NOTE TO USERS

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Alireza Bakhshi Kahnamouei
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)
GRADE / DEGREE

Systems Science
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Applications of Neural Networking Models in Forecasting Methods

TITRE DE LA THÈSE / TITLE OF THESIS

John C. Nash
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

David Wright

Jérome Doutriaux

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

APPLICATIONS OF NEURAL NETWORKING MODELS IN FORECASTING METHODS

By

Alireza Bakhshi Kahnamouei

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master of Science degree
in Systems Science

Faculty of Graduate and Postdoctoral Studies
UNIVERSITY OF OTTAWA
OTTAWA, ONTARIO
JULY 2005

Your file Votre référence
ISBN: 0-494-11212-3
Our file Notre référence
ISBN: 0-494-11212-3

Thank you to my Mother and Father, Mahboobeh and Gholamhossein, for their constant encouragement and support.

Table of Contents

List of Figures

Chapter 6

List of Tables

Acknowledgements

I would like to thank Professor John Nash, my supervisor, for his valuable suggestions and his support on the content of my research.

I have been very appreciative of the books on neural networks written by Abdi,et al., which have greatly assisted the development of this thesis in terms of formulas, definitions, and graphs. While I have changed the notations and lay-outs from Abdi's, but my work still bare a family resemblance.

Abstract

Artificial neural networks are frequently used in business, engineering, and scientific applications. Their increasing popularity is due to the satisfactory results achieved by using artificial neural networks applications for forecasting.

The aim of the thesis is to carry out a comparative analysis of the forecasting performance of different neural network models. The initial portion of the thesis will entail a study of the background as well as the different methodologies of artificial neural networks. Focus will be given to time series forecasting models.

Results of forecasting different sets of data show that using neural network models gives almost the same, and in some cases better results, than other forecasting methods (Paplinski, 2003).

Throughout this thesis paper, I intend to provide an introduction to neural networks, and the models applicable to time series forecasting. Particularly, in the feed-forward model of neural networks, I will provide some examples using R statistical programming language that will help the reader to observe a practical usage of R as a software that helps the user to compute complicated neural networking problems.

Introduction

The history of Neural Networks goes back to the 1940s, when McCulloch and Pitts developed the first neural networking model. Then Rosenbalt in 1963 came up with the perceptron model. His model generated substantial interest within the research community due to its ability to solve classification problems.

The complexity of artificial neural networks has made it necessary to find a way to introduce the basics of neural networks through simple techniques. Useful information can be taken from many available resources. Among these, a neural network software package is part of the *R* statistical open-source system that is available on the Comprehensive *R* Archive Network (CRAN) via the World Wide Web. *R* is a collection of a number of software facilities used for interactive data manipulation, analysis, and graphical display of arrays and matrices. We can find different packages and libraries within *R* which can be used for various mathematical and statistical computations. For example, there are Markov Chains, Feed-forward neural networks, Classification trees, and other methods, (Kohzadi, 1995).

In this paper, I present models applicable to forecasting time series problems. The goal of time series forecasting is to predict the future values based on an existing data

set. Although there are many techniques have been implemented to conduct such forecasting, in this paper, we only look at neural networking algorithms and their application to forecasting data. However, we do compare to ARIMA models for time series data, but do not explain these approaches.

This thesis includes methods for training Multi Layer Perceptrons (MLPs) for time series forecasting purposes. The MLP using the back-propagation (BP) algorithm is the conventional neural network model widely used to train and fit neural networks. BP is commonly used for forecasting, since both BP and time series forecasting use error minimization as the goal in estimating or fitting an appropriate model. To be able to explain such a complicated model as BP, we have taken the empirical approach of presenting first simple models, such as the Perceptron, then working our way towards harder models such as Auto-associative and Hetero-associative memories. Examples of some of these models will be discussed.

In the Comparative Analysis section, using R statistical software, we compare the forecasting results of feed-forward neural networks versus ARIMA models. The following are the main models of neural networks to be discussed in this thesis.

- The perceptron model is a pattern classifier: Based on the proof provided by Minsky and Papert (1969), the perceptron is not a very strong model, but it gives a very good visual look at some important concepts such as the Widrow-Hoff learning rule (Least Mean Square or the Delta rule). This learning rule is based on the continuous update of the gradient of the mean-square error(MSE)

(Kohzadi, 1995).

- The Linear Auto-associative memory: The objective of auto-associative memory is to retrieve a degraded input based on a previously stored set of inputs in the memory. Researchers widely use auto-associative memories in pattern recognition. They are very easy to analyze by both mathematical and statistical techniques, i.e., least squares estimation, singular value decomposition, and principal component analysis. In this section, the Hebbian learning algorithm is introduced in to contrast to Widrow-Hoff learning rule (Abdi, 1999).

