are discussed. To conclude the section, the connection between a biological neuron and the neural network is illustrated. The capabilities and properties of neural networks are given in Section 2.3 and 2.4. The construction of a neural network as well as network architectures will be discussed in Section 2.5 and Section 2.6.

2.2 Historical Development

The first time interest was taken in the term *Artificial Neural Networks* was when Mc Culloch and Pitts (1943) introduced simplified neurons which represented biological neurons, and which could perform computational tasks. In 1969 Minsky and Papert (1969) published their book *Perceptrons*, which focussed on the deficiencies of perceptron models. Unfortunately this caused many researchers to leave the field.

Interest in neural networks in the early eighties re-emerged after the publication of several important theoretical results. This renewed interest is clearly visible in the number of societies and journals associated with neural networks. The INNS (International Neural Network Society) is widely known, and ranges from Europe (ENNS) to Japan (JNNS). The field of Electrical and Electronic Engineering is served by the journal series *IEEE*, and the field of Agriculture by the journal *Neural Network Application Agriculture (NNAA)*. In the field of Economics and Finance there are the *Journal of Economics and Complexity* and the *Journal of Computational Intelligence in Finance etc.* Other well known journals include *Neural Networks and Neural Computation*. Several conferences are held regularly, and for example the World Congress of Neural Networks (WCNN'95) took place in Washington in 1995. A large

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

variety of software packages are available in the market today. Neural networks are popular because of the many application fields in which they can be used. Many articles were written in recent years in several interesting fields, for example on classification (Ripley, 1994) and pattern recognition, where applications include the automatic reading of handwriting and speech recognition (Park *et al*, 1991). Another field of application for neural networks is prediction. In the world of finance, Tam and Kiang (1992) showed that the neural network is a promising method of predicting bankruptcy. In the medical world neural networks are used for instance to predict mortality following cardiac surgery (Orr, 1995).

Several articles have been published in which statistical principles are used with neural networks. Cheng and Titterington (1994) pointed out some links between statistics and neural networks and encouraged cross-disciplinary research to improve results. De Jongh and De Wet (1991) introduced neural networks as a powerful tool that is close to statistical methods in solving various problems like pattern recognition, classification and forecasting. The neural network and statistical literature contain many of the same concepts but usually with different terminology (Sarle, 1996). A list of statistical and neural network terminology is given in the Appendix of this chapter.

In the following paragraphs a complete discussion will be given of what a neural network is and how it compares with a biological neural network.

2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are computer algorithms or computer programs developed to model the activity of the brain (Stern, 1996:205). The term "neural network" originates from the attempt by scientists to imitate the ability of the brain to recognise patterns and to classify. This is the reason why it is called "*Artificial Neural Networks*". Scientists prefer to refer to ANN only as NN (neural networks). In this dissertation they will also be referred to only as neural networks.

The brain is a highly complex, non-linear and massively parallel computer. It can perform computations like pattern recognition, perception and motor control many times faster than any digital computer. This is possible because of the existence of

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

neurons. Neurons are simple elements in the brain connected to each other to form a network of neurons. In Figure 2.1 the biological neuron is illustrated.

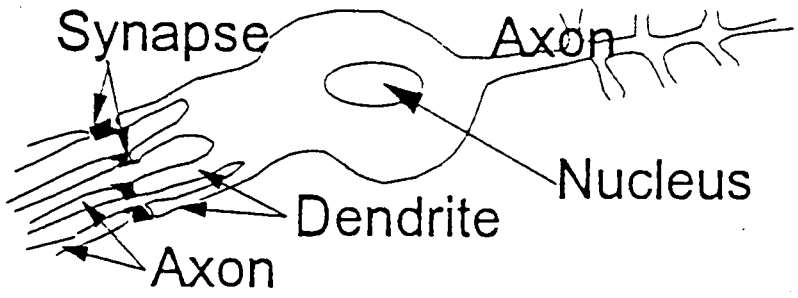

Figure 2.1: The biological neuron (De Jongh and De Wet, 1994:4)

A neuron is activated by another activated neuron(s) and, in turn, activates another neuron(s). *Synapses* are units that mediate the interaction between the neurons. It can either excite or inhibit a receptive neuron. If a neuron receives a signal from a neighbouring neuron, which excites, the neuron's soma potential rises above some threshold value and the neuron "fires". This in turn sends a signal to the next neuron, and so on. New synaptic connections between neurons and the modification of existing synapses constantly take place as the brain learns.

The *axons* are the transmission lines, and the *dendrites* are the receptive zones. The neural network resembles the human brain in two ways, namely that the network has to *learn* to acquire knowledge and then the knowledge is stored in *synaptic weights*, which represent the strength between connected neurons.(Haykin, 1994:2).

2.4 Relevance of Neural Networks in Research

For classification problems, discriminant analysis is used and linear regression analysis is commonly used for prediction. Why would one use neural networks when there are so many other methods available? In this section only a few reasons mentioned by Haykin (1994) are given.

Firstly a neural network develops out of interconnections between neurons which defines a *non-linear* relationship between the neurons whereas the procedure for the above-mentioned techniques are *linear*.

Secondly neural network methodology is a *non-parametric* because the network learns from examples by constructing input-output mapping for the specific problem. Although there are usually many parameters in a neural network model, these parameters are essentially artefacts in the fitting process. The emphasis is therefore not on interpreting their values but on finding the optimal set for the classification or prediction problem. One of the training methods is *supervised learning*. This involves the changing of the synaptic weights by presenting a training set of examples to the network. Every time the weights are changed, an error signal is computed. This error signal is defined as the difference between the actual response of the network and the desired response. The moment the error signal reaches a minimum value, the desired weights for the specific problem have been obtained. Neural networks are therefore *adaptable*. A more detailed discussion on *supervised learning* as a training method is given in Chapter 3.

In classification problems, neural networks are not only designed to give information on which specific pattern to select, but also on the confidence in the decision made. *Evidential response* is thus an important property of neural networks in classification problems.

Neural networks are *fault tolerant* in the sense that, for example if a neuron or its connecting links should be damaged, the neural network will gradually degrade in performance, rather than suddenly.

Another important issue regarding neural networks is that the broad approach is a very computer- intensive method, but with the constant improvement in computer technology it becomes cheaper and faster to do calculations.

Lastly, neural networks are perhaps more user-friendly than other methods because the user only has to differentiate between the independent (input) and dependent (output) variables before selecting a model.

There are other broad classes of neural networks such as Hopfield associative networks, Linear networks, Probabilistic neural networks and Radial Basis Function neural networks. A feed forward neural network, as used in this dissertation, is a name used for a wide variety of mathematical models used to define the connection between a number of explanatory variables and one or more dependent variables. In Section 2.5 it is shown that the architecture of the model depicts the mathematical outlay of the neural network.

2.5 The Model of a Neural Network

As mentioned in Section 2.2, the neurons are the elements that are connected to each other to form the network. In Figure 2.2 the non-linear model of a neuron is illustrated. The different components of a non-linear model are given.

Let $x_1, x_2, \ldots x_p$ be a set of independent or explanatory variables and let y_k be the non-linear function

$$y_k = \varphi(\sum w_{kj} x_j - \theta_k) \tag{2.5-1}$$

The output or result of the neuron model is y_k .

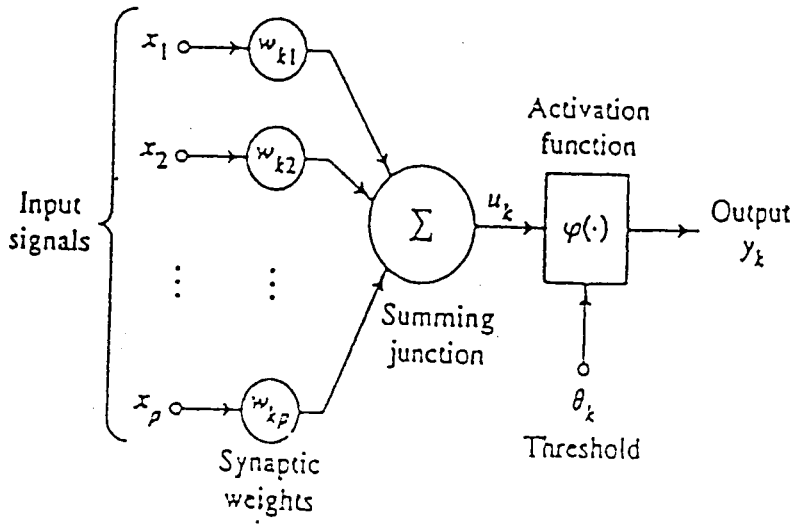

Figure 2.2: The non-linear model of a neuron (Haykin, 1994:8)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The x_j's are the *input signals*. They are each connected to a weight or strength of its own. A signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers. If the weight is positive, the associated synapse is excitatory. If negative, the synapse is inhibitory. The threshold function, θ_k, is a bias or constant parameter added as an extra input with value in most cases chosen as one.
The non-linear regression model can be described as

$$y_k = \varphi(x_j, w_{kj}) + \varepsilon_k \qquad (2.5\text{-}2)$$

where y_k is the output, w_{kj}, $j = 1,2, \ldots p$ are the parameters or weights associated with each input value x_j, $j = 1,2,\ldots p$ and ε_k is the residual term.

The *linear combiner* described as Σ sums the input signal x_j, weighed by the respective synapses of the neuron.

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \qquad (2.5\text{-}3)$$

where u_k the linear combined output.

The *activation function*, φ, transforms the amplitude of the output of a neuron. Normally the amplitude will lie in the interval (0.0;1.0) or (-1.0;1.0), depending on the type of activation function. Three types of activation functions, namely the *threshold function*, the *piecewise linear function*, and the *sigmoidal function* are discussed.

Let v_k be the internal activity level of the neuron; that is:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j - \theta_k \qquad (2.5\text{-}4)$$

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The *threshold function* is defined as

$$\varphi(v) = \begin{cases} 1 \text{ if } v > 0 \\ 0 \text{ if } v < 0 \end{cases} \qquad (2.5\text{-}5)$$

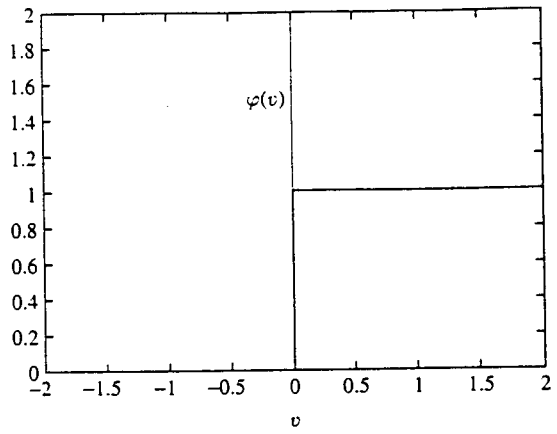

Figure 2.3: Threshold function (Haykin, 1994:11)

The *piecewise linear function* is given by

$$\varphi(v) = \begin{cases} 1 \text{ if } v \geq 0.5 \\ v \text{ if } -0.5 < v < 0.5 \\ 0 \text{ if } v \leq -0.5 \end{cases} \qquad (2.5\text{-}6)$$

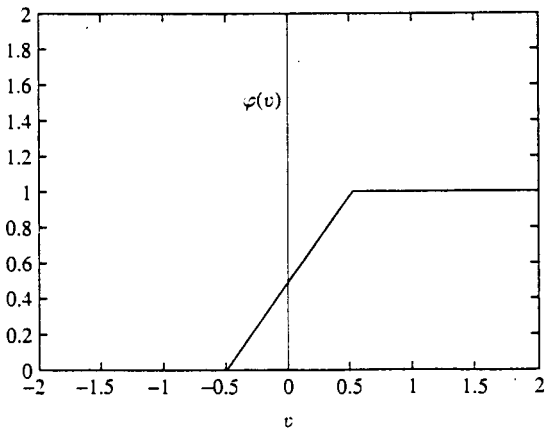

Figure 2.4: The piecewise linear function (Haykin, 1994:11)

The *sigmoidal function* is the most popular activation function. The fact that it is a differentiable function is especially important in the training of neural networks. The

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

reason for this will be discussed later. An example of a sigmoidal function is the *logistic function.*

$$\varphi(v) = \frac{1}{1+\exp(-av)} \quad , \tag{2.5-7}$$

where a is a slope parameter.

Note that the activation functions range from 0 to 1. In Figure 2.5 the sigmoidal function is illustrated for different values of a.

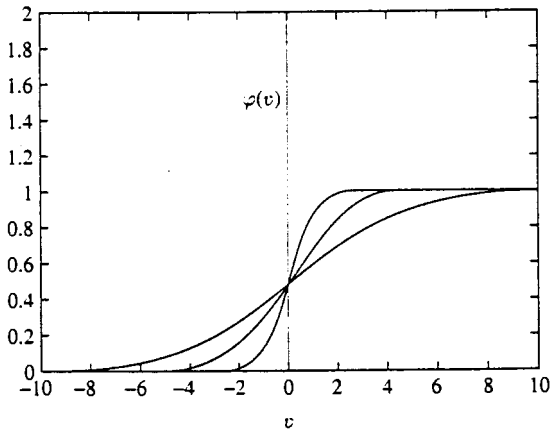

Figure 2.5: The sigmoidal function (Haykin, 1994:11)

The *threshold* has the effect of lowering the net input to the activation function. If the opposite effect is needed a *bias* term can be implemented. The bias and threshold function are, therefore numerically the same but with different signs.

$$y_k = \varphi(u_k - \theta_k) \tag{2.5-8}$$

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

where y_k is the output of the signal, $\varphi(.)$ is the activation function and θ_k is the threshold. The use of the threshold θ_k has the effect of applying an *affine transformation* to the output u_k, of the linear combiner. See Figure 2.6.

$$v_k = u_k - \theta_k \tag{2.5-9}$$

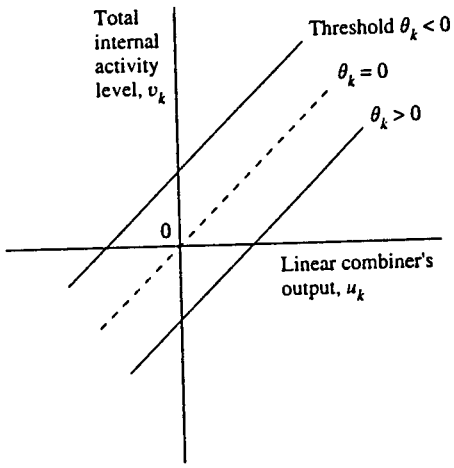

Figure 2.6: Affine transformation produced by the presence of a threshold (Haykin, 1994:9)

Whether the threshold θ_k is positive or not, the relationship between the activation potential v_k of neuron k and the linear combiner output u_k stays constant.

A neural network is an interconnection of neurons. Neurons can be connected in many different ways to define different neural network architectures. In Section 2.5 the architecture of the neural network is illustrated along with the different classes of neural network architectures.

2.6 Architecture of a Neural Network

Neural networks are mostly illustrated as block diagrams in which the various elements of the model are described. There are four different classes of network

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

architectures, namely single-layer *feedforward* networks, multi-layer *feedforward* networks, *recurrent* networks and lattices structures. Only the first three will be discussed.

2.6.1 Single-Layer Feedforward Networks

When a network is organised in a form of layers, it is referred to as a *layered* network. A linear regression model is equivalent to a feedforward neural network in its simplest form with an input and an output layer and a linear activation function. In Figure 2.7, a feedforward network with a single layer of neurons is illustrated. The most simple feedforward neural network has an input and output layer.

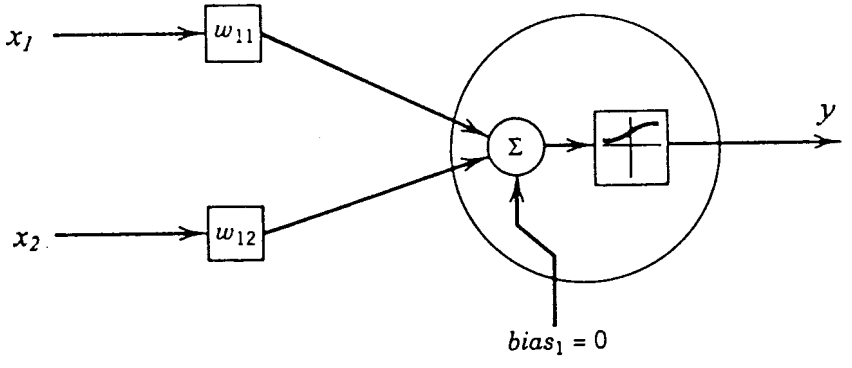

Figure 2.7:Feedforward network with a single layer of neurons (Schalkhoff, 1992:250)

2.6.2 Multi-Layer Feedforward Networks

This network consists of an *input* layer, one or more *hidden* layers and an *output* layer. The computation nodes of the hidden layer are called the *hidden neurons* or *units*. This type of network defines a more complicated model. The relationship between the inputs and outputs can be non-linear functions of non-linear functions, so

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

that the degree of non-linearity increases with each additional layer. In Figure 2.8 a fully connected feedforward network with one hidden layer is illustrated. In a fully connected feedforward network, every input neuron in the input layer is connected to every neuron in the following hidden layer, as well as every neuron in the last hidden layer connected with every neuron in the output layer (Schalkhoff, 1992:236-258).

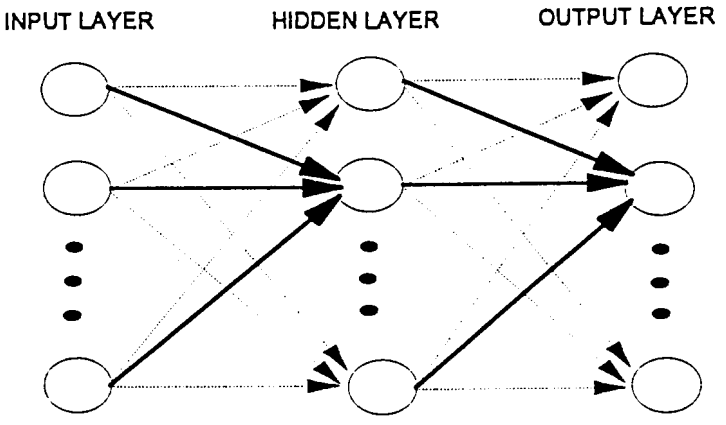

Figure 2.8: Fully connected feedforward network with one hidden layer and output layer(De Jongh and De Wet, 1994:5)

There are many fields of application for the multi-layered feedforward network, like classification, pattern recognition and forecasting. The following example illustrates a time series prediction problem.

Example 2.1

Let y_t, y_{t-1}, y_{t-2} ... be a stationary time series and let y_{t+1} be the value to be predicted. Then a set of d such values y_{t-d+1}, ... y_t can be selected from the time series to be the inputs to a feedforward network, and the next value y_{t+1}, used as the target for the output. In Figure 2.9 a feedforward neural network with one hidden layer is illustrated (Bishop, 1996:303).

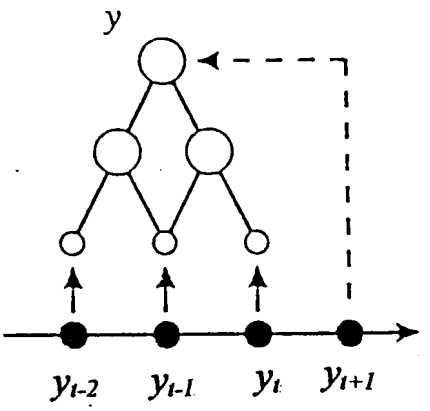

Figure 2.9: A one step multi layered feedforward network (Bishop, 1996:303)

2.6.3 Recurrent Networks

A recurrent network differs from a feedforward network in the sense that it has at least one *feedback* loop. If feedback exists in a neural network, it can be shown that its output depends on all previous input values. Let Y_1, Y_2, ... Y_N be a time series. Suppose that a recurrent network architecture is used to define a model for the time series and that the observation at time t can be written as

$$Y_t = f(Y_{t-1}, \hat{Y}_{t-1}; \underline{w}) + \varepsilon_t \quad , t = 2,3,...N,$$

$$\text{and } \hat{Y}_{t-1} = f(Y_{t-2}, \hat{Y}_{t-2}; \hat{w})$$

In this example of a recurrent network with one feedback loop Y_t therefore depends on Y_{t-1} and the previous output value, $\hat{Y}_{t-1}$, which is characteristic of a recurrent neural network model. By recursive substitution it can be shown that Y_t can be written as a function of Y_{t-1}, Y_{t-2}, ... :

$$\begin{aligned} Y_t &= f(Y_{t-1}, \hat{Y}_{t-1}, \underline{\hat{w}}) + \varepsilon_t \\ &= f(Y_{t-1}, f(Y_{t-2}, \hat{Y}_{t-2}, \hat{w}), \underline{\hat{w}}) + \varepsilon_t \end{aligned}$$

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

If Y_t is generated by a moving average process (see par 4.2), it can be expressed as a linear function of an infinite number of previous time series observations: Y_{t-1}, Y_{t-2}, ... (see 4.2.7). A recurrent network will therefore be suitable in the case of any process with moving average terms.