- The linear Hetero-associative memory: This approach performs like multiple linear regression.

- Back-propagation networks: Paul Werbos introduced the back-propagation algorithm in the 1970's, but it was reintroduced by Rumelhart and McClelland in 1986 (Abdi, 1999).

Why R?

One of our goals in this thesis is to provide easier ways for researchers and practitioners to forecast and to compare forecasting techniques using neural networks (NN). While there are many software packages for statistical and neural network calculations, we want tools that are widely accessible. Fortunately, R has been developed cooperatively as an open source package for researchers based on the S-language from Bell laboratories (Chambers,1999). S-plus is the commercial version of this language (Insightful Corporation, 2005).

The popularity of R as well as its rich and flexible object oriented statistical modeling language and graphical facilities has encouraged the addition of many independent packages, such as the *forecast* package by Hyndman, and the *nnet* package by Brian Ripley.

Because of the complexity of artificial neural networks, it has been necessary for us to provide simple examples and calculations to demonstrate basic NN modeling techniques in early chapters of this thesis before conducting complicated time series modeling using the computerized application, *R*.

Chapter 1

Neural Networks

Neural Networks are adaptive statistical models which have the ability to learn and conduct estimation of the parameters of a model without using a high number of inputs at a time. This information processing model is inspired by the way biological nervous systems process data. They are composed of many interconnected processing neurons which work together to solve problems.

A neural network is built from simple units called neurons. As in biological systems, each unit is interconnected to others by a set of weighted connections. The network learns by modifying the connection weights. Each unit (neuron) corresponds to particular characteristic of a pattern which is going to be analyzed or to be used as a predicting index. All units are organized in a layered manner. In this paper, we only look at feed-forward neural networks as the most commonly used architecture for modern networks.

1.1 Feed-Forward Network

A feed-forward network is a simple aggregation of neurons in which its input, hidden, and output layers are combined together to shape the feed-forward structure. The data to be analyzed is fed to the input neurons (first layer) and then propagated to the hidden neurons (second layer) for further processing. The processed data will be transferred to next layers and so forth until the output,or last, layer.

This process (Input/Ouput) outlines the relation between input and output patterns. The networks learn the pattern by continuously adjusting the connection weights between the units. In the next chapters, we look at the weight adjustment concept as a way to modify the coefficient values of a regression model such as $y_i = \beta_0 + \beta_1 x_i$ where β_0 is the intercept, β_1 is the slope of the line, x_i is the independent variable, and y_i is the dependent variable.

Chapter 2

The Perceptron

The perceptron can be seen as a pattern recognition tool which aims to associate inputs with responses. It consists of input signals,(x_i real numbers, $i = 1, 2, ..., k$), activation functions, and output functions, (y a binary 0/1). In a perceptron, the inputs are fed directly to the outputs using a series of connection weights, Figure 1.

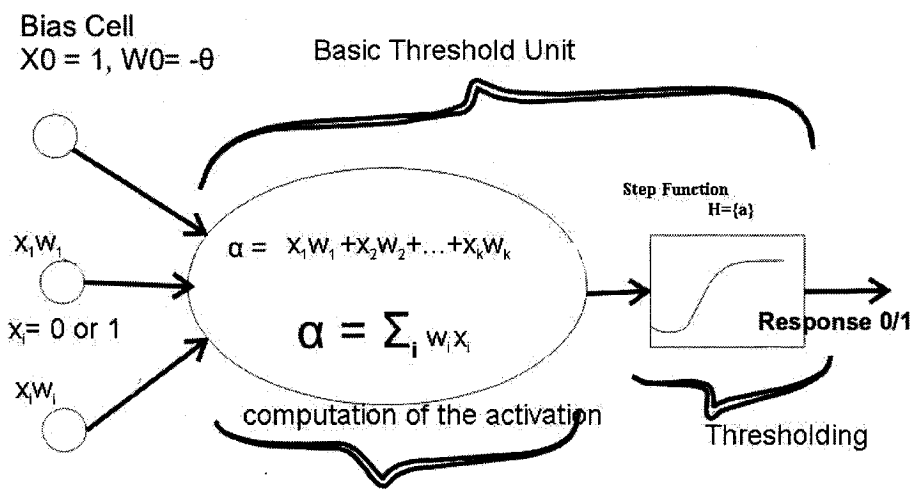


Figure 1: The Perceptron (Abdi, 1999)

2.1 The Threshold and its characteristics

A threshold unit is an object that inputs an array of weighted quantities, adds them up, and outputs a quantity if the *activation value* meets or exceeds some threshold's value. According to Abdi et. al. (1999, page 4), "*Threshold units are very simple models of neurons, which can be interpreted as simple logical machines. They can be in one of two states: inactive or active (i.e., 0 or 1.)*" Note that the threshold is also called theta (θ).

Every threshold unit computes its activation function (i.e. logistic) as the weighted sum of its inputs. When the value is above 0 (*activation* > 0), the neuron fires and responses with the activated value of 1; otherwise *activation* $<= 0$ when the response is 0. Hence, the threshold can either be active (1) or inactive (0). The degree of activation describes the state of threshold unit, Figure 1. The response of the threshold unit depends on its level of activation, which can be computed as the sum of weights coming from active input cells.

We should first compute a (activation) which is the sum of the weights coming from active input cells: $a = \Sigma w_i * x_i = w_1 x_1 + ... + w_k x_k$ (x_i, the i^{th} input cell is either 0 / 1 and w_i is the value of the weight connecting the i^{th} input cell to the threshold unit.)

The response of the threshold units can be given by *step function*(H), which can be identified as

- Ha= 0 for $a \leq \theta$, and 1 for $a > \theta$.

If we consider the specified threshold's value (i.e. an specified weight) as θ then the threshold unit is in the active state if $a > \theta$, otherwise is inactive, $a <= \theta$. We always implement thresholds by keeping one input cell active. This is one of the reasons for having the bias cell (the zero-input cell with $-\theta$ weight), Figure 1. If threshold is treated as a weight, thresholding can be implemented by keeping one input cell always active. Using the bias cell, our step function changes to (Abdi, 1999):

- Response = 1 if $a + w_0 > \theta$, 0 if $a + w_0 \leq \theta$

2.2 Implementing Logical Functions

Logical variables can be represented as binary , i.e. 0 or 1, numbers. Thus a logical function of two logical variables, for example, OR, has only four inputs as [0 0], [0 1], [1 0], [1 1], with responses 0,1,1,or 1, respectively. Looking at this concept from a perceptron aspect, for two active inputs of x_1 and x_2 (both with values of 1) , the response is equal to 1, considering w_1 ,w_2 both equal to 1(the response corresponds to the OR function, $a = w_1x_1 + w_2x_2$ gives $a = (1 * 1) + (1 * 1)$ or $(a = 2)$, hence the response of the output is 1, $(a + w_0 > \theta.)$)

2.3 The Learning Process for The Perceptron

The perceptron learns in a recursive manner by continuously modifying the weights. The following simple learning procedure is based on Rosenblatt's (1959) learning law. More complicated rules are going to be discussed in the next stage (Abdi, 1999):

1. Initialize the weights to a small random number, $W_{[0]}$ is a matrix.

2. For any input matrix , calculate the output , O (i.e. desired output), predicted output ($\hat{O}$), and the error ($\varepsilon = \hat{O} - O$); Note we use step function (H) to calculate the predicted output ($\hat{O} = H(W^T X)$.

3. Update the weights in every steps

4. $W_{[k+1]} = W_{[k]} + (\eta\varepsilon) * x_i$, i.e, k is the iteration number; $0 < \eta < 1$ is a parameter used to control the adaption rate which is generally chosen randomly (Paplinski, 2003).

2.4 The Widrow-Hoff learning Law (The Least-Mean-Square Law)

The Widrow-Hoff supervised learning takes place when the perceptron learns that a mistake has happened and what is the right response. Basically, it is the instantaneous update of the correlation matrices, and the gradient of the MSE (mean-squared error). In this type of learning, each cell learns without being concerned about the output of other cells. Each time a mistake occurs, the cell adjusts its weight in order to make the mistake less likely to occur when the same stimulus is presented again (Abdi, 1999) .

2.4.1 Learning by iteration

For learning purpose, present a set of patterns (randomly) to the perceptron. If after the first epoch (a full set of iterations) the perceptron still makes mistakes, we start a

new epoch with another arbitrary order (i.e, random set of patterns can be generated by *set.seed* command of *R*, chapter 6.) This continues until we see no mistakes by the perceptron.

$W^T_{[K+1]} = W^T_{[K]} + \eta(O_i - \widehat{O_i})x^T_i = W^T_{[K]} + \Delta_{W_{[K]}}$, $\Delta_{W_{[K]}}$ is added to the weights of the active input cells ($W_{[K]}$) in order to take into account the error made by the output cell,i.e. K is the iteration number (Abdi, 1999.)