Basic ModelGen™ 1.0 of Crusader systems, the software package that was used in this study, offers both feed forward and Elman networks, which are recurrent. In Figure 2.10 The Elman-net is illustrated.

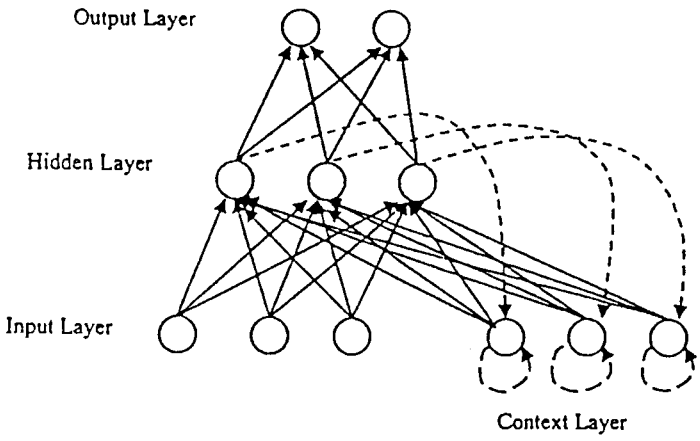

Figure 2.10: An Elman Recurrent network (Basic ModelGen™1.0, 1997:53-54)

In an Elman network one or more of the hidden units from the previous time step are used as an input at the next time step. This is useful when the output depends on the history of the inputs, which is the case in this study. O'Brien (1997) uses a feedforward neural network and an Elman recurrent neural network to extract knowledge of a physical system (O'Brien, 1997).

A neural network can thus be compared with a non-linear model which defines a connection between certain input or explanatory or independent variables and output or dependent variables. Neural networks are used for different applications such as classification and prediction. In order to predict or classify, the parameters in the case of the non-linear model and known in neural network literature as weights, first have to be

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

estimated. The estimation of the weights is referred to as "training" the network. Chapter 3 investigates the various methods of training a network.

Appendix

Terminology

Below is a list of neural network terms and the corresponding statistical terms.(Sarle, 1996: 1-5)

Neural network term	Statistical term
Architecture	Model
Training, Learning, Adaptation	Estimation, Model fitting, Optimisation
Classification	Discriminant analysis
Mapping, Function approximation	Regression
Supervised learning	Regression, Discriminant analysis
Unsupervised learning, Self-organization	Principal components, Cluster analysis, Data reduction
Training set	Sample, Construction sample
Test set, Validation set	Hold-out sample
Input	Independent variables, Predictors, Regressors, Explanatory variables, Carriers
Output	Predicted values
Forward propagation	Prediction
Training values, Target values	Dependent variables, Responses, Observed values
Training pair	Observation containing both inputs and target values

Neural network term	**Statistical term**
Errors	Residuals
Noise	Error term
Generalisation	Interpolation, Extrapolation, Prediction
Prediction	Forecasting
Squashing function	Bounded function with infinite domain
Error bars	Confidence intervals
Weights, Synaptic weights	(Regression) coefficients, Parameter estimates
Bias	Intercept
The difference between the expected value of a statistic and the corresponding true value	Bias
Backpropagation	Computation of derivatives for a multi-layer perceptron and various algorithms based thereon
Least mean squares (LMS)	Ordinary least squares (OLS)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 3

Learning in Neural Networks

3.1 Introduction

One of the most important properties of a neural network is its ability to learn. In neural network methodology, the data set is divided into three different sets, namely a *training set*, a *cross-validation set*, and a *test set*. The training set is used for training the network with the various available learning (optimisation) algorithms. These are discussed in this chapter. Section 3.2 defines the error function associated with neural network learning. In Section 3.3 and Section 3.4 methods based on first and second order Taylor expansions are reviewed, including the *gradient descent method*, *Newton methods*, *conjugate gradient* and *quasi-Newton methods*. The powerful superlinear methods namely *Levenberg-Marquardt algorithm* and *Snyman's leap-frog method*, which involve combinations of first and second order Taylor expansions, are also discussed. The procedure of cross-validation, which forms part of the learning process, is described in Section 3.5. Different algorithms will perform best on different problems. It is therefore not possible to highlight one specific algorithm. The advantages and limitations of different algorithms in respect of all training problems are discussed.

3.2 The Error Function

Optimisation is the process in which an error function E is minimised. There are many optimal error functions, of which the sum of squares error function is the most widely used. The error is a function of the *weights* in the network as well as the training data. For a multi-layered perceptron (MLP) (discussed in Chapter 2), the derivatives of an error function with respect to the weights can be obtained efficiently by using *backpropagation*. The gradient information is of central importance in the use of algorithms for network training. There are four main stationary points at which the local gradient of an error function can possibly vanish. Figure 3.1 illustrates this point by showing the value of an error function E against a single weight w. Point A is called a local minimum because it is not the global minimum value in the error space. Point B is a local maximum, while point C is called a "saddle point". Point C is a

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

region where the error function can be flat and some algorithms tend to get stuck on this flat surface for long periods. This behaviour is also found with local minima and can lead to premature termination of the optimisation algorithm. The desired error minimum is point D, the global minimum, because the value of the error function is minimal at this point.

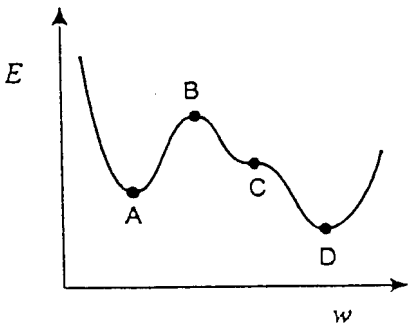

Figure 3.1: An error function with four stationary points (Bishop, 1996:255).

To illustrate basic terminology, the simple case where the error function depends on only two weights, w_1 and w_2, is considered. The problem is to find values for w_1 and w_2 where the error function reaches a global minimum. Figure 3.2 is a geometrical representation of an error function $E(\underline{w})$ as a surface lying above the weight space. Point A represents a local minimum while point B represents the global minimum of the error function. If C is any point in the error surface, then the local gradient of the error function is ∇E, where ∇E is the gradient of the error function with respect to the weights. The gradient in Figure 3.2 will thus be the two-dimensional vector of partial derivatives with respect to the weights:

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}\right) \quad . \qquad (3.2\text{-}1)$$

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

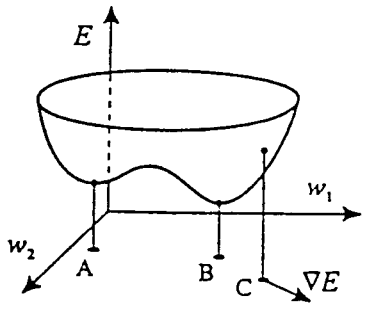

Figure 3.2: Geometrical illustration of the error function $E(w_1,w_2)$ (Bishop, 1996:254).

Neural networks usually have many weight parameters, which may lead to extended training times. As a result, effective algorithms are necessary to find a suitable local minimum of the error function in the shortest time possible. The parameters or weights are not identifiable because more than one set of parameters can give the same error function value. For instance, a two layered neural network with M hidden neurons exhibits a symmetry factor of $M!2^M$ (Bishop, 1996:256) equivalent points which generate the same network mapping and therefore give the same value for the error function. It is clear that the error function therefore does not have a unique global minimum. As a result, the specific values of the weights are not important.

When the error surface is locally convex, the error function can be written as a Taylor series expansion in the vicinity of a local minimum. Optimisation algorithms can be classified according to their relation with the terms in the Taylor expansion. Four categories can be distinguished. Firstly, methods based on zero order Taylor expansions are *simplex and random walk*. These methods are well treated in different texts, and will not be of interest in this work (Fletcher, 1987). A second category includes methods based on first order Taylor expansions such as *gradient descent*. A third category which can involve combinations of either first and of second order Taylor expansions are superlinear methods such as the *Levenberg-Marquardt, leap-frog, quasi- Newton* and *conjugate gradient* methods. Finally, the fourth category

14643261
b14358062

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

includes methods based purely on second order Taylor expansions such as the *Newton methods*.

Because of the non-linearity of the error function, it is impossible to find global solutions to the error. To overcome this problem, algorithms that search through the weight space are used; formulated in mathematical terms as follows:

$$\underline{w}(n+1) = \underline{w}(n) + \Delta\underline{w}(n) \qquad (3.2\text{-}2)$$

where $\underline{w}$ is a weight vector (chosen at random) in a specific update direction, n the iteration step and $\Delta\underline{w}(n)$ the adjustment of the weight vector. Different algorithms involve different choices for $\Delta\underline{w}(n)$. Algorithms such as *conjugate gradient* and *quasi-Newton* are formulated not to increase the error function as the weights change. Non-linear optimisation algorithms cannot guarantee global minimum and therefor the disadvantage of these methods is that the error function can get stuck in a local minimum, which, in turn, leads to premature termination.

3.3 Taylor Series Approximations

The process of learning or estimation involves the process of optimising the error function by finding the vector of weights which minimises the error function. By obtaining a local quadratic approximation by using a Taylor series expansion, the error function can be minimised in a convex region around a local minimum.

Let E be a function of more than one argument, $E(\underline{w}) = f(w_1, w_2, \ldots, w_n)$ (Hamilton, 1994:735-738). Furthermore, let E: $\mathfrak{R}^n \rightarrow \mathfrak{R}$ with continuous second order derivatives. A first order Taylor series expansion of $E(\underline{w})$ around any point $\underline{w}_1$ is given by

$$E(\underline{w}) = E(\underline{w}_1) + \frac{\partial E}{\partial \underline{w}}\Big|_{\underline{w}=\underline{w}_1}^{T} (\underline{w}_2 - \underline{w}_1) + R_1(\underline{w}_1, \underline{w}_2). \qquad (3.3\text{-}1)$$

$\frac{\partial E^T}{\partial \underline{w}} = \nabla E^T$ is a $(1 \times n)$ vector (the transpose of the gradient) and $R_1(.)$ is a remainder term of second and higher derivatives.

If $E(.)$ has continuous second order derivatives, a second order Taylor series expansion of $E(\underline{w})$ around $\underline{w}_1$ is given by

$$E(\underline{w}_2) = E(\underline{w}_1) + \frac{\partial E^T}{\partial \underline{w}}\Big|_{\underline{w}_1}(\underline{w}_2 - \underline{w}_1) + \frac{1}{2}(\underline{w}_2 - \underline{w}_1)^T \frac{\partial^2 E}{\partial \underline{w}^2}\Big|_{\underline{w}=\underline{w}_1}(\underline{w}_2 - \underline{w}_1) + R_2(\underline{w}_1, \underline{w}_2) \qquad (3.3\text{-}2)$$

where $R_2(.)$ is a remainder term of third and higher derivatives.

Consider a second order Taylor series approximation of the multi-variable error function $E(\underline{w})$: $R^m \rightarrow R^1$ around some point $\underline{w}_1$ in the weight space, which is to be minimised. When $E(\underline{w})$ is twice differentiable, the second order Taylor series approximation of $E(\underline{w})$ around $\underline{w}_1$ is

$$E(\underline{w}_2) \cong E(\underline{w}_1) - \underline{b}(\underline{w}_2 - \underline{w}_1) + \frac{1}{2}(\underline{w}_2 - \underline{w}_1)^T H(\underline{w}_2 - \underline{w}_1) \qquad (3.3\text{-}3)$$

where $\underline{b}$ is the gradient of E evaluated at $\underline{w}_1$

$$\underline{b} = \frac{\partial E(\underline{w})}{\partial \underline{w}}\Big|_{\underline{w}_1} \qquad (3.3\text{-}4)$$

and H the Hessian matrix of second order derivatives defined by

$$(H)_{ij} = \frac{\partial^2 E}{\partial w_i w_j}\Big|_{\underline{w}_1} \qquad (3.3\text{-}5)$$

The first derivative of the approximated error function (3.3-3) with respect to $\underline{w}$ is given by

$$\frac{\partial E(\underline{w})}{\partial \underline{w}} = \underline{b} + H(\underline{w}_2 - \underline{w}_1) \qquad (3.3\text{-}6)$$

The gradient is set equal to zero at a stationary point, say $\underline{w}_2 = \underline{w}^*$ and solved for $\underline{w}$, then

$$H\underline{w}^* = H\underline{w}_1 - \underline{b}, \quad \underline{w}_2 = \underline{w}^* \qquad (3.3\text{-}7)$$

$$\underline{w}^* = \underline{w}_1 - H^{-1}\underline{b}. \qquad (3.3\text{-}8)$$

This forms the basis of many of the learning algorithms because, for points $\underline{w}$ that are close to $\underline{w}^*$, these above expressions will give acceptable approximations to the error function and its gradient.

3.4 Training

3.4.1 The Method of Gradient Descent

The method of gradient descent, also known as steepest descent, makes use of the first order Taylor expansion of the error function, E($\underline{w}$) as given by (3.3-1). As mentioned in Section 3.2, different algorithms involve different choices for $\Delta\underline{w}(n)$, the weight vector update. An initial guess for the weight vector, denoted by $\underline{w}_0$, is required. The weight vector is updated in such a way that with each iteration step n the direction of the movement is towards the negative gradient of the error function $\frac{\partial E(\underline{w}_n)}{\partial \underline{w}_n}$, evaluated at $\underline{w}_n$. This is a discretisation of a first-order differential equation in $\underline{w}$ so that the trajectory in weight space leads to a stationary point $\underline{w}^*$. Thus for the error function

$E(\underline{w}_{n+1}) = E(\underline{w}_n) + \underline{b}^T(\underline{w}_{n+1} - \underline{w}_n)$, with

$\frac{\partial E(\underline{w}_n)^T}{\partial \underline{w}_n} = \underline{b}$, the weight vector update is

$$\Delta \underline{w}(n) = -\eta \frac{\partial E(\underline{w}_n)}{\partial \underline{w}_n} \quad . \qquad (3.4.1\text{-}1)$$

The value of η, the *learning* or *weight update parameter*, must satisfy the criterion:

$$0 < \eta < \frac{2}{\lambda_{\max}},$$

where $\lambda_{\max}$ is the largest eigenvalue of $H(\underline{w})$ (Haykin, 1994:49). This leads to guaranteed convergence, although in practice η is determined empirically or by rule of thumb.

The algorithm that implements the gradient descent learning strategy for feed forward neural networks is known as *back propagation*. It can be described in two steps. The first step is a forward propagation of the input pattern from the input to the output layer of the network. The second step is a back propagation of the error vector from the output layer to the input layer of the network (Berthold *et al*, 1999: 228-229).

3.4.2 Superlinear Methods

Superlinear methods involve either first or second order Taylor expansions, such as *Levenberg-Marquardt*, *leap-frog*, and *quasi-Newton* methods. Consider (3.3-8)

$$\underline{w}^* = \underline{w}_1 - H^{-1}\underline{b},$$

where the *Levenberg-Marquardt methods* replace H^{-1} with $\eta(I)$ where η is a step size parameter and I the unity matrix. When η is equal to 1, $\eta(I)\underline{b}$ dominates and the method resembles to gradient descent and when η becomes small the method becomes second order.

It is important to mention that if residuals, which is the difference between the model output and the desired output, are not small, it should be better to use general quasi-

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

methods or hybrids thereof (see later in this section). The Levenberg-Marquardt algorithm (Bishop, 1996:290-292) minimises the error function while at the same time it tries to keep the step size small to ensure that the linear approximation remains valid.

The *leap-frog* optimisation method of Snyman (1982:449-462) is a reliable and robust algorithm which will generally compute acceptable minima without being overly sensitive with respect to difficult environmental conditions. The method is based on the motion of a particle of unit mass in an N-dimensional conservative force field where the total energy of the particle is conserved. In the case of neural networks the force field is the weight space. The energy of the particle consists of two components, namely potential energy and kinetic energy from the fact that a force acts on a unity mass particle. The force accelerates the particle to a point where it has maximum kinetic energy and minimum potential energy. When this happens the particle stops because the force is zero. The force acting on the particle is the negative of the gradient of the potential energy (Snyman, 1983). The *leap-frog* method is based on Euler relations for dynamic systems. The updating equation (cf 3.2-2) for this method is

$$\underline{w}(n+1) = \underline{w}(n) + \underline{v}(n)\Delta t(n) \quad \text{with}$$

$$\underline{v}(n)\Delta t(n) = \Delta \underline{w}(n) = \underline{w}(n+1) - \underline{w}(n) \quad \text{and with} \qquad (3.4.2\text{-}1)$$

$$\underline{v}(n+1) = \underline{v}(n) + \underline{a}(n+1)\Delta t(n)$$

where $\Delta \underline{w}(n) = \underline{v}(n)\Delta t(n)$ is the step size in the weight space, $\Delta t(n)$ the time step, and $\underline{a}(n)$ the acceleration of the particle at step n. (3.4.2-1) relates the trajectory of the particle in the weight space to the particle velocity $\underline{v}$ and acceleration $\underline{a}$.

Quasi-Newton methods use the local gradient of the error function, but instead of calculating the Hessian directly and then evaluating its inverse, an approximation A of the inverse Hessian is built up over a number of steps. The minimum of the quadratic error function is typically reached after W steps (where W is the number of weights and biases in a network) as is the same with the method of conjugate gradient. A

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

sequence of matrices is generated to represent increasingly accurate approximations to the inverse Hessian H^{-1}. This is accomplished by the use of only first order derivatives of the error function (by using backpropagation). By starting from a positive definite matrix such as the unity matrix, the problem of non-positive definite Hessian matrices is eliminated. The update procedures are such that the approximation to the inverse Hessian H^{-1} is guaranteed to remain positive definite although the condition number of H^{-1} may become inconveniently large. Two commonly used update procedures are the *Broyden-Fletcher-Goldfarb-Shanno (BFGS)* and the *Davidson-Fletcher-Powell (DFP)* (Bishop, 1996:288) with BFGS the method of choice.

For the method of *gradient descent*, the direction vector is the negative of the gradient vector. Small steps are taken to reach the minimum of the error function, which can be a time consuming effort. The method of *conjugate gradients* guarantees the error function is minimised within W search steps, without calculating the Hessian matrix. This property is an improvement on the *gradient descent* method because the gradient descent needs many steps to minimise a simple quadratic error function. For *conjugate gradient* the choice for $\Delta\underline{w}(n)$ (the weight update in (3.2-2)) is

$$\Delta\underline{w}(n) = \eta\,\underline{p}(n), \tag{3.4.2-2}$$

where $\underline{p}(n)$ denotes the search direction vector at iteration n of the algorithm and η is the learning rate parameter. The initial direction vector, $\underline{p}(0)$, is set equal to the negative gradient vector,

$$\underline{b}_0 = \frac{\partial E(\underline{w})}{\partial \underline{w}}\bigg|_{\underline{w}=\underline{w}^*} \tag{3.4.2-3}$$

and can then be written as

$$\underline{p}(0) = -\underline{b}_0 \tag{3.4.2-4}$$

Each successive direction vector is then calculated as a linear combination of the current gradient vector and the previous direction vector:

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$\underline{p}(n+1) = -\underline{b}_{n+1} + \beta(n)\underline{p}(n) \qquad (3.4.2\text{-}5)$$

where $\beta(n)$ is a time varying parameter that is determined in terms of the gradient vectors $\underline{b}_n$ and $\underline{b}_{n+1}$. The *Fletcher-Reeves* formula and the *Polak-Ribière* formula can be used to determine $\beta(n)$ (Bishop, 1996:280-281). This method was applied to small Boolean learning problems and it was shown that backpropagation learning based on *conjugate gradient* required fewer iterations than the standard backpropagation algorithm, but is more complex to compute (Kramer and Sangiovanni-Vincentelli, 1989 and Johansson *et al*, 1992). There are known problems with conjugate gradient when the dimensionality of the network becomes large. This is due to cummulative mis-adjustment on the search direction update formula given in (3.4.2-5).