- K is the iteration number
- η is a positive constant between 0 and 1
- Δ_W is the needed change in W; this quantity takes into consideration the state of the input cell as well as the error made by the output cell
- O is the desired response (0/1)
- $\widehat{O}$ is the response;$(O - \widehat{O})$ is the error

Eta (η) is a small value which needs to be added or subtracted from the weights. Selecting η is very delicate; an appropriate η speeds up the process of learning. A suggested value for η is 0.65 when it is not a function of K (the iteration index). Otherwise, we pick a high number (i.e. 0.85) and then let the iteration decide the true value of η (Abdi, 1999 page 11-14).

The following is the input and output of the 'OR' logical function:

$$\begin{array}{c} x_1\ x_2\ ->\ O \\ \left(\begin{array}{cccc} 0 & 0 & -> & 0 \\ 1 & 0 & -> & 1 \\ 0 & 1 & -> & 1 \\ 1 & 1 & -> & 1 \end{array}\right) \end{array}$$

We can rewrite the table in a matrix format,

$$X^T = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right],$$

$$\left(\begin{array}{ccccccc} x_1^T & = & 1 & 0 & 0 & -> & 0 \\ x_2^T & = & 1 & 1 & 0 & -> & 1 \\ \ldots\ldots\ldots & . & . & . & . & .. & . \\ x_4^T & = & 1 & 1 & 1 & -> & 1 \end{array}\right)$$

The above description of a logical *OR* function during the training of our network can be implemented in *R*, Appendix 2.

2.4.2 Training an *OR* Network, a numerical approach

In order to learn the function of OR by a perceptron, we need a perceptron which is composed of three input cells: one cell to implement thresholds ($x_0 = 1$), and two cells for the values of the logical function. First, we assume the initial weights are zero,

$$W_{[0]} = \left[\begin{array}{c|c} w_0 & 0 \\ w_1 & 0 \\ w_2 & 0 \end{array}\right],$$

We use x_2^T (i.e. 2nd input is randomly chosen) to find the right set of weights for the OR network.

$a = W_{[0]}^T X_2 = (0 \times 1) + (0 \times 1) + (0 \times 0) = 0$, Hence $\widehat{O} = H_{a=0} = 0$(step-function)

The error can be calculated as $\triangle w_i$ ($O = 1$ the target value, and $\hat{O} = 0$ the response of the output cell). Suppose the value of eta (η) is equal to 0.1 (i.e. we can consider any number between 0 and 1). Hence, the value of the correction to the inputs are:

- $\Delta w_0 = \eta(O - \hat{O})x_0 = 0.1 \times (1 - 0) \times 1 = 0.1$
- $\Delta w_1 = \eta(O - \hat{O})x_1 = 0.1 \times (1 - 0) \times 1 = 0.1$
- $\Delta w_2 = \eta(O - \hat{O})x_2 = 0.1 \times (1 - 0) \times 1 = 0.1$

Now, we must add the value of w_0 to Δw_0:

$W_{[1]} = W_{[0]} + \Delta_{W_{[0]}} =$

$$\begin{bmatrix} w_0 = 0 \\ w_1 = 0 \\ w_2 = 0 \end{bmatrix} + \begin{bmatrix} \Delta_{w_0} = 0.1 \\ \Delta_{w_1} = 0.1 \\ \Delta_{w_2} = 0.1 \end{bmatrix} = \begin{bmatrix} w_0 = 0.1 \\ w_1 = 0.1 \\ w_2 = 0.1 \end{bmatrix}$$

If we continue the process with the 3rd input, the correction weight matrix should be applied accordingly (i.e. x_3^T):

$a = W_{[1]}^T.X_2 = (0.1 \times 1) + (0.1 \times 0) + (0.1 \times 1) = 0.1$, and the response function is $\widehat{O} = Ha = 0.1 = 1$

The corrections applicable to the weights are as follows:

$$\Delta w_0 = \eta(O - \hat{O})x_0 = 0.1 \times (1 - 1) \times 1 = 0$$

$$\Delta w_1 = \eta(O - \hat{O})x_1 = 0.1 \times (1 - 1) \times 0 = 0$$

$$\Delta w_2 = \eta(O - \hat{O})x_2 = 0.1 \times (1 - 1) \times 1 = 0$$

To find out the right connection weights for the *OR* function, we need to apply the error correction to the perceptron weights (i.e. $W_{[1]}$ becomes $W_{[2]}$):

$$W_{[2]} = W_{[1]} + \Delta_{W_{[1]}} = \begin{bmatrix} w_0 = 0.1 \\ w_1 = 0.1 \\ w_2 = 0.1 \end{bmatrix} + \begin{bmatrix} \Delta_{w_0} = 0.0 \\ \Delta_{w_1} = 0.0 \\ \Delta_{w_2} = 0.0 \end{bmatrix} = \begin{bmatrix} w_0 = 0.1 \\ w_1 = 0.1 \\ w_2 = 0.1 \end{bmatrix}$$

It is also possible to represent the state of neurons as +1 and -1 (called excitation and inhibition), and 0 as inactive. Using +1, and -1 is a useful way of dealing with binary, but the inactive state, 0, offers us a ternary perceptron. Let us see what changes this choice implies for our computations.

To train a ternary (+1, -1 and 0 valued) perceptron, we use the model to distinguish the difference between 4-door Sedan cars, and 4 × 4 pick-up trucks. (i.e. $x_{1,1}$ = the first element of a car (i.e. number of doors), $x_{2,1}$ = the second element of a car , ...)

We look at number of doors (2/4 door), number of cylinders (4/*morethan*4), and the type of rear-back (*tailgate/trunk*) as three features in each vehicle; X_k is a 3-dimensional vector.

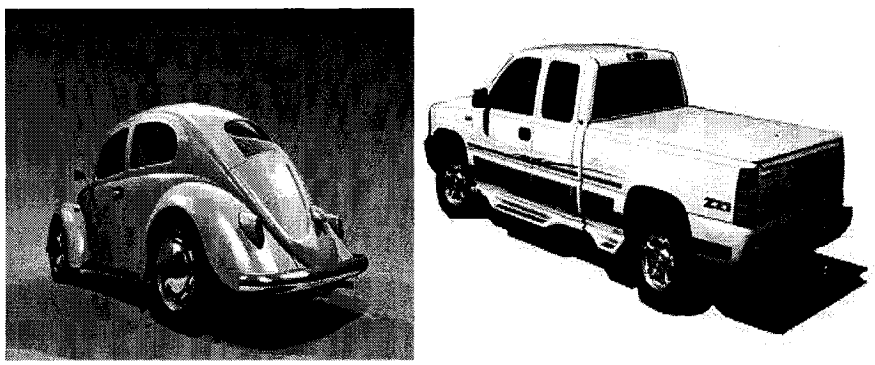


Figure 2: Vehicle Images; 4-door Sedan and 4*4 Trucks

First, we allocate excitation and inhibition characteristics to each vehicle.

In our example, we have 3 different cars and trucks.

$$\text{Cars: } \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix}, \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix}, \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix}$$

$$\text{Trucks: } \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}, \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix}, \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}$$

$X_1^T = [+1 + 1 + 1]$,represents the first car, and

$X_6^T = [+1 - 1 - 1]$, represents the 3^{rd} truck (i.e. 6^{th} vehicle).

Every cell of the perceptron corresponds to one of the 3 elements by taking the value of the element (if the feature is not available, we consider it as 0). With the given information, any perceptron with three input cells and one output cell can categorize a car or a truck of the types considered. We define the output as cars correspond to a value of 1 and trucks to a value of 0. Given the value of $\eta = 1$ and 3 input neurons, every perceptron needs 3 iterations to learn correctly how to categorize cars and trucks, given the availability of all the features.

We need the following formula to find the right weights for creating the appropriate network for cars.

$$W_{[K+1]}^T = W_{[K]}^T + \eta(O - \widehat{O})X_i^T = W_{[K]}^T + \Delta_{W_{[K]}}$$

The initial weight, $W_{[0]}^T = [0\ 0\ 0\ ]$,

$a = \sum W^T.X_1 = 0$, $\widehat{O} = 0$,$O = 1$, $\eta = 1$, and the error is $O - \widehat{O} = 1$. Using

$W_{[K+1]}^T$ formula we get,

$$W_{[1]} = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix},$$

Respectively, in the second iteration, our weight matrix becomes,

$$W_{[1]} = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix},$$

$a = \sum W^T.X_2 = +1$, $\widehat{O} = 0$, $O = 1$, $\eta = 1$, and the error is $O - \widehat{O} = 0$

By calculating $W^T_{[2]}$, we get

$$W_{[2]} = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix},$$

Now, in the second iteration, our weight matrix becomes,

$$W_{[2]} = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix},$$

$a = \sum W^T.X_3 = +1$, $\widehat{O} = 1$, $O = 1$, $\eta = 1$, and the error is $O - \widehat{O} = 0$

again, by calculating $W^T_{[K+1]}$, we get

$$W_{[3]} = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix},$$

Likewise, repeating the same process for trucks (i.e. $O = 0$), we get

$$W_{[3]} = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix},$$

as the final weight vector.