3.4.3 The Newton Methods

Newton methods may take only a few steps to converge, but are relatively slow because the *Newton methods* make explicit use of the full Hessian matrix, H, which has to be calculated. These can lead to computational expense. However, these methods are elegant and applicable and must be treated in this text.

Consider the second order Taylor expansion given in (3.3-3) and assume that $E(\underline{w})$ is twice differentiable. Then from (3.3-8), where the weight vector $\underline{w}$ corresponding to the minimum of the error function satisfies

$$\underline{w} = \underline{w}^* - H^{-1}\underline{b}. \qquad (3.4.3\text{-}1)$$

The vector $-H^{-1}\underline{b}$ is known as the Newton direction or step (Bishop, 1996:287). Since the quadratic approximation used to determine (3.3-3) is not exact, the weight update equation has to be applied iteratively and H re-evaluated at each new search point. The Newton method has significant disadvantages. For non-linear networks, the Hessian matrix take NW^2 steps to compute, where W is the number of weights in the network and N the number of patterns in the training set. It also has to be inverted,

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

requiring $\frac{1}{2}NW^3$ steps. The Hessian has to be positive definite to ensure descent in the Newton direction.

3.5 Cross-validation as Part of the Learning Process

Cross validation is the process whereby a trained network's performance is measured on unseen data records. The data is divided into three different subsets, namely the training set, cross validation set and the test set. The cross validation set is used during the training process in order to stop training when the network starts to overtrain. With each iteration, the root mean square error (RMSE) of both the training and the cross validation sets are calculated and compared. The RMSE of the training set will typically decrease, and after a number of iterations, stabilise. The RMSE as a measure of performance can be determined by calculating the sum of squares of the error function

$$E(\underline{w}) = \frac{1}{2}\sum_{j=1}^{N}(y_j - f(\underline{x}_j, \underline{w}))^2 \qquad (3.5\text{-}1)$$

where y_j is the target output, $f(\underline{x}_j; \underline{w})$ a function of the input vector $\underline{x}_j$ and the weight vector $\underline{w}$ and N is the number of training patterns. The MSE (mean squared error) is then

$$MSE = \frac{E(\underline{w})}{N} \qquad (3.5\text{-}2)$$

Overtraining occurs when the network model fits the training set well but fails to generalise on samples of the same population. The problem is more accute in situations where the number of observations is small in comparison with the number of weights. To overcome the problem of overtraining, the neural network model fitted to the training set is also (in a sense) fitted to the cross validation set. A measure of fit such as the root of the mean squared error (RMSE) is calculated for both data sets. If the RMSE of the cross validation set increases after a number of iterations while the

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

corresponding value decreases for the training set this is a sign of overtraining and unacceptable generalisation results.

The remaining data is used to test the network's performance after training. This will be discussed in Chapter 4 by means of an example, when the test set is used to test the performance of the model after training.

3.6 Conclusions

In this chapter different training or learning algorithms are given. It is emphasised that there is no ideal learning algorithm and that discretion must be used in selecting the correct method for a specific problem. In the next chapter time series analysis with neural network models and Box-Jenkins autoregressive moving average (*ARMA*) models are discussed.

Chapter 4

ARMA Processes and Neural Network Models

4.1 Introduction

The aim of this chapter is to illustrate the identification, estimation and evaluation of neural network models and autoregressive moving average (*ARMA*) models for a stationary time series. Neural network models have been used in time series prediction (Le Baron and Wiegend, 1996). Autoregressive moving average models (*ARMA*) are well-known for purposes of time series analysis (Ansuj *et al*, 1996). These models define linear relationships between a time series observation at time t, the dependent variable, and a set of time series observations that occurred prior to time t. Any linear model can be expressed as a simple feed forward neural network model with only linear activation functions. This is referred to as a regressor. The versatility of a neural network lies in the fact that it is used to model non-linear relationships between input and output variables. This can result in very complicated models with a large number of parameters, where the parameters have no physical meaning. In model building there is always a trade-off between models with a large number of parameters that fit very well, but with poor generalisation abilities, and models with fewer parameters that do not fit all that well but produce better forecasts because of better generalisation abilities. In Section 4.2 to Section 4.5 the *ARMA(p,q)*-process is defined, maximum likelihood estimation and the evaluation of the results are discussed. It is shown that these procedures can also be applied as part of the neural network methodology.

4.2 The *ARMA(p,q)*-process

Suppose that Y_t, t =... *-1, 0, 1* ... is an equally spaced, weakly stationary or covariance stationary, time series. A well-known class of linear models for the analysis of time series in the time domain belongs to an autoregressive moving average (*ARMA*) class of the form (4.2-1) (Hamilton, 1994:59-61),

$$Y_t = \theta_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} \qquad (4.2\text{-}1)$$

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

where $\{\varepsilon_t\}$ is a sequence of uncorrelated variables, also referred to as a white noise process, with conditions

$$E(\varepsilon_t) = 0 \text{ and}$$
$$E(\varepsilon_t, \varepsilon_\tau) = \begin{cases} \sigma_a^2 & \text{for } t = \tau \\ 0 & \text{otherwise} \end{cases} \qquad (4.2\text{-}2)$$

and θ_0, ϕ_1, ... ϕ_p, θ_1, ... θ_q are unknown constants or parameters. The model (4.2-1) is an *ARMA(p,q)* model or Box-Jenkins model.

The *ARMA(p,q)* model (4.2-1) can be expressed as

$$(1-\phi_1 B-\phi_2 B^2-\ldots-\phi_p B^p)Y_t = \theta_0 + (1+\theta_1 B+\theta_2 B^2+\ldots+\theta_q B^q)\varepsilon_t \qquad (4.2\text{-}3)$$

where B is the backshift operator, that is

$$BY_t = Y_{t-1}$$

$$\Phi(B) = 1-\phi_1 B-\phi_2 B^2-\ldots-\phi_p B^p \qquad (4.2\text{-}4)$$

$$\Theta(B) = 1+\theta_1 B+\theta_2 B^2+\ldots+\theta_q B^q$$

The process is stationary if the roots of the equation $\phi(B) = 0$ all lie within the unit circle. The *ARMA(p,q)* model can be expressed as a moving average (*MA*) model,

$$Y_t = \mu + \psi(B)\varepsilon_t \qquad (4.2\text{-}5)$$

where

$$\psi(B) = \frac{(1+\theta_1 B+\theta_2 B^2+\ldots+\theta_q B^q)}{(1-\phi_1 B-\phi_2 B^2-\ldots-\phi_p B^p)} = \frac{\theta(B)}{\phi(B)}$$

and

$$\mu = \frac{\theta_0}{(1-\phi_1 B-\phi_2 B^2-\ldots-\phi_p B^p)}$$

or, as an autoregressive (*AR*) model,

$$\Pi(B)Y_t = \mu^* + \varepsilon_t$$

$$= \psi^{-1}(B)$$

$$\text{with } \Pi(B) = \frac{\phi(B)}{\underline{\theta}(B)}, \quad \mu^* = \frac{\theta_0}{\underline{\theta}(B)}. \tag{4.2-6}$$

Let $\Pi(B) = 1 - \pi_1 B - \pi_2 B^2 - \ldots$ .The equation (4.2-6) can be written as

$$(1 - \pi_1 B - \pi_2 B^2 - \ldots)Y_t = \mu^* + \varepsilon_t$$

$$Y_t = \mu^* + \varepsilon_t + \pi_1 Y_{t-1} + \pi_2 Y_{t-2} + \ldots \tag{4.2-7}$$

Any *ARMA* model can therefore be written as an *AR* or *MA* model (Cryer, 1986:73-74).

Box and Jenkins (Box, Jenkins and Reinsel, 1994) developed a general framework for time series modelling. They suggested a model-building strategy where model identification, estimation and diagnostic checking are done iteratively to select the best possible model for a given series.

4.3 Identification

The *ARMA* model is based on linear relationships between successive observations as measured by the autocorrelation function. Plots of the sample autocorrelation function (ACF) and the sample partial autocorrelation function (PACF) are compared to the corresponding population functions to identify the *ARMA* model by selecting the values of p and q, the autoregressive and moving average order, of the model. (Cryer, 1986:111).

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The autocovariance function is defined as

$$\gamma_k = Cov(Y_t, Y_{t-k}) \qquad \text{for } t \text{ and } k = 0, \pm 1, \pm 2, \ldots \tag{4.3-1}$$

$$= \phi_1 \gamma_{k-1} + \ldots + \phi_p \gamma_{k-1} + \gamma_{Y\varepsilon}(k) - \theta_1 \gamma_{Y\varepsilon}(k-1) - \ldots - \theta_q \gamma_{Y\varepsilon}(k-q) \tag{4.3-2}$$

where

$$\gamma_{Y_\varepsilon} = Cov(Y_{t-k}, \varepsilon_t) = 0 \text{ if } k > 0$$
$$\neq 0 \text{ if } k \leq 0 \tag{4.3-3}$$

The autocorrelation function (ACF) is defined by

$$\rho_k = \frac{\gamma_k}{\gamma_0} \tag{4.3-4}$$

with $\gamma_0 = var(Y_t)$ for $t = 1,2,3\ldots$

The partial autocorrelation function (PACF) is

$$\phi_{kk} = Corr(Y_t, Y_{t-k} \mid Y_{t-1}, Y_{t-2}, \ldots, Y_{t-k+1}) \tag{4.3-5}$$

ϕ_{kk} is the correlation coefficient in the bivariate distribution of Y_t, Y_{t-k} conditional on Y_{t-1}, Y_{t-2}, ... , Y_{t-k+1} (Cryer, 1986:106).

The ACF of an *AR(p)* process shows an exponential decay or a damped sine wave, while its PACF is zero for lags greater than p.

A pure moving average model of order q has nonzero autocorrelations only on the first q lags and its partial autocorrelations are not zero after q lags.

Two time series are used to illustrate the concepts of identification, estimation and evaluation of a model. Time Series 1 consists of 150 terms generated from an *AR*(2)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

model with ϕ_1 =1.3, ϕ_2 = -0.5, and zero mean, and $\varepsilon_t \sim N(0,\sigma_a^2)$. Time Series 2 consists of 336 data points of electricity consumption, measured hourly, for two weeks.

Example 4.1

As mentioned in Section 4.3, the sample autocorrelation function and sample partial autocorrelation function can be used to identify a possible model that can be used to describe a time series.

A time plot can be used to identify a trend, seasonal patterns, outliers, discontinuities, etc. A time plot of hundred observations of Series 1 is given in Figure 4.1.

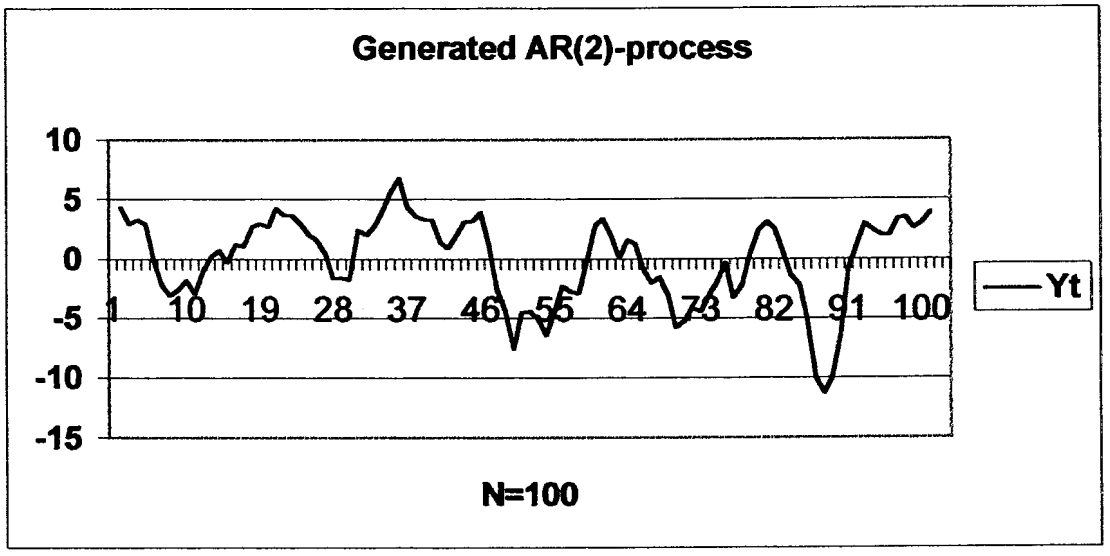

Figure 4.1: A generated *AR*(2)-process (SAS System for Windows v6.12)

Figure 4.2 is the sample autocorrelation function. It has the appearance of a damped sine wave which is typical of an *AR* model while the sample partial autocorrelation illustrated in Figure 4.3 gives a clear indication of an *AR*(2) model. The first two partial autocorrelations differ significantly from zero while the rest do not differ from zero on the 5% significance level.

Lag	Covariance	Correlation	-1 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9 1	Std
0	7.670817	1.00000	|********************|	0
1	6.013996	0.78401	| . |***************. |	0.100000
2	3.712388	0.48396	| . |**********. |	0.149310
3	2.166189	0.28239	| . |******. |	0.164249
4	1.322617	0.17242	| . |***. |	0.169035
5	0.765067	0.09974	| . |**. |	0.170784
6	0.505147	0.06585	| . |*. |	0.171366
7	0.830336	0.10825	| . |**. |	0.171619
8	1.042072	0.13585	| . |***. |	0.172300
9	0.744163	0.09701	| . |**. |	0.173368
10	0.323893	0.04222	| . |*. |	0.173910
11	-0.124980	-0.01629	| . | . |	0.174012
12	-0.368546	-0.04805	| . *| . |	0.174028
13	-0.454332	-0.05923	| . *| . |	0.174160
14	-0.431367	-0.05623	| . *| . |	0.174362
15	-0.391593	-0.05105	| . *| . |	0.174543
16	-0.338712	-0.04416	| . *| . |	0.174692
17	-0.359287	-0.04684	| . *| . |	0.174804
18	-0.769711	-0.10034	| . **| . |	0.174929
19	-1.156028	-0.15070	| . ***| . |	0.175504
20	-1.668117	-0.21746	| . ****| . |	0.176793
21	-2.225459	-0.29012	| .******| . |	0.179448
22	-2.764961	-0.36045	| ********| . |	0.184079
23	-2.763432	-0.36025	| .********| . |	0.191007
24	-1.983370	-0.25856	| . *****| . |	0.197685

'.' marks two standard errors

Figure 4.2: Sample autocorrelation function for an AR(2) with ϕ_1=1.3 and ϕ_2 = -0.5 (SAS System for Windows v6.12)

Lag	Correlation	-1 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9 1
1	0.88347	| . |****************** |
2	-0.42102	| ********| . |
3	0.00224	| . | . |
4	0.00211	| . | . |
5	-0.00117	| . | . |
6	0.02936	| . |* . |
7	0.02602	| . |* . |
8	0.08493	| . |** . |
9	0.06300	| . |* . |
10	0.11536	| . |** . |
11	-0.06218	| . *| . |
12	-0.00949	| . | . |
13	-0.08332	| . **| . |
14	0.19943	| . |**** |
15	-0.13970	| .***| . |
16	-0.04186	| . *| . |
17	-0.00209	| . | . |
18	-0.14519	| .***| . |
19	-0.07380	| . *| . |
20	0.02051	| . | . |
21	0.04710	| . |* . |
22	-0.06309	| . *| . |
23	-0.15938	| .***| . |
24	-0.00652	| . | . |

Autocorrelation Check for White Noise

Figure 4.3: Sample partial autocorrelation function for an AR(2) with ϕ_1=1.3 and ϕ_2 = -0.5 (SAS System for Windows v6.12)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Figure 4.4 is a time plot of Series 2.

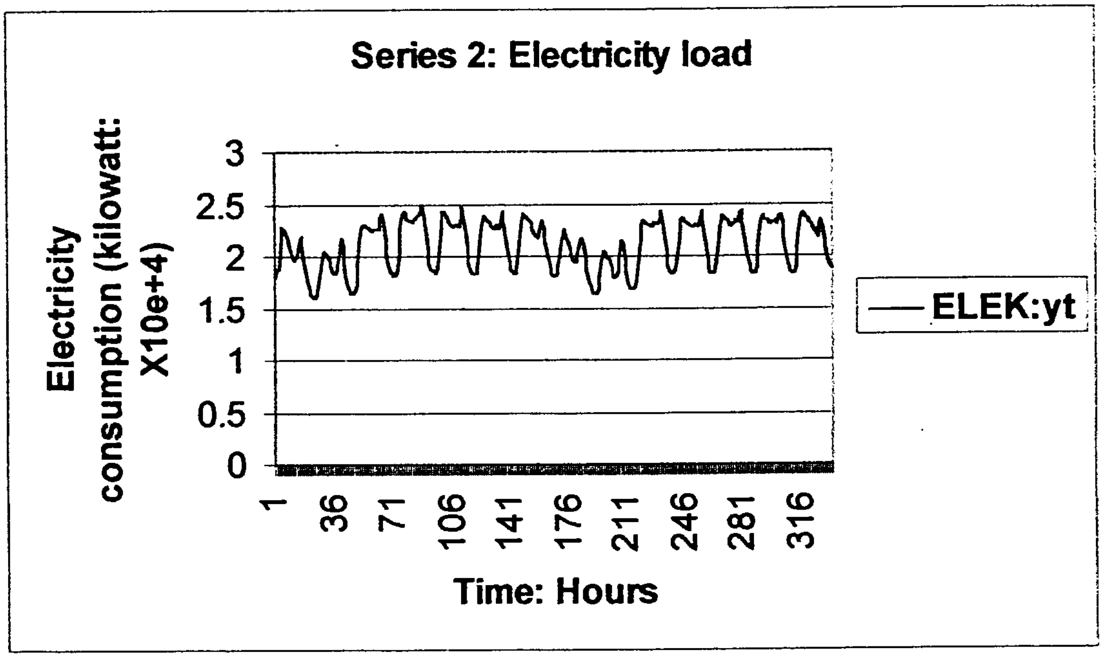

Figure 4.4: The electricity consumption for two weeks

There are explicit daily, 12-hourly and weekly seasonal patterns in the electricity consumption, due to consumer behaviour. The autocorrelation function depicted in Figure 4.5 shows evidence of a sine wave, which may suggest an $AR(2)$ component. The sample autocorrelations from lags 20 to 24 are quite high due to the 24-hour seasonal pattern.

```
Lag Covariance Correlation -1 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9 1
  0    5842606     1.00000  |                    |********************|
  1    5379454     0.92073  |                 .  |******************  |
  2    4337118     0.74233  |               .    |***************     |
  3    3086162     0.52822  |              .     |***********         |
  4    1901803     0.32551  |             .      |*******             |
  5     917772     0.15708  |             .      |***   .             |
  6     194390     0.03327  |             .      |*     .             |
  7    -238615    -0.04084  |             .     *|      .             |
  8    -361234    -0.06183  |             .     *|      .             |
  9    -197386    -0.03378  |             .     *|      .             |
 10     146418     0.02506  |             .      |*     .             |
 11     456639     0.07816  |             .      |**    .             |
 12     511863     0.08761  |             .      |**    .             |
 13     243946     0.04175  |             .      |*     .             |
 14    -219729    -0.03761  |             .     *|      .             |
 15    -631504    -0.10809  |             .    **|      .             |
 16    -783852    -0.13416  |             .   ***|      .             |
 17    -635189    -0.10872  |             .    **|      .             |
 18    -208851    -0.03575  |             .     *|      .             |
 19     465597     0.07969  |             .      |**    .             |
 20    1324200     0.22665  |             .      |***** .             |
 21    2276192     0.38959  |             .      |********            |
 22    3199092     0.54755  |             .      |***********         |
 23    3893269     0.66636  |            .       |*************       |
 24    4086444     0.69942  |            .       |**************      |
                            "." marks two standard errors
```

Figure 4.5: The autocorrelation function for the sample of electricity consumption

The partial autocorrelation function's first two lags are significant and the remaining lags are significantly reduced which indicates an $AR(2)$- process. This is illustrated in Figure 4.5.

Lag	Correlation	-1 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9 1
1	0.89902	|. |********** |
2	-0.40365	|********. | . |
3	-0.08511	|. **| . |
4	-0.12721	|. ***| . |
5	-0.22907	|*****. | . |
6	-0.17068	|. ***| . |
7	0.15599	|. |***. |
8	0.13347	|. |***. |
9	0.04207	|. |* . |
10	0.02022	|. | . |
11	-0.04473	|. *| . |
12	-0.12335	|. **| . |
13	-0.09437	|. **| . |
14	0.08936	|. |** . |
15	-0.19785	|****. | . |
16	-0.09166	|. **| . |
17	-0.00793	|. | . |
18	-0.13995	|. ***| . |
19	0.00804	|. | . |
20	-0.10087	|. **| . |
21	-0.02190	|. | . |
22	-0.06457	|. *| . |
23	-0.00557	|. | . |
24	0.03435	|. |* . |

Figure 4.6: The partial autocorrelation function for the sample of electricity consumption.