Using this weight vector is equivalent to counting the number of positive feature values for the vehicles. Hence, if there are more negative values than positive ones then

the vehicle is categorized as a truck, otherwise it is a car. A perceptron can make a decision even if a value of a feature is not available.

Chapter 3

Linear Auto-associative Memories

Linear Auto-associative Memories are a feed-forward single-layer networks where the output is produced in a single feed-forward computation. All the units operate in parallel, thus they name it associative memory. The objective of using auto-associative memory is to store a set of input pattern and to retrieve the stored patterns when cued with incomplete versions of these input patterns. Their content addressability helps researchers conduct pattern recognition using the algorithms, Kohonen (1977), i.e. image processing.

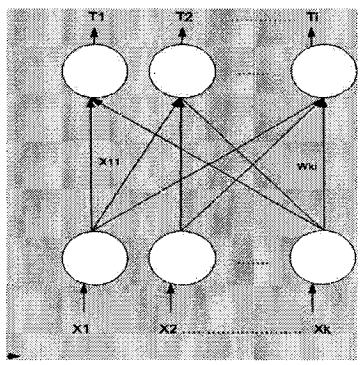


Figure 3: Auto-associative memory

Linear auto-associative memories are built from fully interconnected linear units (very similar to the basic threshold). The only difference between threshold and linear units

is the transfer function used to transform the activation into a response. Before transforming the activation to response, the linear units compute their activation level by summing up all the weighted external activation values, $a = \Sigma_i w_i.x_i$. According to Abdi, the output or response of the cell is proportional to the activation function, hence response of the network is $\gamma a = \gamma W^T X$ (γ is a constant) which is proportional to its activation.

An auto-associative memory is composed of i cells which has an input matrix, X_i, and weight matrix of $W_{i,i'}$. The connection between two different cells is bidirectional and generally symmetrical, $i_1$and i_2, i.e. $W_{i_1 i_2} = W_{i_2 i_1}$ (Abdi, 1999 p. 23-24).

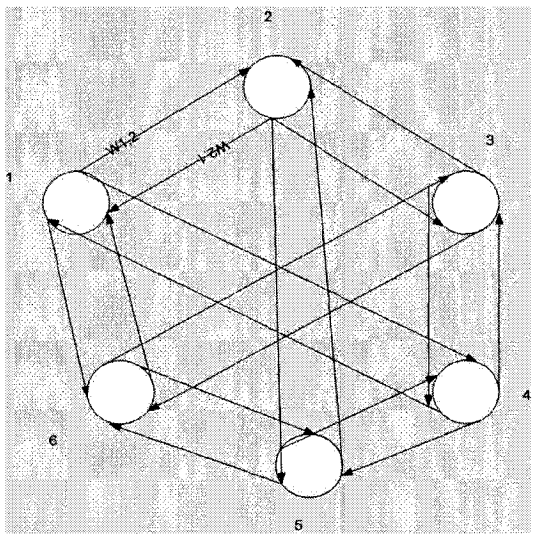


Figure 4: Auto-associative memory; connection weights

3.1 Hebbian vs. Widrow-Hoff Learning Rule

Here we look at Hebbian learning rule and its application in associative learning through a numerical example. Hebbian learning rule is an unsupervised learning algorithm and can be related to self-organizing neural networks. It extracts from data a set of principal directions. Each of these directions is represented by a relevant connection weight vector. During the learning period, Hebbian rule declares that the connection between two different neurons is proportional to the correlation of their activation values. The Hebbian learning rule is defined by $W = \gamma \Sigma X.X^T$, i.e. multiply each of the i patterns vector stored in a linear auto-associative memory by its transposed and adding the resulting product matrices. We consider γ, the proportionality constant, equal to "1" throughout the next numerical example (Hebbian Learning Rule). We look at the previous sample from chapter 2, i.e. car and truck comparison :

- $W_{[k+1]} = W_{[k]} + X_{[k+1]}.X^T_{[k+1]}$
- Number of doors ($2- > +1$ or $4- > -1$)
- Number of cylinders ($4- > +1$/ more than $4- > -1$)
- Type of rear-back (tail gate $- > +1$ or trunk $- > -1$)
- $X_1^T = [+1 + 1 + 1]$ car (1^{st} Vehicle)
- $X_2^T = [-1 - 1 - 1]$ truck (2^{nd} Vehicle)
- $X_3^T = [-1 + 1 - 1]$ truck (3^{rd} Vehicle)
- $X_4^T = [+1 + 1 - 1]$ car (The fourth Vehicle)