A spectral analysis of the data can be used to identify the most prominent cycles in the data. A periodogram of the data is given in Figure 4.7.

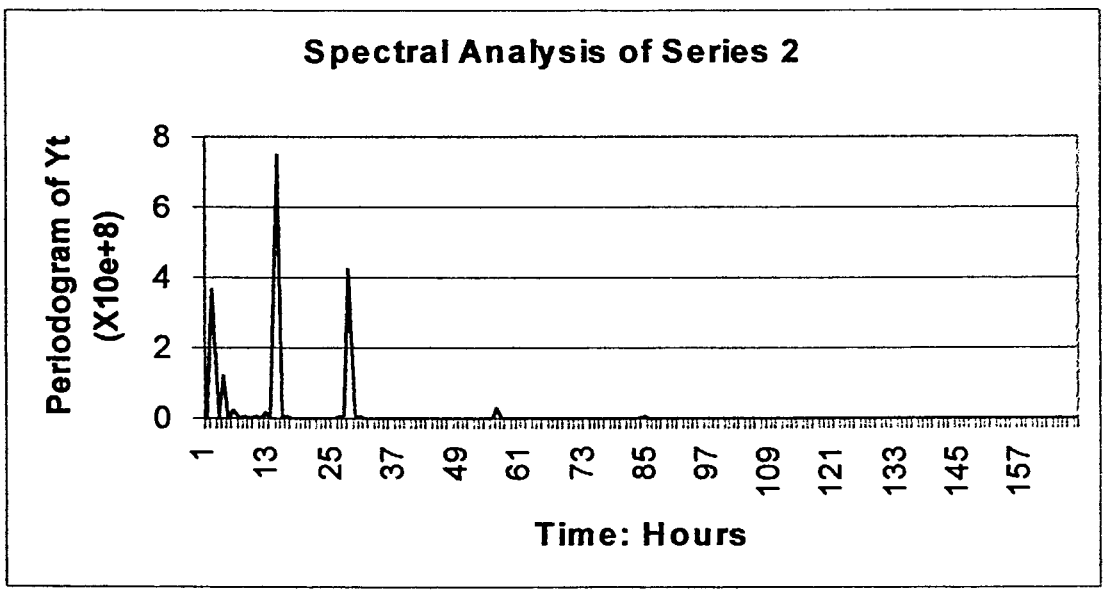

Figure 4.7: Spectral analysis of electricity data

The graph shows peaks at 12-hour, 24-hour (the most) and 48-hour periods. A large percentage of the total variation in the data can therefore be ascribed to the identified cycles, and the information can be used to define input variables for a neural network model.

Through the method of spectral analysis, the series of observations Y_t can be described as a weighed sum of periodic functions of the form $\cos(\omega t)$, where ω denotes a particular frequency (Hamilton, 1994:152). For Series 2 with dominant periods of 24 hours and 12 hours, as seen in Figure 4.7, the frequencies can be calculated as follows:

$$\text{Period} = 24 = \frac{2\pi}{\varpi_1} \quad \text{therefore } \varpi_1 = 0.2618$$

$$\text{Period} = 12 = \frac{2\pi}{\varpi_2} \quad \text{therefore } \varpi_2 = 0.5236.$$

The proposed model for electricity consumption is

$$Y_t = \mu + \alpha_1 \cos(\varpi_1(t-1)) + \delta_1 \sin(\varpi_1(t-1)) + \alpha_2 \cos(\varpi_2(t-1)) + \delta_2 \sin(\varpi_2(t-1)) + Y_{t-1} + Y_{t-2} + \varepsilon_t. \qquad (4.3\text{-}6)$$

The input variables for the neural network model will be:

C1= $\cos(\varpi_1(t-1))$, C2=$\cos(\varpi_2(t-1))$, S1=$\sin(\varpi_1(t-1))$, S2=$\sin(\varpi_2(t-1))$, Y_{t-1}, and Y_{t-2}.

Different numbers of nodes in the hidden layer were tried and it was decided to use no more than two nodes in the hidden layer. The added complexity introduced by adding nodes in the hidden layer did not result in a significant improvement of the model fit.

4.4 Estimation

4.4.1 Maximum Likelihood Estimation (MLE)

Let $Y_1, Y_2, \ldots, Y_N$ denote a stationary time series, and suppose that the observation at time t can be described by the model

$$Y_t = f(Y_{t-1}, Y_{t-2}, \ldots Y_{t-p}, \underline{w}) + \varepsilon_t, \qquad (4.4.1\text{-}1)$$

where $\underline{w}$ is the parameter vector or weight vector and the error terms $\ldots \varepsilon_{t-1}, \varepsilon_t, \varepsilon_{t+1} \ldots$ are assumed to be white noise. If f is a linear function, the above model is an $AR(p)$ model.

The parameters of an *ARMA* model can be estimated by the method of maximum likelihood. There are various other methods, for example, the *conditional least square method (CLS)* and the *unconditional least squares method (ULS)*. In the case of an *AR* model, if the error terms are assumed to be independent normal, it can be shown that the maximum likelihood method is equivalent to the least squares method. For purposes of this dissertation only the method of maximum likelihood will be discussed.

Let $\underline{\gamma} \equiv (\theta_0, \phi_1, \phi_2, \ldots, \phi_p)$

be the vector of parameters for an $AR(p)$ model and let $(Y_1, Y_2, \ldots Y_N)$ be a stationary time series. The value of γ that maximises the joint probability or likelihood function

$$f_{Y_N, Y_{N-1}, \ldots, Y_1}(y_N, y_{N-1}, \ldots, y_1, \underline{\gamma}) \qquad (4.4.1\text{-}2)$$

is the maximum likelihood estimate. The likelihood function for an $AR(p)$-process, conditions on both y's and ε's.

If the initial values of $\underline{y}_0 \equiv (y_0, y_{-1}, \ldots, y_{-p+1})'$ and $\underline{\varepsilon}_0 \equiv (\varepsilon_0, \varepsilon_{-1}, \ldots, \varepsilon_{-q+1})'$ are given, the sequence $\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N\}$ can be calculated from $\{y_1, y_2, \ldots, y_N\}$ by iterating on

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$\varepsilon_t = y_t - \theta_0 - \phi_1 y_{t-1} - \ldots - \phi_p y_{t-p}.$$

For $t = 1,2,\ldots,N$ the conditional log likelihood is then

$$\begin{aligned} L(\underline{\gamma}) &= \log f_{Y_N, Y_{N-1}, \ldots, Y_1 | \underline{Y}_0, \underline{\varepsilon}_0}(y_N, y_{N-1}, \ldots, y_1 | \underline{y}_0, \underline{\varepsilon}_0; \underline{\gamma}) \\ &= -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma^2) - \sum_{t=1}^{N} \frac{\varepsilon_t^2}{2\sigma^2}. \end{aligned} \qquad (4.4.1\text{-}3)$$

with the option of setting the initial y's and ε's equal to their expected values (Hamilton, 1994:132). The error function used for maximum likelihood estimation can be written as

$$E(\underline{\gamma}) = -\ln L(\underline{\gamma}) \qquad (4.4.1\text{-}4)$$

4.4.2 Estimation when using Neural networks

In neural network terminology, estimation of the parameters of the model takes place during the training period, during which the network learns with generalisation through certain learning algorithms. Weights connected to each input value (parameters) are constantly updated until the error function, $E(\underline{w})$, which is often the sum of squared errors (see Chapter 2), reaches its global minimum.

Example 4.2

An $AR(2)$ model was fitted, by using the SAS System for Windows v6.12, to Series 1 by the method of maximum likelihood. The estimates, approximate standard errors and corresponding t-ratios are given in Table 4.1.

Table 4.1: The maximum likelihood estimation results for Series 1.

Parameter	Estimate	Approx. Std Error	T-Ratio	Lag
MU	-0.67277	0.89122	-0.75	0
AR1,1	1.26298	0.09195	13.74	1
AR1,2	-0.42242	0.0937	-4.51	2

The white noise variance estimate is $S^2 = 3.515$.

$$\hat{S}^2 = \frac{1}{N-p}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

For large sample sizes the t-ratio may be used to test the significance of the estimates with respect to the null hypothesis that the corresponding parameter is zero. The t-value for the *mean* is small, not rejecting the null hypothesis. This is expected, since the time series was generated with a zero mean. The parameter is redundant and should be excluded from the model. The t-values for the two estimates are large, indicating that ϕ_1 and ϕ_2 should not be excluded from the model. In Figure 4.8 the time plots of the actual and the fitted series is given.

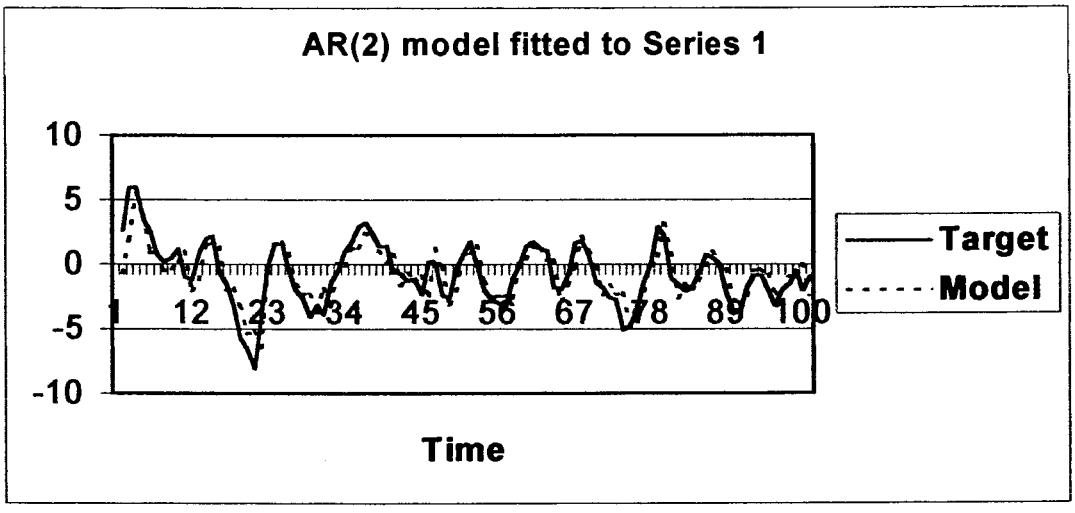

Figure 4.8: *AR*(2) model fitted to Series 1

A neural network is fitted to the electricity consumption time series. The program package used for this purpose is Basic ModelGen™1.0 from Crusader Systems.

Estimation of a neural network model takes place during training by means of the training algorithm. For the training set 70% of the data is used, 20% for the cross-validation set and the rest is used for testing the model. The model (4.3-6) has six input variables, as mentioned in Example 4.1, as well as an intercept term included in both the input layer and the hidden layer. The neural network model is non-linear with two neurons in the hidden layer. The sigmoidal function (2.5-7) is used as activation function. The sum of squared error function (3.5-1), which is a function of the weights in the network, is minimised. This is done efficiently by using backpropagation. As mentioned in Chapter 3, the cross-validation set is also used during training. The root mean square error (RMSE) of both the training set and the cross-validation set decrease with each iteration. The RMSE of the training set will usually decrease more than the RMSE of the cross-validation set because the network is trained on the data of the training set. If too many parameters are included in the model relative to the number of data points in the training set, over-training can occur. An increase in the RMSE of the cross-validation set gives an indication that the network is starting to over-train and training stops immediately. In Figure 4.9 a plot of the RMSE of the training set and the cross-validation set is illustrated.

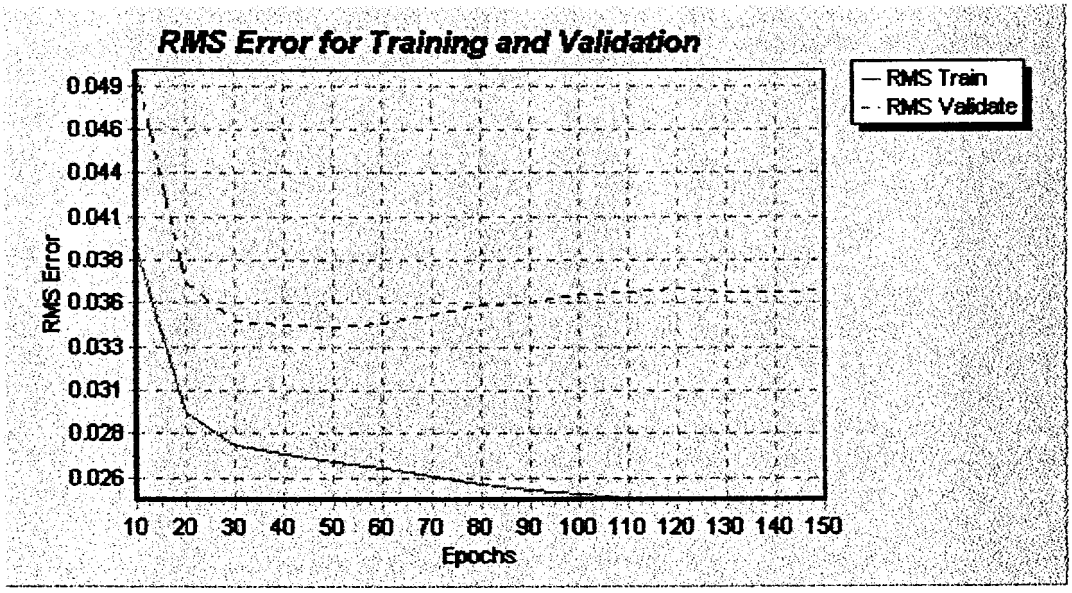

Figure 4.9: A plot of the RMSE of the training set and cross-validation set

The output of the model is a function of the weights but the weights themselves do not have any significance value. When the vector of weights, which minimises the gradient of the error function, has been determined through a learning algorithm, training stops. The output of the model containing the six input variables (and one

hidden layer with two neurons) as indicated in Section 4.3 fitted to Series 2 is given in Figure 4.10.

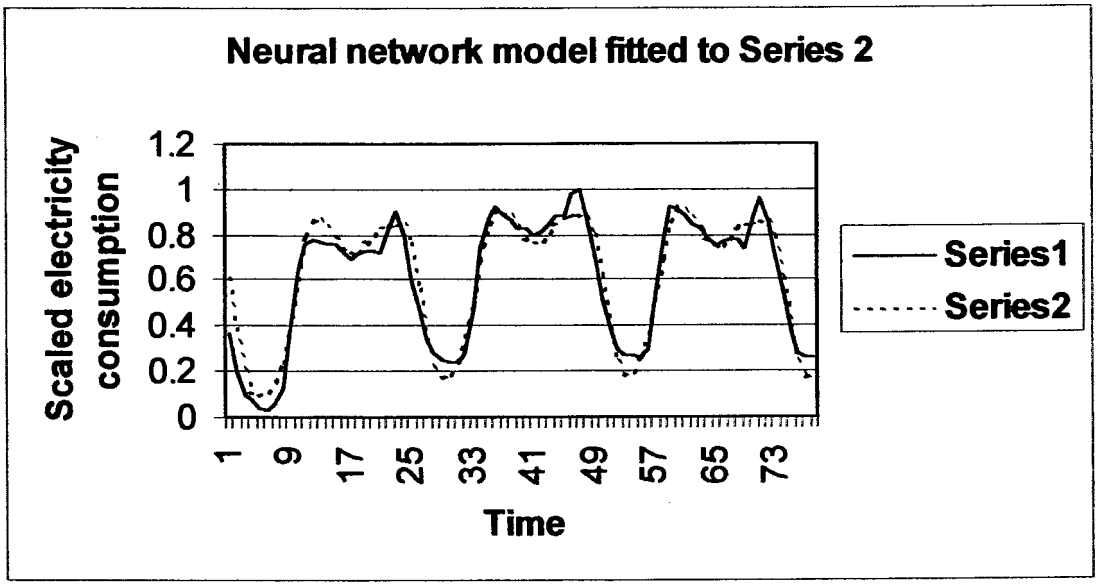

Figure 4.10: Electricity consumption modelled with feedforward neural network

Although the data set is quite small, the model fits the data reasonably well, considering Figure 4.10. A rule of thumb that is often used and accepted by practitioners is that the number of observations should be at least a factor of ten times the number of weights ($N=10W$).

In this section, the estimation of linear and neural network models is discussed. The evaluation of the estimated model is discussed in the next section.

4.5 Evaluation

After a model has been identified and the parameters estimated, the model must be evaluated. Evaluation procedures are the same for both the ARIMA and neural network models. If the model fits the data well, then the residuals should almost have the properties of uncorrelated, identically distributed, random variables with zero mean and fixed standard deviation (Cryer, 1986:147). If the estimated value of Y_t is given by

$$\hat{Y}_t = f(Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p}, \hat{\underline{w}}), \quad t = 1,2\ldots,N \tag{4.4-1}$$

then the residual at time t is defined as

$$\hat{\varepsilon}_t = Y_t - \hat{Y}_t, \quad t = 1,2\ldots,N \tag{4.4-2}$$

The residuals of a fitted model are useful indicators of any inadequacies in the specification of the model or violations of underlying assumptions.

Examination of various plots of the residuals, is an indispensable step in the evaluation process of any model (Box and Jenkins, 1994:289). If a plot of the residuals exhibits a trend over time, it is an indication of a trend in the data that is not adequately modelled.

A histogram of standardised residuals should correspond with a symmetrical normal curve if the model fits the data well. Any outliers will imply significant differences between the $\hat{Y}_t$ and the corresponding observed value, Y_t that should be investigated since the model fits the data poorly at those points.

The sample autocorrelation function of the residuals, $\hat{r}_k$, can be observed to check for the independence of the residuals in the model. Usually the sample autocorrelations are approximately uncorrelated and normally distributed with mean zero and variance $1/n$. A χ^2 (Chi-squared) test is used to test whether residuals are correlated (Cryer, 1986:153).

Different programme packages evaluate neural network models differently. A comprehensive residual analysis is offered by the BasicModelGen^{TM}1.0 Crusader systems. The mean square error (MSE) in (3.5-2) and the root of the mean square error (RMSE) are determined for every model and the model with the smallest of these values should be selected. As mentioned, the residuals play an important role in the evaluation process. A fast Fourier transform of the model residuals is calculated. If the spectrum of the residuals is constant, white noise is implied and it can be concluded that the model fits the data relatively well.

Other methods of evaluation are a sensitivity analysis performed to establish the most influential input variable of the model, and output analysis where the relationship between an output and specific input is shown. The t-test statistic is used to determine whether a model's output differs from the desired output. Together with this test the mean and the standard deviation of the model's output are calculated. The quality of the model can be observed by viewing a graphical representation of the model's output versus the desired output. Another useful measure of evaluation is the correlation (R) between the target output and the actual model output. In the following example the evaluation of the models fitted to the $AR(2)$ and the electricity demand time series is illustrated.

Example 4.3

The residuals for the estimated $AR(2)$ model in Example 4.2 were analysed. A time plot and other residual plots revealed no deviations from the model and assumptions. A χ^2 (Chi-squared) test for independence of the residuals was performed and the null hypothesis was not rejected. $AR(1)$, $ARMA(2,1)$ and $AR(3)$ models were also fitted, but the $AR(2)$ model fit was superior as measured by Akaike's information criterion (Cryer, 1986:122) and the significance of the estimated parameters. This is an expected result, since the time series is generated by an $AR(2)$-process.

The evaluation procedure points out some inconsistencies with respect of the neural network model fitted to the electricity data, which can be expected because the data is seasonal and the model selected to fit the series perhaps does not include all the variables necessary to fit the data. This problem is examined in Chapter 6.

The observed spectrum of the residuals of the neural network model has some peaks which indicate that there is some seasonal information present in the residuals.

The sensitivity analysis, illustrated in Figure 4.11, shows that the variable Y_{t-1} has the greatest influence on the model output which is quite reasonable because it represents the electricity consumption during the previous hour.

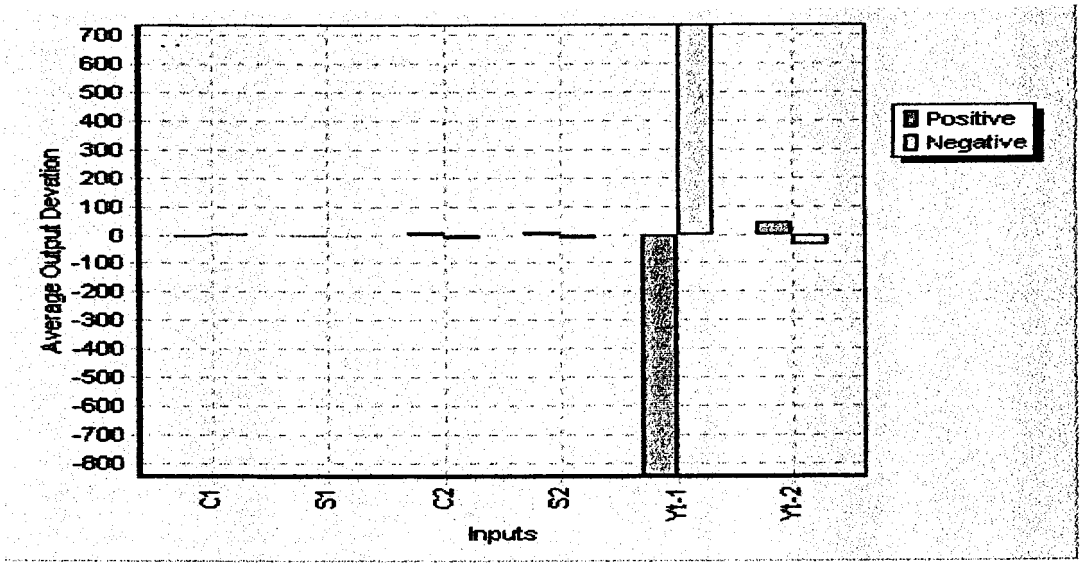

Figure 4.11: The sensitivity analysis of the neural network model

The t-test is used to determine how significantly a model's output differs from the desired or target output. The time plot of the model output and the desired output for the test set are given in Figure 4.12.

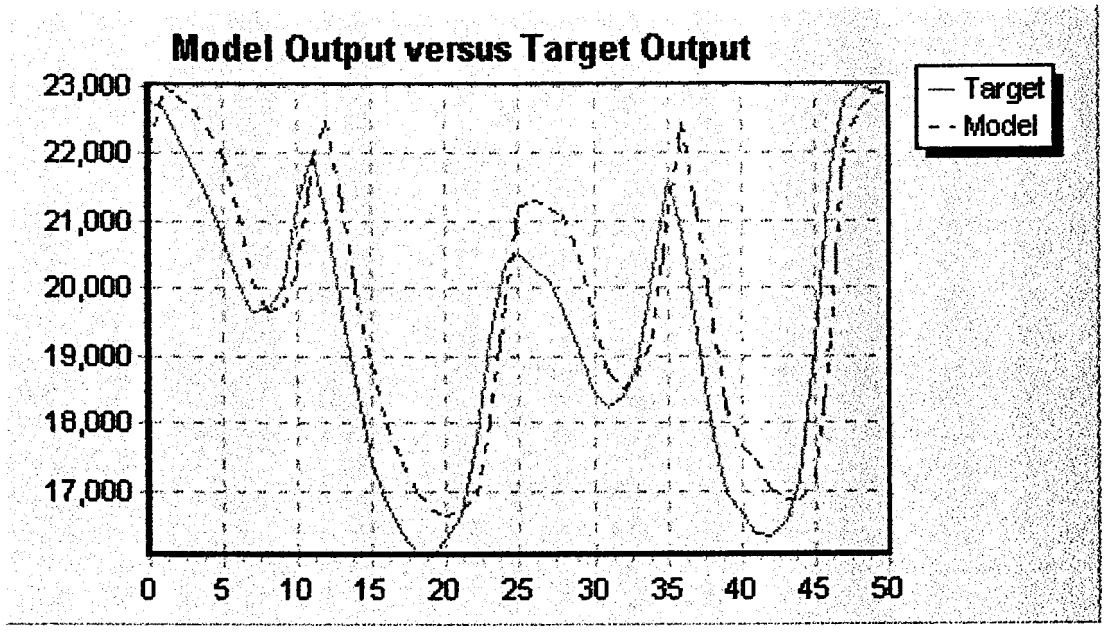

Figure 4.12: Output of the target and the model for the test set

The correlation between the target output and model output is 0.897. This value differs significantly from 0. Table 4.2 provides this information together with the mean and standard deviation of the model and the desired outputs for the test set, as well as a 95% confidence interval for *R*.

Table 4.2: The correlation coefficient for the test set as determined by the neural network model

	Mean	Standard Deviation	R	R^2	95% Confidence Limits for R
Backprop net	19861.1	2065.443	0.89722	0.80501	[0.826-0.940]
Desired	19461.3	2144.093			

Although the evaluation of the model for the electricity consumption may indicate some inadequacies, one can always improve the model by trial and error. In Chapter 6 an electricity load forecasting will be done on one year's data.

In this chapter the process of identification, estimation and evaluation was discussed by means of examples. When the process of evaluation has been completed, and the model gives a satisfactory description of the data, the model can be used for forecasting or classification. In Chapter 5 forecasts and prediction intervals are derived and calculated for neural network models.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 5

Forecasting a Time Series using Neural Networks

5.1 Introduction

In this chapter a neural network model is used to forecast a time series. It is shown that bootstrap methods can be used to calculate standard errors and prediction limits for the forecasts.

In Section 5.2 one- and two-step minimum mean square error (MSE) forecasts are given for both a linear autoregressive model and a non-linear neural network model. In case of the non-linear model, a different approach has to be considered. A few alternative theoretical approaches to obtain forecasts for non-linear time series models are discussed in Section 5.3 and the bootstrap technique is selected for the purpose of obtaining prediction intervals. Section 5.4 shows how bootstrap methodology can be used to construct prediction intervals.

One- and multi-step predictions, together with their standard errors and 95% prediction intervals are calculated for the simulated $AR(2)$ series.

5.2 The Forecasting Model

Consider the $ARMA$ model (4.2-1) for a stationary time series $\{Y_t\}$ with $t = 1,2, \ldots, N$. The linear $AR(1)$ model is given by

$$Y_t = \phi_1 Y_{t-1} + \varepsilon_t \qquad (5.2\text{-}1)$$

where $\{\varepsilon_t\}$ is a white noise process that satisfies the conditions (4.2-2). Suppose that the white noise variables are identically distributed.

The problem is to predict the value of Y_{N+1} where

$$Y_{N+1} = \phi_1 Y_N + \varepsilon_{N+1}, \tag{5.2-2}$$

from $Y_1, Y_2, \ldots, Y_N$.

The minimum mean square error (*MSE*) linear predictor for one step is the conditional expectation of Y_{N+1}, given $Y_1, Y_2, \ldots Y_N$:

$$\begin{aligned} Y_N(1) &= E(Y_{N+1}|Y_1, Y_2, \ldots, Y_N) \\ &= E(\phi_1 Y_N + \varepsilon_{N+1}|Y_1, Y_2, \ldots, Y_N) \\ &= \phi_1 Y_N \end{aligned} \tag{5.2-3}$$

By using (5.2-1) for the second step prediction, Y_{N+2}, the *MSE* linear predictor can be written in terms of the observed data Y_N:

$$\begin{aligned} Y_N(2) &= E(Y_{N+2}|Y_1, Y_2, \ldots, Y_N) \\ &= \phi_1 E(Y_{N+1}|Y_1, Y_2, \ldots, Y_N) + 0 \\ &= \phi_1 Y_N(1) \\ &= \phi_1^2 Y_N \end{aligned} \tag{5.2-4}$$

In case of a non-linear model such as the neural network,

$$Y_t = f(Y_{t-1}, \alpha) + \varepsilon_t \tag{5.2-5}$$

the one-step minimum *MSE* linear predictor is given by

$$Y_N(1) = E(Y_{N+1}|Y_1, Y_2, \ldots, Y_N) = f(Y_N, \alpha) \tag{5.2-6}$$

Two or more step forecasts are expectations of non-linear functions, for example

$$\begin{aligned} Y_N(2) &= E[f(Y_{N+1}, \alpha) + \varepsilon_{N+2}|Y_1, Y_2, \ldots, Y_N] \\ &= E\{f[f(Y_N, \alpha) + \varepsilon_{N+1}], \alpha]|Y_1, Y_2, \ldots, Y_N\} \\ &= E\{f[Y_N(1) + \varepsilon_{N+1}], \alpha]|Y_1, Y_2, \ldots, Y_N\} \end{aligned} \tag{5.2-7}$$

Alternative approaches have been proposed to calculate the forecasts; such methods are discussed in the following paragraphs.

5.3 Non-Linear Forecasting

Five techniques for forecasting have been discussed by Brown and Mariano (1989) and summarised by Lin and Granger (1994). Some of the methods are naïve in the requirement of assumptions regarding the distribution of the white noise terms. The methods vary in respect of ease of implementation and computing sensitivity.

5.3.1 The Naïve Method

This technique is widely used and easy to implement but due to the fact that the forecast is biased, it is not satisfactory. If one considers the non-linear model for time series forecasting given in (5.2-5), the naïve technique states that the two-step forecast of Y_{N+2} is given by

$$Y_N(2) = f(Y_N(1), \hat{\alpha}) \qquad (5.3.1\text{-}1)$$

so that ε_{N+2} is put equal to its mean value of zero at each time and $Y_N(1)$ is substituted for Y_{N+1} in the model equation. The estimator $\hat{\alpha}$ is obtained by minimising the sum of squared error terms or by maximising the likelihood function of the observations. Usually the expected value of the function is not equal to the function of the expected value and therefore the forecast will be biased. Even if the functional form $f(\,)$ is known, the bias will not go to zero if the sample size N is large.

5.3.2 Closed Form

The closed form technique involves an integral to obtain the two-step forecast

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$Y_N(2) = \int f[(Y_N(1) + \varepsilon), \hat{\alpha}) dF(\varepsilon),$$
$$= \int f[f(Y_N, \hat{\alpha}) + \varepsilon), \hat{\alpha}] dF(\varepsilon) \qquad (5.3.2\text{-}1)$$

where $F(\,)$ is the distribution function of ε and $f(.)$ a non-linear function of ε. It is necessary to know the distribution of ε, which is in practice generally unknown. Brown and Mariano (1989) assume a $N(0,1)$ distribution, but this has to be verified. Numerical integration can be used to approximate (5.3.2-1). Apart from the fact that this forecast can be difficult to calculate, it may be incorrect, because of the wrong assumption on the distribution of ε. The dimensionality of the error distribution will increase with the lead time of the forecast, and will consequently result in an increase in computer time.

5.3.3 Monte Carlo

For large values of N this technique has the same disadvantage as the closed form technique (in that the distribution the error terms is needed) but it is easier to implement. The two-step forecast is defined as

$$Y_N(2) = \frac{1}{T} \sum_{j=1}^{T} f(Y_N(1) + e_j) \qquad (5.3.3\text{-}1)$$

where the sequence e_j's are independent and identically distributed and chosen from the error terms. This is a particular form of numerical integration based on simulations. The mathematical expectation (5.2-7) is approximated by the arithmetic average from a sample of possible realisations of Y_{N+2} given $Y_1, Y_2, \ldots, Y_N$.

5.3.4 Bootstrap

The bootstrap technique is based on the estimated residuals. The two-step forecast is given by

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$Y_N(2) = \frac{1}{T}\sum_{k=1}^{T} f(Y_N(1) + \hat{\varepsilon}_k) \text{ where} \qquad (5.3.4\text{-}1)$$

$$\hat{\varepsilon}_k = Y_k - Y_{k-1}(1), \quad k = 2,...N$$

are the realised one-step forecast errors arising up to time N. No assumption on the distribution of the error terms is therefore necessary. The forecast improves as time advances and is easily formed (Lin and Granger, 1994:2). This technique is used for purposes of this dissertation and discussed further in Section 5.4.

5.3.5 Direct Method

This method involves the direct modelling of the relationship between Y_{N+2} and Y_N by using the same model as before but estimating a new set of coefficients directly. The previous techniques involved the use of the same function with the same set of coefficients, $f(f(\,))$, for the two-step forecasts, which decreases its quality. The $f(\,)$ is rarely known in practice and has to be approximated from a specific search. If

$$\hat{Y}_N(1) = \hat{f}(Y_N, \hat{\alpha}) \qquad (5.3.5\text{-}1)$$

a specification of the form

$$\hat{Y}_N(2) = \hat{g}(Y_N, \hat{\beta}) \qquad (5.3.5\text{-}2)$$

could be considered. For a non-parametric procedure such as neural networks this would be a sensible technique.

Lin and Granger (1994) investigated all these techniques with a simulation study for the two-step case. The results lead to a mild recommendation of the bootstrap predictor because of its small bias and not more than 5% inefficiency in mean squared error, when compared with the parametric model, using the correct specifications. Further study on broader classes of time series models is recommended.

5.4 The Bootstrap Technique

The bootstrap technique can be applied to obtain interval forecasts for an autoregressive time series. Masarotto (1990) finds the bootstrap technique useful for three reasons, namely: it is distribution-free, it takes into account that the parameters and order of the model are unknown, and improved computer technology makes the difficult calculations involved with the bootstrap technique easier. Boraine (2000) showed that the bootstrap results for linear models can be extended to non-linear time series models. A discussion is given in this section. In Section 5.4.1 the prediction limits for linear autoregressive models are given. These are the standard results given in any time series text book. The multi-step forecasts for the neural network models are introduced in Section 5.4.2 and the prediction limits for the multi-step forecasts are given in Section 5.4.3.

5.4.1 Prediction Limits for Autoregressive Models

Let $Y_1, Y_2, \ldots, Y_N$ be a stationary time series described by an $AR(p)$ model,

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \varepsilon_t,$$

From (5.2-3) it follows that the h-step ahead minimum MSE predictor for Y_{N+h} is

$$Y_N(h) = E(Y_{N+h}|Y_1, Y_2, \ldots, Y_N) = \phi_1 Y_N(h-1) + \phi_2 Y_N(h-2) + \ldots + \phi_p Y_N(h-p) \quad (5.4.1\text{-}1)$$

where $Y_N(j) = Y_{N-j}$ if $j < 0$.

The forecasts are calculated recursively and converge to the mean of the time series for large values of h.

If it is assumed that the distribution of $\ldots, \varepsilon_{t-1}, \varepsilon_t, \ldots$ is normal, the distribution of

$Y_{N+h}|Y_1, Y_2, \ldots, Y_N$ is normal with mean $Y_N(h)$ and variance $\sigma_a^2(1 + \sum_{j=1}^{h-1} \varphi_j^2)$ where the j's are the weights in the moving average representation (4.2-5) of Y_t, namely

$$Y_t = \theta_0 + \varepsilon_t + \varphi_1 \varepsilon_{t-1} + \varphi_2 \varepsilon_{t-2} + \ldots \qquad (5.4.1\text{-}2)$$

The approximation of 1-α probability limits for $Y_{N+h}|Y_1, Y_2, \ldots, Y_N$ is given by

$$\hat{Y}_N(h) \pm z_{1-\frac{\alpha}{2}} \left\{ \sigma_a^2 \left(1 + \sum_{j=1}^{h-1} \varphi_j^2\right) \right\}^{\frac{1}{2}} \qquad (5.4.1\text{-}3)$$

where $z_{1-\frac{\alpha}{2}}$ is the $100(1-\frac{\alpha}{2})^{th}$ percentile of the standard normal distribution (Box, Jenkins and Reinsel; 1994:142).

5.4.2 Multi-step Forecasts for Neural Network Models

Let $Y_1, Y_2, \ldots, Y_N$ be a stationary time series described by a non-linear neural network model

$$Y_t = f(Y_{t-1}, \ldots, Y_{t-p}; \underline{w}) + \varepsilon_t \qquad (5.4.2\text{-}1)$$

where $f(\,)$ is a non-linear function defined by the neural network architecture, p is the number of input variables in the network, and $\underline{w}$ the weight vector . The ε_t's are uncorrelated, identically distributed random variables with mean zero and variance σ_a^2. The observation at time $N+1$ can be written as

$$Y_{N+1} = f(Y_N, Y_{N-1}, \ldots, Y_{N+1-p}; \underline{w}) + \varepsilon_{t+1} \ . \qquad (5.4.2\text{-}2)$$

The minimum *MSE* forecast for a single step is (compare 5.2-6),

$$Y_N(1) = f(Y_N, Y_{N-1}, \ldots, Y_{N+1-p}; \underline{w}) \text{ with estimator} \qquad (5.4.2\text{-}3)$$

$$\hat{Y}_N(1) = f(Y_N, Y_{N-1}, \ldots, Y_{N+1-p}; \hat{\underline{w}})$$

The observation at time $N+2$ is

$$Y_{N+2} = f(Y_{N+1}, Y_N, \ldots, Y_{N+2-p}; \underline{w}) + \varepsilon_{t+2} \qquad (5.4.2\text{-}4)$$

with the minimum *MSE* forecast

$$Y_N(2) = E[f(Y_{N+1}, Y_N, \ldots, Y_{N+2-p}; \underline{w}) | Y_1, Y_2, \ldots, Y_N] \qquad (5.4.2\text{-}5)$$

For the estimation of the minimum *MSE* forecast, the bootstrap methodology can be used and therefore the bootstrap forecast for $Y_N(2)$ is proposed as

$$\hat{Y}_N(2) = \frac{1}{m}\sum_{j=1}^{m} f(Y_{N+1}^{*(j)}, Y_N, \ldots, Y_{N+2-p}; \hat{\underline{w}}) \qquad (5.4.2\text{-}6)$$

where

$$Y_{N+1}^{*(j)} = f(Y_N, Y_{N-1}, \ldots, Y_{N+1-p}; \hat{\underline{w}}) + \varepsilon_{N+1}^{*(j)} \qquad (5.4.2\text{-}7)$$

and $\varepsilon_{N+1}^{*(j)}$ is an observation, drawn with replacement, from $\hat{\varepsilon}_{p+1}, \ldots, \hat{\varepsilon}_N$ with

$$\hat{\varepsilon}_t = Y_t - f(Y_{t-1}, \ldots, Y_{t-p}; \hat{\underline{w}}) \text{ with } t = p+1, p+2, \ldots N. \qquad (5.4.2\text{-}8)$$

For a forecast of h time steps, Y_{N+h}, the estimator is

$$\hat{Y}_N(h) = \frac{1}{m}\sum_{j=1}^{m} f(Y_{N+h-1}^{*(j)}, Y_{N+h-2}^{*(j)}, \ldots, Y_{N+h-p}^{*(j)}; \hat{\underline{w}}) \qquad (5.4.2\text{-}9)$$

where

$$Y_{N+h}^{*(j)} = f(Y_{N+h-1}^{*(j)}, Y_{N+h-2}^{*(j)}, \ldots, Y_{N+h-p}^{*(j)}; \hat{\underline{w}}) + \varepsilon_{N+h}^{*(j)} \text{ and} \qquad (5.4.2\text{-}10)$$

$$Y_{N+h-p}^{*(j)} = Y_{N+h-p} \text{ if } h\text{-}p \le 0 \qquad (5.4.2\text{-}11)$$

The bootstrap procedure can be summarised in a few steps:
First fit the model (5.4.2-1) to the time series $Y_1, Y_2, \ldots, Y_N$. Then calculate estimates of the residual term using (5.4.2-8). Thirdly calculate $Y_{N+1}^*, \ldots, Y_{N+h}^*$ conditional on $Y_1, Y_2, \ldots, Y_N$:

$Y_{N+h}^* = f(Y_{N+h-1}^*, Y_{N+h-2}^*, \ldots, Y_{N+h-p}^*; \hat{\underline{w}}) + \varepsilon_{N+h}^*$ where

$Y_{N+h-p}^* = Y_{N+h-p}$ if $h\text{-}p \le 0$ and ε_{N+h}^* is an observation drawn randomly with replacement from $\hat{\varepsilon}_{p+1}, \ldots, \hat{\varepsilon}_N$. Repeat this for m times where m =100, usually. Now calculate the one- to h-step forecasts for the time series generated in the previous step using (5.4.2-3), (5.4.2-6) and (5.4.2-9).

In the next section it is shown that bootstrap methodology can also be used for the construction of prediction intervals for Y_{N+h}.