We put all the connection weights in a $3 * 3$ matrix. We first initialize all the weights to zero (i.e. for convenience). Starting from X_1^T :

$$W_{[0]} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$W_{[1]} = W_{[0]} + X_1 X_1^T =$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} \times \begin{bmatrix} +1 & +1 & +1 \end{bmatrix} = \begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & +1 \\ +1 & +1 & +1 \end{bmatrix}$$

$$W_{[2]} = W_{[1]} + X_2 X_2^T =$$

$$\begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & +1 \\ +1 & +1 & +1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \times \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} +2 & +2 & +2 \\ +2 & +2 & +2 \\ +2 & +2 & +2 \end{bmatrix}$$

$$W_{[3]} = W_{[2]} + X_3 X_3^T =$$

$$\begin{bmatrix} +2 & +2 & +2 \\ +2 & +2 & +2 \\ +2 & +2 & +2 \end{bmatrix} + \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} \times \begin{bmatrix} -1 & +1 & -1 \end{bmatrix} = \begin{bmatrix} +3 & +1 & +3 \\ +1 & +3 & +1 \\ +3 & +1 & +3 \end{bmatrix}$$

$$W_{[4]} = W_{[3]} + X_4 X_4^T =$$

$$\begin{bmatrix} +3 & +1 & +3 \\ +1 & +3 & +1 \\ +3 & +1 & +3 \end{bmatrix} + \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix} \times \begin{bmatrix} +1 & +1 & -1 \end{bmatrix} = \begin{bmatrix} +4 & +2 & +2 \\ +2 & +4 & 0 \\ +2 & 0 & +4 \end{bmatrix}$$

From $W_{[4]}$, we can see that the connection between cell 1 and 2, cell 1 and 3, cell 2 and 1, and 3 and 1 have a value of 2. This means that the values of the features (doors, cylinders, and trunk shape) corresponding to these cells are the same ,($+1 and +1$) or (-1*and*-1), in 3 vehicles (+1 =the same), and different for the rest (-1 =different), i.e. $3 * (+1) + 1 * (-1) = +2$. In Hebbian learning rule, the matrix of connection weights captures the statistical structures of the set of data (i.e. the structure of the vehicles). These regularities comprise the knowledge of the memory. We can use this knowledge to reconstruct partially ablated input patterns. Without the indication of the number of cylinders (feature 2 = 0), X_4^T= [+1 0 +1] car (4^{th}vehicle), we can use the network to fill the connectivity pattern, $W_{[4]} = \begin{bmatrix} +4 & \mathbf{+2} & \mathbf{+2} \\ \mathbf{+2} & +4 & 0 \\ \mathbf{+2} & 0 & +4 \end{bmatrix}$

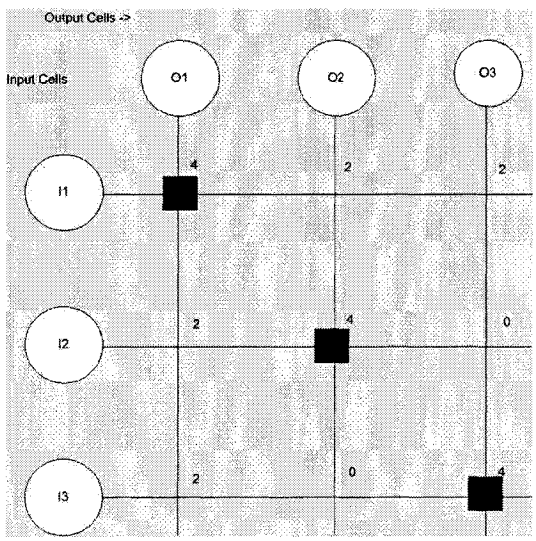


Figure 5: Connection weights on the weight matrix, Abdi (1999)

As we can see the connection weight between features 3 and 1 is stronger than 3 and 2 , i.e. $(2 > 0)$. Since feature 1 has the strongest connection with 3, hence, we give the value of feature 1 to the missing feature 3 (i.e. we give the value of $+1$ to feature 3 if feature 1 is $+1$, and we give the value of -1 if feature 1 is -1.) For example in X_4 (4^{th}) the missing value of feature 3 is equal to -1 just because feature 1 is -1.