5.4.3 Prediction Limits for Y_{N+h}

The ideas used in this section are based on the method proposed by Masarotto (1990) for linear time series models. The prediction error is

$$e_N(h) = Y_{N+h} - \hat{Y}_N(h). \qquad (5.4.3\text{-}1)$$

$$\text{Let } \sigma = \sqrt{Var[Y_N(h)]} = \sqrt{Var[e_N(h)]} \qquad (5.4.3\text{-}2)$$

be the standard error of $Y_N(h)$. The standardised prediction error is defined as:

$$r_N(h) = \frac{e_N(h)}{\hat{\sigma}} . \qquad (5.4.3\text{-}3)$$

By using the bootstrap methodology the distribution of $r_N(h)$ can be approximated using the Monte Carlo algorithm.

An approximate $1\text{-}\gamma$ prediction interval for Y_{N+h} is

$$[\hat{Y}_N(h) + r_L \hat{\sigma}; \hat{Y}_N(h) + r_U \hat{\sigma}] \text{ and} \qquad (5.4.3\text{-}4)$$

r_L and r_U are the $B(\frac{\gamma}{2})-$ th and $B(1-\frac{\gamma}{2})-$ th order statistic of $r_N(h)_1^*,\ldots, r_N(h)_B^*$ where B is the number of bootstrap replications.

To estimate the standard error of $e_N(h), \sigma$, a large number of bootstrap realisations of $e_N(h)$ are required. Note that $r_N(h)$ is a function of $\hat{\sigma}$. A $\hat{\sigma}$ is required for each $r_N(h)$. For good approximation of the distribution of $r_N(h)$, at least 1000 bootstrap replications are required.

Once again the procedure can be described in a few steps:

First calculate the bootstrap time series $Y_1^*,...,Y_{N+h}^*$. To do this a neural network model of the form (5.4.2-1) is fitted to the observed time series, Y_1, Y_2,... Y_N. If $\hat{\underline{w}}$ is the estimated weight vector, the estimated residuals as in (5.4.2-8) are

$$\hat{\varepsilon}_t = Y_t - f(Y_{t-1},...,Y_{t-p};\hat{\underline{w}}) \text{ with } t = p+1, p+2,...,N.$$

The $\hat{\underline{w}}$ and residuals mentioned above are used to construct the bootstrap time series, $Y_{-3p}^*,...,Y_{N+h}^*$ by using

$Y_t^* = f(Y_{t-1}^*, Y_{t-2}^*,...,Y_{t-p}^*;\hat{\underline{w}}) + \hat{\varepsilon}_t^*$ where $\hat{\varepsilon}_t^*$ is drawn randomly with replacement from $\hat{\varepsilon}_{p+1},...,\hat{\varepsilon}_N$.

It is assumed that the bootstrap time series values $Y_{-3p-1}^* = Y_{-3p-2}^* = ... = Y_{-4p}^* = \overline{Y}$.

This assumption is necessary to do the calculation of successive values recursively.

For the second step of the process, calculate $\hat{Y}_N^*(h)$ and $\hat{\sigma}^*$ based on the first N observations. The procedure of determining $\hat{Y}_N^*(h)$ is described in Section 5.4.2 and $\hat{\sigma}^*$, the standard error of $\hat{Y}_N^*(h)$, is calculated by using a bootstrap procedure within the bootstrap procedure, that estimates the distribution of the standardised residual terms. The calculation of $\hat{\sigma}^*$ is done in the next steps.

The neural network model of the form (5.4.2-1) is fitted to the time series, $Y_1^*,...,Y_N^*$. The residuals of the model are determined by using (5.4.2-8). By using this model residuals, a bootstrap series is generated namely,

$$Y_1^{**}, Y_2^{**}...,Y_{N+h}^{**},$$

so that

$$Y_t^{**} = f(Y_{t-1}^{**}, Y_{t-2}^{**}...,Y_{t-p}^{**};\hat{\underline{w}}) + \hat{\varepsilon}_t^{**}.$$

The forecasts, $\hat{Y}^{**}_N(h)$ are calculated by taking into account the first N observations, $Y^{**}_1, Y^{**}_2 \ldots, Y^{**}_N$. The prediction error defined in (5.4.3-1) is

$$e^{**}_N(h) = Y^{**}_{N+h} - \hat{Y}^{**}_N(h).$$

This is repeated m times. The standard deviation of $e^{**}_N(h)_{(1)}, \ldots, e^{**}_N(h)_{(m)}$ is used as an estimate for $\hat{\sigma}^*$ which is

$$\hat{\sigma}^* = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}\left[\hat{e}^{**}_N(h) - \bar{e}^{**}_N(h)\right]^2}$$

Furthermore the bootstrap residual terms can be calculated by

$$r_N(h)^* = \frac{Y^*_{N+h} - \hat{Y}^*_N(h)}{\hat{\sigma}^*}$$

By repeating step one, two and the latter, which are the calculation of the bootstrap time series $Y^*_1, \ldots, Y^*_{N+h}$, as well as the calculation of $\hat{Y}^*_N(h)$ and $\hat{\sigma}^*$ and $r_N(h)^*$, B times where B must be at least 1000, the interval (5.4.3-4) can be constructed.

A practical application of forecasting a time series from a linear $AR(2)$ model is given in the following example.

Example 5.1

A Fortran program (see Appendix: Chapter 6) was developed to train the network and to implement the bootstrap for forecasting and the calculation of the prediction limits. The program uses a Gauss-Newton algorithm to estimate the parameters of the neural network.

As an example a generated *AR(2)* time series (see Example 4.1 to 4.3) consisting of 200 values is used to illustrate the calculation of the forecasts and the corresponding 95% prediction limits. The generated process is linear, and therefore it can not be expected that the neural network model would produce better prediction results than a linear model. A feed forward neural network with two input nodes for two past values of the time series and one hidden layer with a sigmoidal activation function was trained. Bootstrap methodology was used to calculated the prediction limits. Figure 5.1 and Figure 5.2 illustrate the prediction results. The results for the neural network where bootstrap methods were used and the linear model correspond more or less.

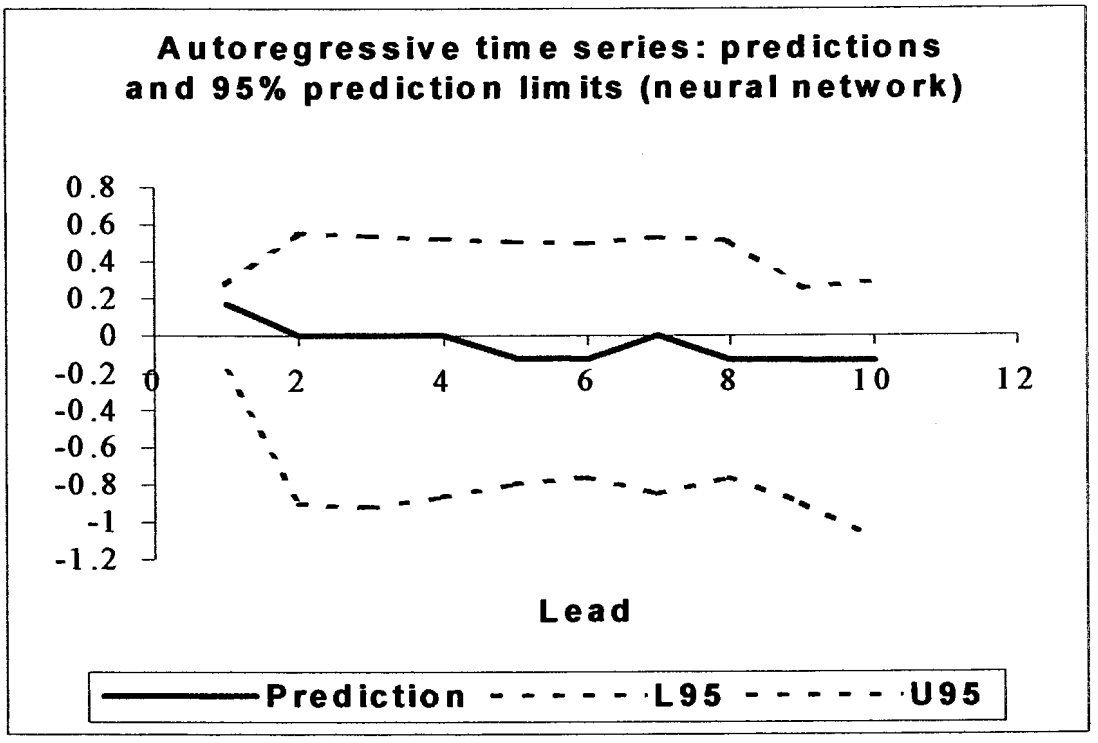

Figure 5.1: A neural network model predicting an AR(2) time series

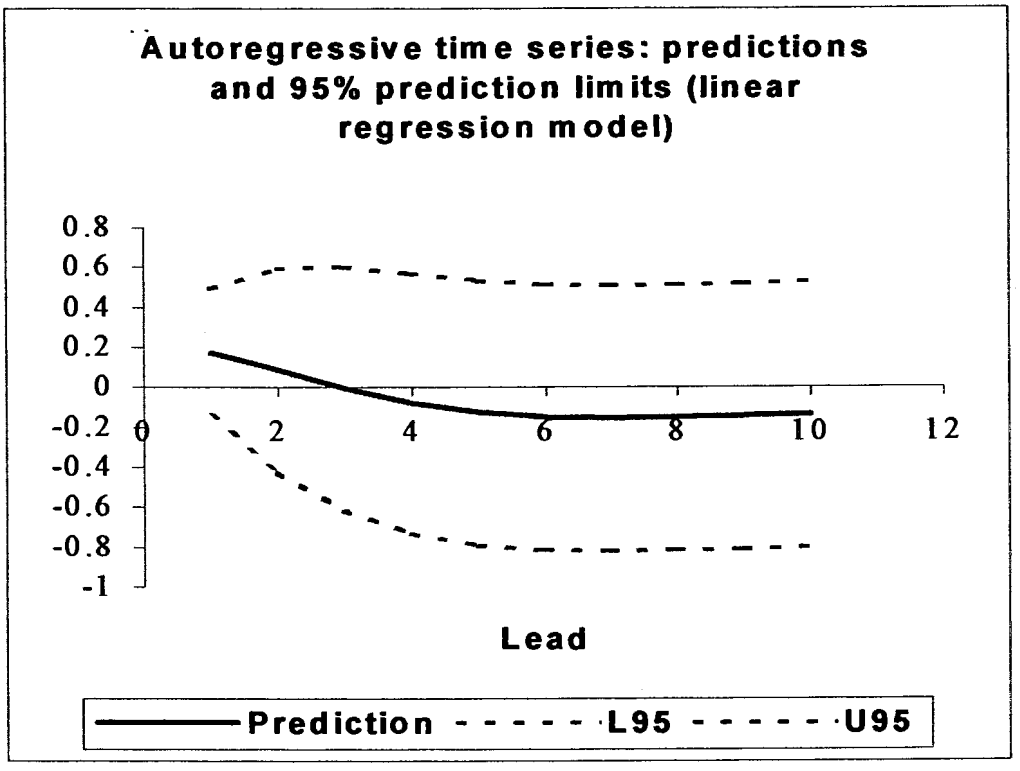

Figure 5.2: A linear regression model predicting an AR2) time series.

It is shown that bootstrap methodology can be used to calculate one- and multi-step predictions of neural networks with their standard errors and prediction limits. A minimum of 1000 time series have been resampled from the observed times series and for each resampled time series a network is trained to do one step predictions. These highly computer intensive calculations were no problem considering the ever-increasing speed of computers.

In can be seen in the two examples given that neural networks as well as linear regression models are used to model the time series. Predictions and prediction limits are calculated. The results compare reasonably well.

Further research is required to establish whether the bootstrap results can be improved. For an example, the increase in the number of bootstrap replications may lead to an improvement in the results.

Chapter 6

Forecasting Electricity Time Series by using Neural Networks

6.1 Introduction

The objective of this chapter is to forecast a time series by using neural networks. Some scientists devote careers to the building of models for electricity load forecasting. The purpose of this chapter is not to improve on the work done on model estimation but only to introduce the neural network as a forecasting tool.

Several studies have been done on electricity load forecasting. For example: the modelling of one-hour-ahead hourly electricity demand prediction has been done by Connor (1996) while Hwang and Ding (1997) focused on the construction of prediction intervals for electricity load forecasting. Lee *et al* (1992) used neural networks for short term load forecasting by dividing the data into different classes of daily and weekly load variations while Peng *et al* (1993) proposed a new strategy in selecting training cases for a neural network model.

The data used here is ESKOM data taken hourly, measuring the average electricity consumption for the Republic of South Africa. Section 6.2 deals with the analysis of the data. The different strategies and techniques used to prepare the data for estimation of the models are proposed in Section 6.3. The model is investigated in Section 6.4 by means of evaluation. Forecasting one to twelve hours ahead by using the selected model is done in Section 6.5. Unfortunately the programme package, Basic ModelGen does not include the bootstrap method as a forecasting technique. Two methods, namely the naïve technique discussed in paragraph 5.3.1 and the direct method discussed in paragraph 5.3.5 are used and compared in Section 6.5 by the use of the programme package, Basic ModelGen. The bootstrap method is then used in Section 6.5.3 on the above mentioned electricity data.

6.2 Model Identification

As mentioned earlier, ESKOM data, taken hourly, measuring electricity consumption was analysed. A one-year period was considered starting from the 25th of October

1997 to the 25th of October 1998, a total of $N = 8760$ observations. The data therefore contains seasonal components, as seen in Figure 6.1, where a four-week period is illustrated.

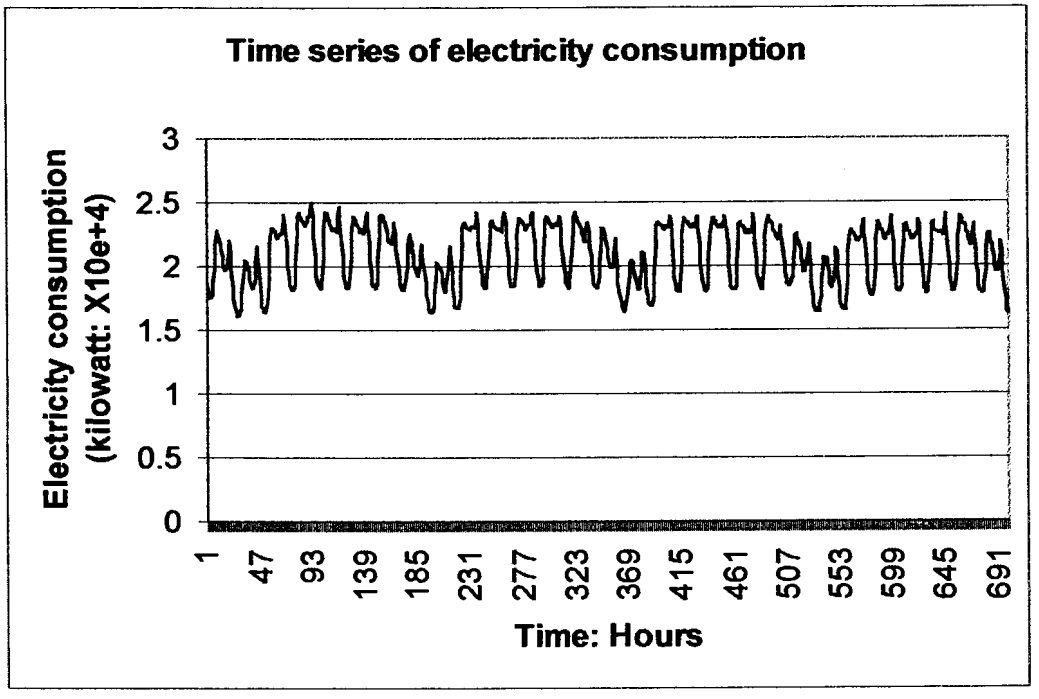

Figure 6.1: ESKOM electricity data for a four-week period

Seasonal components cause difficulties in the estimation of the model parameters. The sample autocorrelation function (ACF) and sample partial autocorrelation function (PACF) point out a definite $AR(2)$ component. Therefore the consumption in periods t-1 and t-2 will be considered as two of the inputs in the neural network model.

From the electricity data it is clear that there is more than one seasonal component. Several seasonal components can influence the behaviour of a time series. A spectral analysis can be performed to attribute the total variation in the electricity consumption to cycles of different frequencies.

The results of the spectral analysis indicated the 24-hour and the 12-hour periods respectively as the more important seasonal components.

Several other factors were considered, for example the effect of the day of the week and of public holidays, and the average temperature for each day. Galpin (1997) investigated the inclusion of rainfall, temperature and humidity as input variables in a regression

model. Although she found that these climate data decrease the forecast error, it was found that public holidays have an overriding impact on the forecasting process.

These variables were all considered as explanatory variables. Networks with different combinations of input variables and a different number of hidden inputs were trained. The model that was finally selected to describe the electricity data has as explanatory variables periodic components that were determined by a spectral analysis (see Example 4.1), and the electricity consumption of the two previous hours. If Y_t denotes the electricity consumption at time t, the proposed model is:

$$Y_t = f(Y_{t-1}, Y_{t-2}, \cos[\omega_1(t-1)], \sin[\omega_1(t-1)], \cos[\omega_2(t-1)], \sin[\omega_2(t-1)]; \underline{w}) + \varepsilon_t \quad (6.2\text{-}1)$$

as well as two nodes in the hidden layer. The ε_t's are assumed to be generated by a white noise process and $\omega_1 = 0.2618$ and $\omega_2 = 0.5236$.

The ideal network is one that is not redundant, in other words, the specific network with the fewest parameters and which represents the data the best, is preferred (Hwang and Ding, 1997:748-757).

6.3 Estimation

The data is divided into a training set (first 70% of data), a cross-validation set (10% of data excluding the training set) and the rest is used as a test set (last 20%). By calculating the *MSE* for each model, the one with the smallest *MSE* value was selected. This was done for both the feed forward neural network, which uses backpropagation, and the Elman recurrent neural network. The feedforward neural network is a function of a fixed number of previous values of a time series, while the Elman recurrent neural network, implicitly depends on all previous observations of a time series. The feedforward models considered in this dissertation, performed better as can be seen from the MSE values in Figure 6.2.

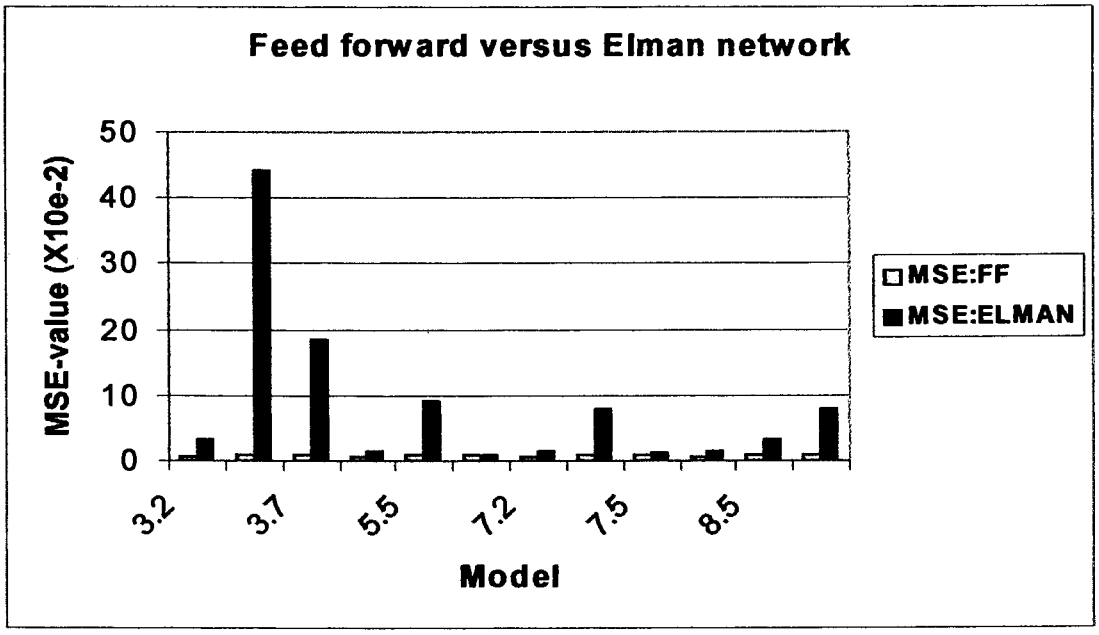

Figure 6.2: The feedforward neural network vs the Elman recurrent neural network

The labels on the horizontal axis denote the number of input nodes and the number of hidden nodes of the different models considered. For example: 3.2 indicates a neural network with three inputs and two hidden nodes. The number of parameters, W, in the fully connected model is calculated as follows:

$$W = (n_i + 1)n_h + (n_h + 1), \qquad (6.3\text{-}2)$$

where n_i is the number of input nodes and

n_h is the number of nodes in the hidden layer.

(The term, one, that is added, provides for a bias or intercept in both the input and hidden layer.)

Therefore the neural network model (6.3-1) with six input nodes and two hidden nodes, has 17 parameters or weights. This is acceptable because the total number of patterns in the training set should ideally not be less than $32W$ (Bishop, 1996:410), but $10W$ is also an accepted norm in some cases.

The software package of the Basic ModelGen™ 1.0 from Crusader Systems was used. In the training phase the input signals are passed through an activation function (see

Chapter 3). The activation function used here is a symmetric sigmoidal function (see 2.5-7). The backpropagation method was used to calculate the gradient of the error function. The training stopped after 250 iterations. After the network has been trained, the model has to be evaluated. Procedures for evaluation differ from programme package to programme package. A few of the procedures available in the Basic ModelGen™ 1.0 from Crusader Systems, are discussed in Section 6.4.

6.4 Evaluation of the model

Figure 6.3 is a representation of the root mean squared error (*RMSE*), and therefore also the mean square error (*MSE*), of the training and cross validation sets during training. This gives the user an indication of how well the training progresses with each iteration or epoch and, should there be some default, training can be stopped immediately.

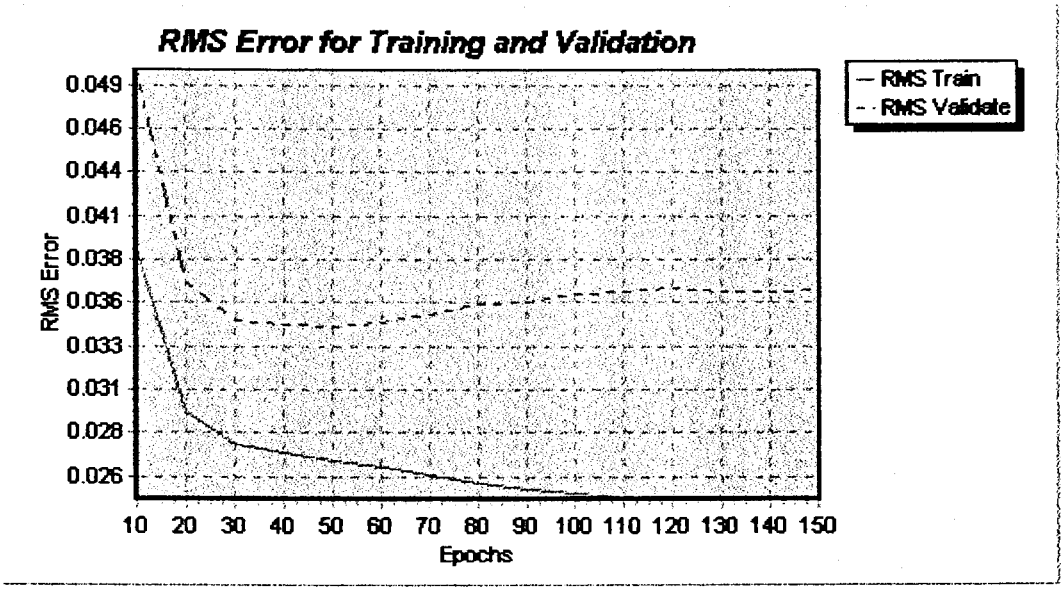

Figure 6.3: The error graph with the RMSE of the training and cross-validation sets

The RMSE for both the training and cross-validation sets fall sharply at the beginning of the iterative process. The RMSE of the training set is lower and decreases steadily, while the RMSE of the cross-validation set shows a slight upward trend before it

stabilises after about 130 iterations. This is an indication that the model should fit reasonably well to an independent data set from the same population.

The investigation of the model residuals by means of the Fourier transform gives a spectrum that is not zero. This means the residuals may still contain some information of the underlying time series.

The sensitivity analysis points out that the model is the most sensitive for the input variables Y_{t-1} and Y_{t-2}. To inspect the sensitivity of the model output to the other input variables, Figure 6.4 can be observed.

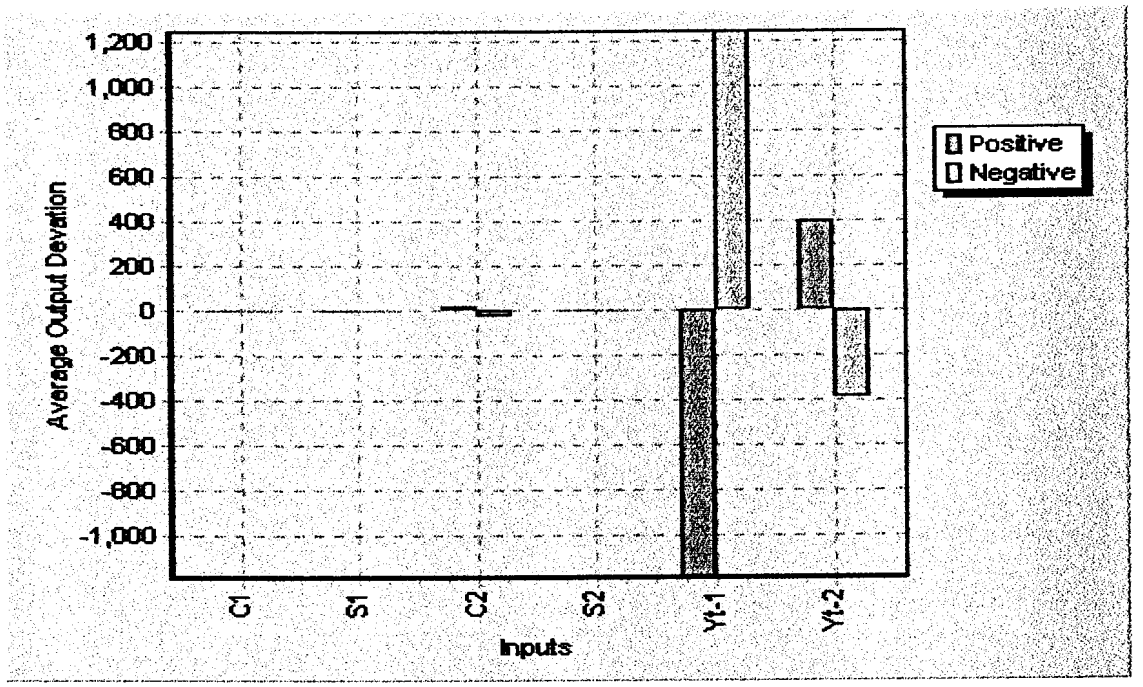

Figure 6.4: Sensitivity analysis

A graph of the output of the model and the desired outputs also gives an indication of the quality of the fit of the model. Figure 6.5 shows these results for a randomly selected section of the time series. The graph appears satisfactory because the model output is relatively similar to the target or desired output values. The last peak is over-estimated by the model. The performance of the model on weekends should be investigated further.

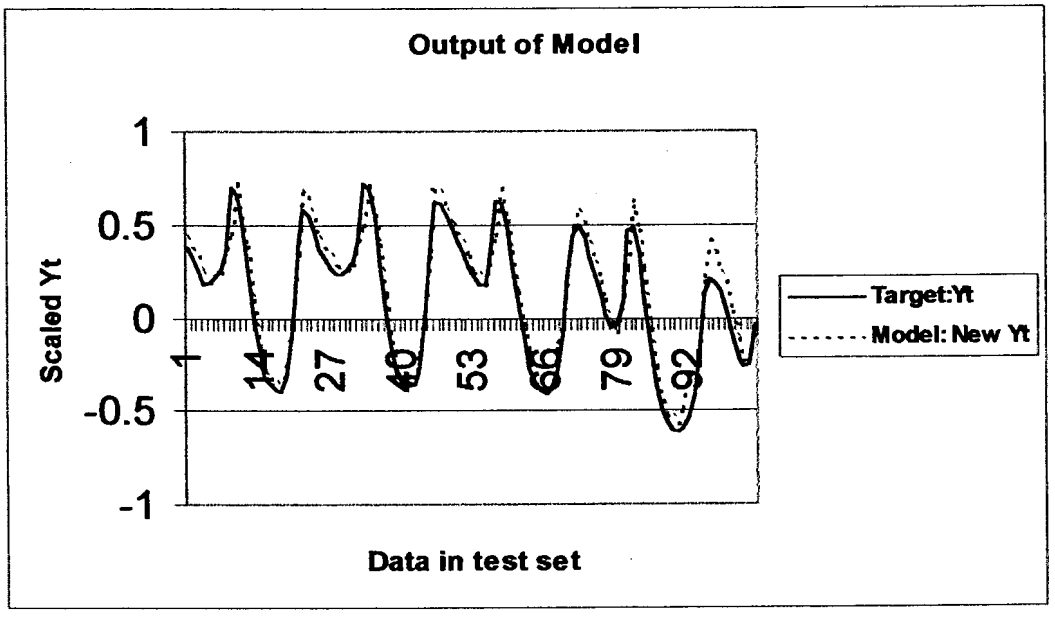

Figure 6.5: Output of model with 6 input variables and 2 neurons in the hidden layer

The following table provides the correlation coefficient, *R*, between the outputs predicted by the model and the actual desired output on the test set. The test set is used for model evaluation. This is data that was not used during the training phase.

Table 6.1: Output of the correlation analysis of the model

MODELS	Mean	Standard Deviation	R	R^2	95% Confidence Limits for R
Feed Forw	0.096	0.341	0.9821	0.9645	[0.980 - 0.984]
Desired	0.035	0.341			
Desired	0.035	0.341			

The *R* of 0.9821 is quite high and lies in the 95% confidence interval. This indicates that the model fits the data well. The narrow interval is due to the large number of observations in the test set.

The evaluation of the model has been done by means of various methods of measurements. If this model is satisfactory, the forecasting process can take place. In Section 6.5 the forecasting of twelve hours ahead is done.

6.5 Forecasting by Using the Neural Network Model

The model (6.2-1) was estimated and evaluated in the previous sections. This model is now used to forecast. Basic ModelGen™ 1.0 from Crusader Systems unfortunately is not a specialised time series analysis package. It only provides one-step predictions. For a given set of input variables, the predicted model value (6.2-1) can be calculated.

It was decided to use the naïve-, the direct method and the bootstrap method (see Section 5.3 and 5.4) to forecast twelve electricity consumption values. A Fortran programme (see Appendix: Chapter 6), was used to produce forecasts and prediction intervals using bootstrap methodology. In the next section these three methods are implemented.

6.5.1 The Naïve Method

To implement the naïve method of forecasting, a test set was selected, say $Y_1, Y_2, \ldots, Y_N$. $\hat{Y}_N(1)$ is calculated, using the estimated model.

$$\hat{Y}_N(1) = \hat{Y}_{N+1} \qquad (6.5.1\text{-}1)$$
$$= f(Y_N, Y_{N-1}, \cos[0{,}2618(N)], \sin[0.2618(N)], \cos[0.5236(N)], \sin[0.5236(N)]; \hat{\underline{w}})$$

The two-step and three-step forecasts are

$$\hat{Y}_N(2) = \hat{Y}_{N+2}$$
$$= f(\hat{Y}_N(1), Y_N, \cos[0{,}2618(N+1)], \sin[0.2618(N+1)], \qquad (6.5.1\text{-}2)$$
$$\cos[0.5236(N+1)], \sin[0.5236(N+1)]; \hat{\underline{w}})$$

and

$$\hat{Y}_N(3) = \hat{Y}_{N+3}$$
$$= f(\hat{Y}_N(2), \hat{Y}_N(1), \cos[0{,}2618(N+2)], \sin[0.2618(N+2)], \qquad (6.5.1\text{-}3)$$
$$\cos[0.5236(N+2)], \sin[0.5236(N+2)]; \hat{\underline{w}})$$

Four- to twelve-step forecasts are calculated similarly.

The following graph shows how this method performed. The forecast output values are compared with the target outputs for twelve hours.

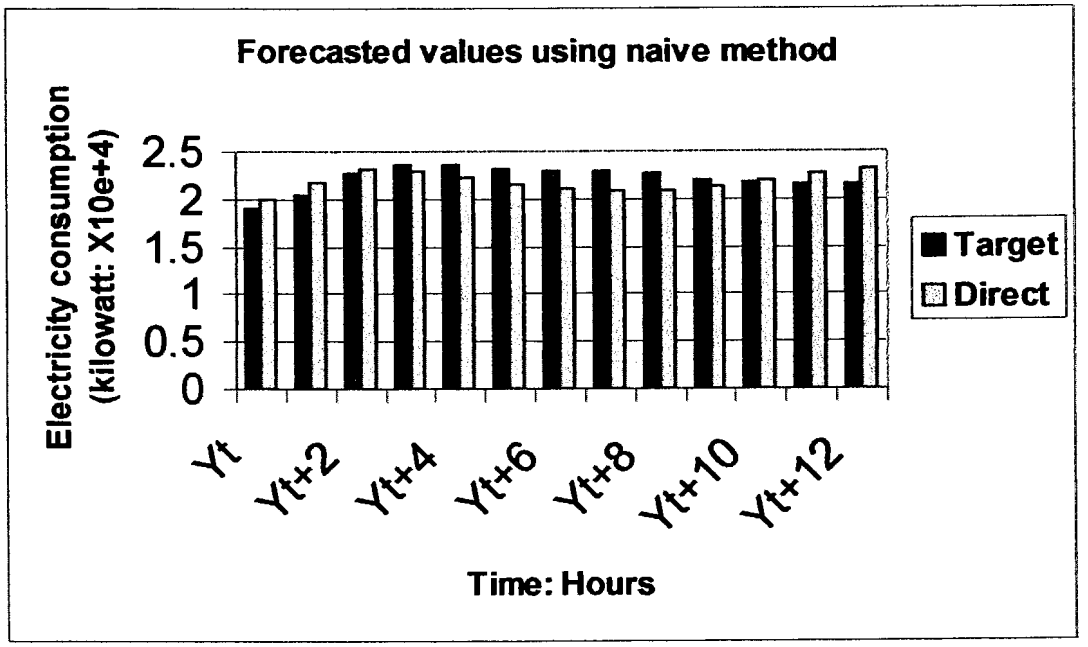

Figure 6.6: The predicted values calculated by using the naïve method

By inspection of the graph one can see that after two forecasts the forecast values predicted by the model are less than the target values. This can be a problem because it can cause an underproduction of electricity.
The next method implemented is the direct method.

6.5.2 The Direct Method

For the direct method, a different model has to be estimated for each lead time. To forecast two steps, the neural network is trained to estimate Y_{t+2} from Y_t and Y_{t-1}:

$$Y_{t+2} = g(Y_t, Y_{t-1}, \cos[0{,}2618(t+1)], \sin[0.2618(t+1)], \cos[0.5236(t+1)], \sin[0.5236(t+1)]; \underline{w}) + \varepsilon_{t+2} \qquad (6.5.2\text{-}1)$$

In general, to forecast l steps, the following model is used:

$$Y_{t+l} = h(Y_t, Y_{t-1}, \cos[0{,}2618(t+l-1)], \sin[0.2618(t+l-1)], \cos[0.5236(t+l-1)], \sin[0.5236(t+l-1)]; \underline{w}) + \varepsilon_{t+l} \qquad (6.5.2\text{-}2)$$

Figure 6.7 illustrates the performance of the direct method with once again the forecast output values compared with the target outputs for twelve hours.

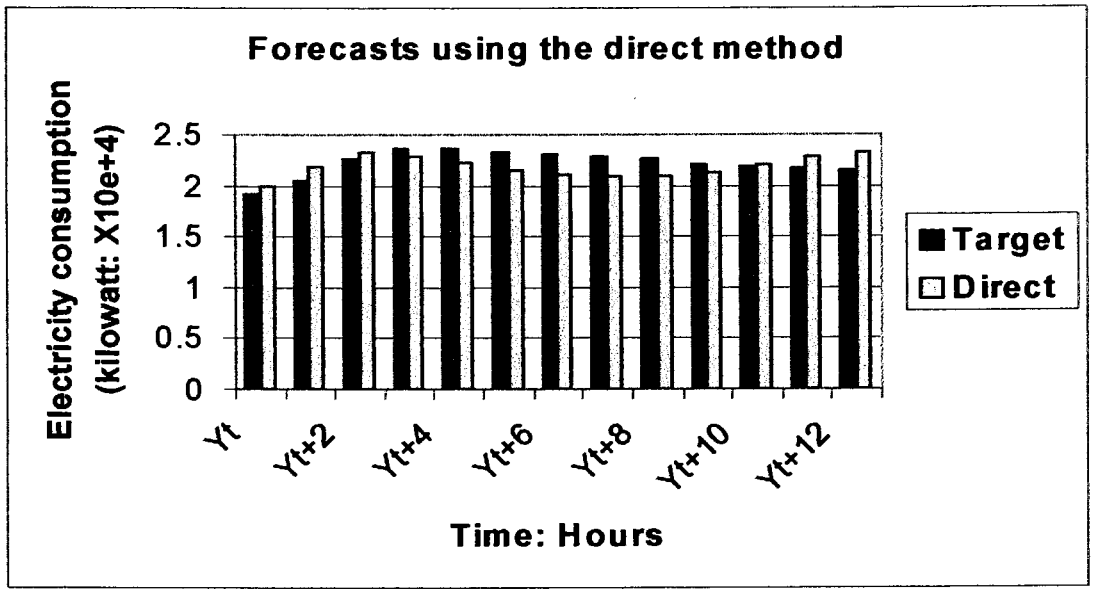

Figure 6.7: The predicted values by using the direct method

Here the model- forecast values differ a bit less from the target values. To compare the naïve method with the direct method Figure 6.8 is shown.

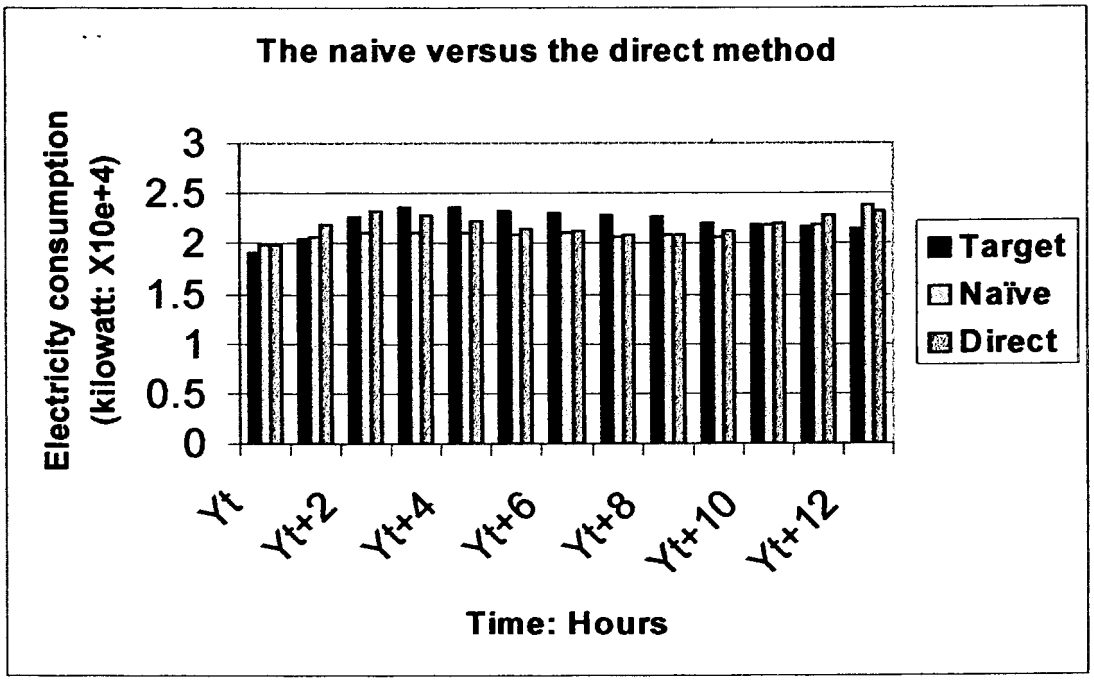

Figure 6.8: The comparison between the direct method and the naïve method

Figure 6.8 illustrates that the direct method performs better. The predicted values of the model for this method are for most of the time periods the nearest to the target.

In this chapter a neural network was fitted to hourly electricity consumption. Two methods for determining the forecasts were discussed. The direct method performed better for most of the forecast values.

Chatfield (1996) investigated alternative approaches to forecasting. The investigation of neural networks as a forecasting model showed that models with less parameters generally give better out-of-sample predictions than those neural network models with more parameters, even though this smaller models fit worse than less parsimonious models. Chatfield (1996) also found that wider prediction intervals may reflect model uncertainty better. Neural network software programs may be improved by including methods for determining prediction intervals.

6.5.3 The Bootstrap Method

The same neural network model used in Chapter 5 is used to predict the electricity consumption for 1 to 24 hours ahead. A time plot of the data is given in Figure 6.9.

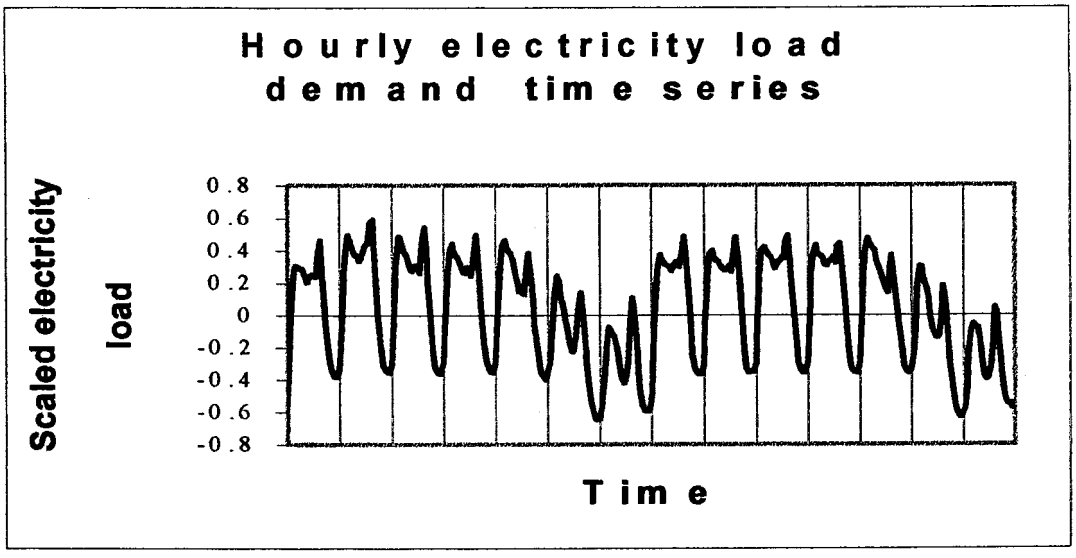

Figure 6.9: Hourly electricity load demand time series

Bootstrap methodology is used for prediction together with their standard errors and a 95% prediction interval. The results are illustrated in Figure 6.10.

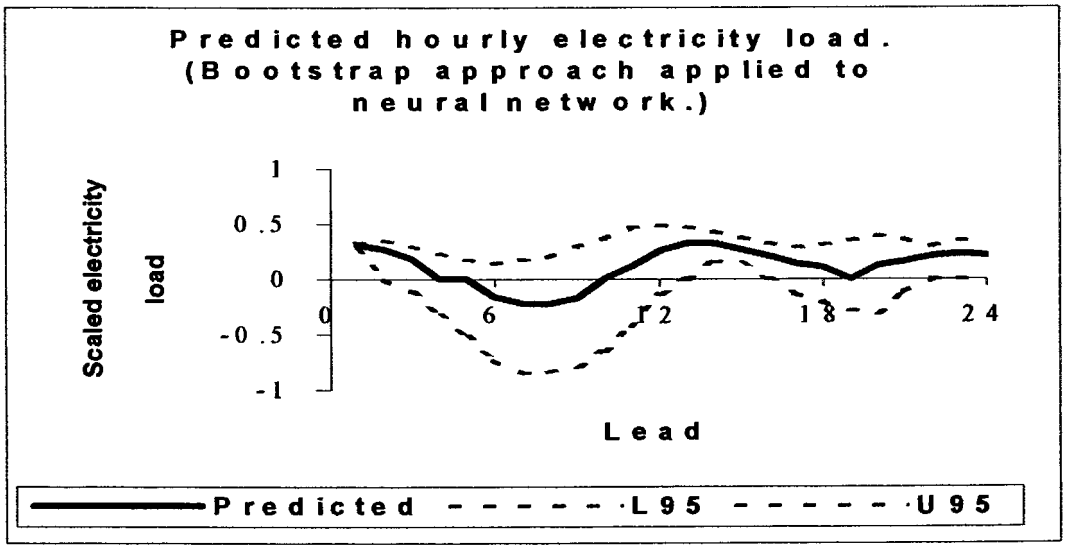

Figure 6.10: Bootstrap applied to neural network to predict hourly electricity load.

A linear regression model was also used to predict the electricity load. The same input variables were used and it was assumed that the prediction errors are normal distributed when determining the prediction limits. In Figure 6.11 the prediction limits as well as the predictions are illustrated.

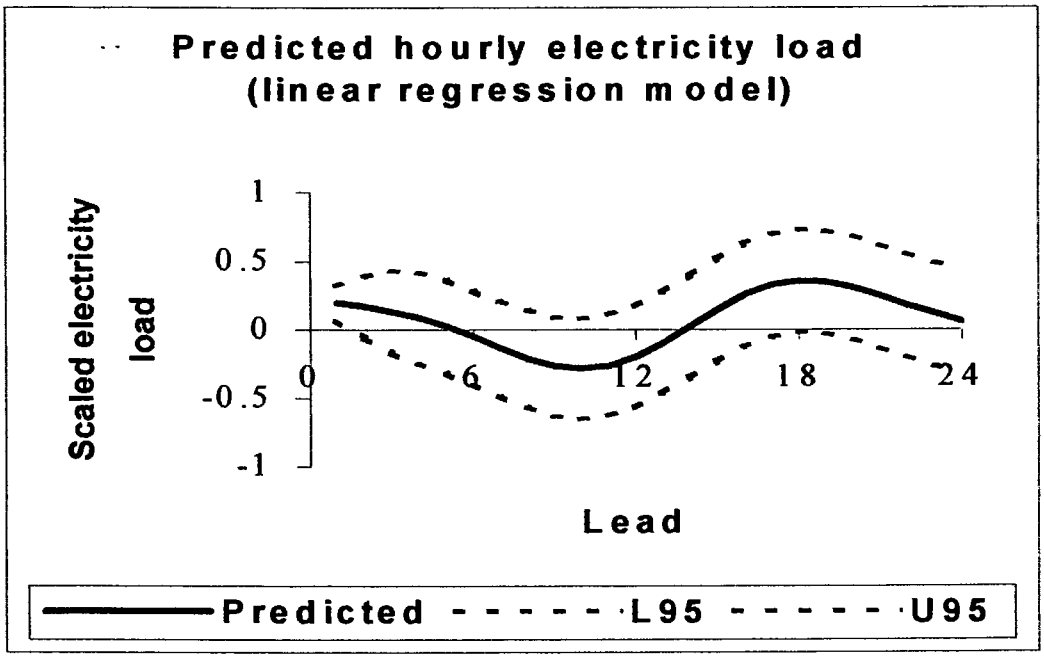

Figure 6.11: Linear regression applied to predict electricity load

It can be seen that the forecasts obtained by the neural network (Figure 6.10) and the linear model (Figure 6.11) compares well. In the case of the linear model the predictions and the prediction intervals are smoother than that obtained by the bootstrap method for the neural network model. This is because the bootstrap method is data driven and does not rely on any assumptions on the probability distribution of the data.

Appendix

Fortran Computer Programme

PREDICTNN is an artificial feed forward neural network that does one-step and multi-step time series prediction, as well as the calculation of standard errors and prediction intervals based on bootstrap methodology. (Reference).

Minimum hardware specifications:
The program runs under DOS Version 4.
Memory: 32 MB
CPU: 486
Disk space: 1 MB

Disclaimer: The authors do not take responsibility in any terms for any damages that may result from using **PREDICTNN**.

Installation:

Create a new directory, say TSPREDICT.
Copy PREDICTNN.EXE to the TSPREDICT directory.
All input and output files will be written to or read from TSPREDICT directory.

Running PREDICTNN:

Go to DOS Window.
cd\path to TSPREDICT
Type PREDICTNN.

Input:

PREDICTNN requires parameter input (from screen or ASCII file) and the training data set (from ASCII file).

Parameter input:

The user can either specify the parameters by
typing it in as prompted by the program, or
the parameters can be read from an input file:

```
Enter 0 if input from screen or 1 if input from file:
```

If the user choose to specify the parameters by typing it in from the screen, the following values must be entered as prompted by the program:

```
Enter the number of patterns in the training set:
Enter the number of bootstrap samples to be drawn:
Enter the number of input variables:
Enter the number of hidden neurons:
Enter the number of lags of the dependent variable:
Enter the number of forecasts to be calculated:
Enter 100*(1-?)% confidence interval:
Enter the maximum number of iterations:
Enter the number of nn models to fit:
Enter the input file name:
```

Notes on input parameters:

It is assumed that the training set is the first pattern in the data file up to the *number of patterns in the training set* as specified.
The *number of bootstrap samples to be drawn* refers to the number of samples to be used to calculate the prediction interval. A minimum of 1000 bootstrap samples is recommended in the literature. Note that an ANN is trained on every bootstrap sample. If you only want predictions and standard errors, you may specify a value of 3 for this parameter. The program will draw 100 bootstrap samples from the original time series to calculate predictions and standard errors.
The *number of input variables* is the number of independent/ explanatory variables in the model including the number of lagged values of the dependent/ output variable. The dependent variable is a time series.
The *number of lags of the dependent variable* is the autoregressive order of the model.
Number of hidden neurons – number of neurons in hidden layer.
Number of forecasts to be calculated – size of forecasting window.
100(1-?)% confidence interval* – a value of 0.05 will, for instance, specify a 95% confidence coefficient.
Maximum number of iterations – iterative optimization algorithm will stop after the maximum number of iterations has been reached, even if convergence to the global minimum has not taken place. More complex problems require more iterations. In relatively small problems, (20 parameters), 60 iterations should suffice.
The *number of nn models to fit* refers to the number of ANN's to be trained from random starting weights. The network with the lowest mean square error is then selected. Values between 3 and 10 should suffice.
The *input file name* is the name of the data file. (Not longer than 20 characters).

The values entered are automatically written to the file PARAMB.DAT.

Input files:

If the parameters are read from a file, the program will read the parameter values from the ASCII file PARAMB.DAT one value per line in the following order:

```
NTRAIN: Number of patterns in the training set.
NBOOT: Number of bootstrap samples to be drawn.
NINP: Number of input variables.
NHID: Number of hidden neurons.
NLAGS: Number of lags of the dependent variable.
LEAD: Number of forecasts to be calculated.
PALPHA:100*(1-?)% confidence interval.
MAXITE: Maximum number of iterations.
NFIT: Number of neural network models to fit.
FNAME: Input file name.
```

Data file:

The data file is an ASCII file. The data are real numbers separated by one or more blanks. The name of the data file is specified by the user (see input parameters).
The first pattern in the training set is given in the first row of the data file, starting with the output variable, the independent variables (if any) the first lag of the output variable, the second lag of the output variable, and so on. More than one line of data may be used for each pattern in the training set. The computer will go to a new line after a pattern has been read.

Output files:

FORECAST.OUT
Format of output:
Forecast lead (e.g. 1 for one step), forecast, standard error of forecast, lower and upper prediction limits, length of prediction interval.

BOOT.OUT
Format of output:
Bootstrap sample number, value of root mean squared error for trained network (on training set).

OPTIM.OUT
Work file used by optimization algorithm. This file may have useful information on optimization problems. Iteration results are given for the last network trained.

Neural network architecture:
Pre processing: Input and output variables are scaled to (-1:1).
Initialization of weights: A set of weights are generated using a random number generator. In order to prevent local minimum problems, a number (as specified by user) of ANN's are trained from random points, and the best ANN is selected.
One hidden layer with sigmoidal activation function.
Error function: Root mean squared error.
One output.

Dimension of problems that can be handled by PREDICTNN:

Number of input variables (NINP) and number of nodes in hidden layer (NHID):
NHID*(NINP+2)<100
Number of patterns in training set (NTRAIN): NTRAIN<1000
(NINP+1)*NTRAIN < 6000
Number of forecast lead times (LEAD): LEAD<100
Number of bootstrap samples (NBOOT): NBOOT < 2000
LEAD*NBOOT < 10000

Example:

The data file ELECS.TXT contains 716 patterns of an electricity load demand time series. Each pattern consists of the output variable and 8 input variables including two lagged values of the output variable.

Suppose we want to train a network with two nodes in the hidden layer on the first 400 patterns and produce 5 forecasts with 95% prediction limits, using 1000 bootstrap samples.

The input file, PARAMB.DAT, will look like this:

```
400
1000
8
2
2
5
0.05
40
3
elecs.txt
```

To run the program, type PREDICTNN. The program will estimate the required running time. If you want to terminate the program, hit Ctrl Break.
The output file are FORECAST.OUT and BOOT.OUT.

Chapter 7

Conclusions and Recommendations for Further Research

The temptation exists to use neural networks as "black boxes", and scientists often succumb. This is very unfortunate because a neural network is a powerful statistical device, used for many different problems. Statisticians should provide the knowledge to implement statistical methods together with neural networks to bring out its best performances.

In this dissertation, neural networks were used to analyse a time series and especially in request of the problem of forecasting. Statistical methods, used for time series analysis, are usually linear but may not succeed in forecasting a non-linear data set. This study specifically investigated the use of *ARMA* model in comparison with the neural network model for the problem of forecasting.

Two time series, namely a generated $AR(2)$ and an electricity consumption time series, were used to illustrate the process of time series analysis. Box and Jenkins proposed three phases for the analysis of a time series: identification, estimation and evaluation. These phases were given for *ARMA* models and also applied to neural network models in Chapter 4.

To illustrate the use of neural networks for the problem of forecasting, electricity consumption data as a time series was used. A neural network model was estimated to predict twelve hours ahead. The model for electricity consumption can be improved.

Different forecasting methods suitable for non-linear models were discussed. A program was developed to determine prediction intervals for the examples of the linear model and the neural network model by using the bootstrap method. The neural network program package that was used in this study can only calculate one-step predictions and does not permit the implementation of the bootstrap method. Calculation of multi-step forecasts and prediction intervals should be implemented in neural network software.

Neural networks as a versatile, adjustable tool are highly recommended. This study focussed on the identification, estimation and evaluation of neural network models for time series, and forecasting. Many of the standard techniques in statistics can be compared with neural network methodology, especially in application with large data sets.

Further research includes the use of bootstrap methods to calculate predictions, standard errors and prediction intervals for the forecasts. Bootstrap techniques can be implemented in neural network computer software packages. Research in the application and comparison by use of bootstrap techniques with neural networks are recommended.

References

ANSUJ, A.P., CAMARGO M.E., RADHARAMANAN R., and PETRY D.G. (1996). Sales forecasting using time series models and neural networks. *Computers and Industrial Engineering*, 31(1,2), 421-424.

BISHOP, C.M. (1996). *Neural networks for pattern recognition.* Oxford University press, New York.

BERTHOLD, M., HAND, D.J., (1999). *Intelligent data analysis: An introduction.* Springer- Verlag.

BROWN, B. and MARIANO, R. (1989). Residual based stochastic predictors and estimation in non-linear models. *Econometrica*, 52, 321-343.

BORAINE, H. (2000). Prediction intervals for forecasts of neural network models for time series. *Proceedings of the second ICSC Smymposium on Neural Computation*, Berlin, Germany (23-26 May, 2000).

BOX, G.E.P., JENKINS, G.M. and REINSEL, G.C. (1994). *Time Series Analysis: Forecasting and Control.* Third Edition, Prentice Hall, Englewood Cliffs, New Jersey.

CHATFIELD, C (1996). Model Uncertainty and Forecast Accuracy. *Journal of Forecasting*, 15, 495-508.

CHENG, B. and TITTERINGTON, D.M. (1994). Neural networks: A view from a statistical perspective. *Statistical Science*, 9(1), 2-54.

CONNOR, J.T. (1996). A robust neural network filter for electricity demand prediction. *Journal of Forecasting*, 15(6), 458(22).

CRUSADER SYSTEMS (Pty)Ltd. (1998). *Basic ModelGen™1.0 user's guide*, 1-142.

CRYER, J.D. (1986). *Time series analysis*. Boston, Duxbury Press.

DE JONGH, P.J. and DE WET, T. (1994). *An Introduction to neural networks*, 1-19.

FLETCHER, R. (1987). *Practical Methods of Optimization* (Second ed.). New York: John Wiley.

GALPIN, J.S. (1997). Investigation of improvement of load forecasting by using rainfall, temperature and humidity data. *ESKOM Technology research report*, 2-93.

GEMAN, S., BIENENSTOCK, E., DOURSAT, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4, 1-58.

HAYKIN, S. (1994). *Neural networks - A comprehensive foundation.* Macmillan College Publishing Company, Inc., printed in USA.

HWANG, J.T.G. and DING, A.A. (1997). Prediction intervals for neural networks. *Journal of the American Statistical Association*, 92(438*), Theory and Methods*, 748-757.

HAMILTON, J.D. (1994). *Time series analysis*. Princeton University Press, Princeton, New Jersey.

JOHANNSON, E.M., DOWLA, F.U. and GOODMAN, D.M. (1992). Backpropagation learning for multi-layer feedforward neural networks using conjugate gradient method. *International Journal of Neural Systems*, 2(4), 188-197.

KRAMER, A.H. and SANGIOVANNI-VINCENTELLI, A. (1989). Efficient parallel learning algorithms for neural networks. In Touretzky (Ed.), *Advances in Neural Information Processing Systems*, 1 , 40-48. San Mateo, CA: Morgan Kaugmann.

LE BARON, B. and WEIGEND, A.S. (1996). *Evaluating neural network predictors by bootstrapping*, http://www.cs.colorado.edu/homes/andreas/public_html/Home.html.(January 1998).

LEE, K.Y., CHA, Y.T., PARK, J.H., (1992). Short term load forecasting using an artificial neural network. *IEEE: Transactions on Power systems*,7, 124-132.

LIN, J.L. and GANGER, C.W.J. (1994). Forecasting from Non-linear Models in Practice. *Journal of forecasting*,13, 1-9.

MASAROTTO, G. (1990). Bootstrap prediction intervals for autoregressions. *International Journal for Forecasting*, 6, 229-239.

MC CULLOCH, W.S. and PITTS, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.

MC LEOD, A.I. (1978). On the distribution of residual autocorrelation in Box-Jenkins models. *Journal of Royal Statistical Society*, A40, 296-302.

MINSKY, M. and PAPERT, S. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

ORR, K.D. (1995). Use of an artificial neural network to predict ICU length of stay in cardiac surgical patients. *Society of Critical Care Medicine*. San Francisco, CA, January 1995.

O'BRIEN, P. (1997). goddard@cs.und.ac.za.

PARK, D.C., EL-SHARKAWI, M.A., MARKS II, R.J., ATLAS, L.E., DAMBORG, M.J., (1991). Electric load forecasting using neural networks. *IEEE Transactions on Power Systems*, 6(2), 442-449.

PENG, T.M, HUBELE, N.F. and KARADY, G.G. (1993). An adaptive neural networks approach to one week ahead load forecasting. *IEEE transactions on Power Systems*, 8(3), 1195-1203.

RIPLEY, B.D. (1993). *Statistical aspects of neural networks*. Invited lectures for SemStat, Sandbjerg, Denmark, 25-30 April 1992. To appear in the proceedings to be published by Chapman & Hall (January 1993).

RIPLEY, B.D. (1994). Neural networks and related methods of classification. *Journal of the Royal Statistical Society*, 56(3), 409-456.

SARLE, W.S. (1996). *Neural Network and Statistical Jargon*. saswss@unx.sas.com, (April 29, 1996).

SCHALKHOFF, R. (1992). *Pattern Recognition: Statistical, Structural and Neural approaches*. John Wiley and Sons, Inc., Canada.

SNYMAN, J.A. (1982). A New Dynamic Method for Unconstrained Minimization. *Applied Mathematical Modelling*, 6, 449-462.

SNYMAN, J.A. (1983). An updated version of the Original Leap-frog, Dynamic Method for Unconstrained Minimisation (LFOP1(b)). *Applied Mathematical Modelling*, 7, 216-218.

STERN, H.S. (1996). Neural networks in applied statistics. *American Statistical Association and the American Society for Quality Control: Technometrics*, 38(3), 205-213.

TAM, K.Y. and KIANG, M.Y. (1992). Managerial applications of neural networks in the case of bank failure predictions. *Journal of Management Science*, 38(7), 926-947.