To predict the i^{th} learned pattern from the memory, we can pre-multiply the vector x_i (i.e. the input pattern) by the connection weight matrix:

$$\widehat{X_{i=memory-response}} = W.X_i$$

The quality of memory response can be checked by the following formula (Abdi,1999):

$\cos(\widehat{X_i}, X_i) = \frac{[X_i.(\widehat{X_i^T})]}{[\|X_i\|.\|X_i^T\|}$,i.e $\|X_i\| = \sqrt{X_i^T X_i}$.
A cosine value closer to "1" between the two vectors expresses the similarity between the stimulus learned by the memory and its reconstruction (i.e. cosine =1 means perfect reconstruction).

To retrieve the first vehicle, $\widehat{X}_1$, we must multiply the vector X_1 by the connection weight matrix, $\boldsymbol{W}$:

$$\widehat{X_1} = W.X_1 = \begin{bmatrix} +4 & +2 & +2 \\ +2 & +4 & 0 \\ +2 & 0 & +4 \end{bmatrix} \times \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} = \begin{bmatrix} +8 \\ +6 \\ +6 \end{bmatrix}$$

$$\widehat{X_1^T} X_1 = \begin{bmatrix} +8 & +6 & +6 \end{bmatrix} \times \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} = 20$$

$\cos(\widehat{X_1}, X_1) = \frac{20}{11.66 \times 3} = 0.57$; This indicates that the retrieval is not a very good one (i.e. $0.57 < 1$). If we bring the complete estimation of the set of vehicles, we have the following:

$$\widehat{X} = W.X = \begin{bmatrix} +4 & +2 & +2 \\ +2 & +4 & 0 \\ +2 & 0 & +4 \end{bmatrix} \times \begin{bmatrix} +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 \\ +1 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} +8 & -8 & +4 & +4 \\ +6 & -6 & -2 & +6 \\ +6 & -6 & -6 & -2 \end{bmatrix}$$

Every column represents the reconstruction of each vehicles.

Although we clearly observed the ability of Hebbian learning algorithm in reconstruction of some stored input patterns, we can't neglect the limitations of employing such algorithm. For example, if we assume that the first and third vehicles share 2 common features and 2 different features

(i.e. an additional feature for each car, we assume $X_1 = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$, and $X_3 = \begin{bmatrix} +1 \\ -1 \\ +1 \\ -1 \end{bmatrix}$), an orthogonal behavior appears, (perfect recall, Abdi (1999)), meaning the cosine between both vectors is equal to one. This creates a perfect recall from the memory.

- $W = X.X^T = \begin{bmatrix} +1 & +1 \\ +1 & -1 \\ +1 & +1 \\ +1 & -1 \end{bmatrix} \times \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \end{bmatrix} = \begin{bmatrix} +2 & 0 & +2 & 0 \\ 0 & +2 & 0 & +2 \\ +2 & 0 & +2 & 0 \\ 0 & +2 & 0 & +2 \end{bmatrix}$
- $\widehat{X_1} = W.X_1 = \begin{bmatrix} +4 \\ +4 \\ +4 \\ +4 \end{bmatrix}$
- $\widehat{X_1}.X_1 = 16$
- $\|X_1\|.\|\widehat{X_1^T}\| = 16$
- $\cos(\widehat{X_1}, X_1) = \frac{[X_1.(\widehat{X_1^T})]}{[\|X_1\|.\|\widehat{X_1^T}\|}} = \frac{16}{16} = 1$, this is a perfect fit. Identical calculations will generate the same results for the X_3. This can also be recognized by the orthogonal (correlated) behavior, i.e. $X_1^T.X_3 = 0$ (scalar product).

Now, if we look at the same vehicles with the original specifications (i.e. only three features, one in common), we can see an imperfect recall from the memory according to the following calculations:

- $W = X.X^T = \begin{bmatrix} +1 & -1 \\ +1 & +1 \\ +1 & -1 \end{bmatrix} \times \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & -1 \end{bmatrix} = \begin{bmatrix} +2 & 0 & +2 \\ 0 & +2 & 0 \\ +2 & 0 & +2 \end{bmatrix}$
- $\widehat{X_1} = W.X_1 = \begin{bmatrix} +4 \\ +2 \\ +4 \end{bmatrix}$
- $\widehat{X_1}.X_1 = 10$

- $\|X_1\|.\|\widehat{X_1^T}\| = \sqrt{108}$
- $\cos(\widehat{X_1}, X_1) = \frac{[X_1.(\widehat{X_1^T})]}{[\|X_1\|.\|\widehat{X_1^T}\|} = \frac{10}{\sqrt{108}} = 0.96$, this is not a perfect fit for X_1. Identical calculations will generate the same results for the X_3. This can also be recognized by non-orthogonal (correlated) behavior, i.e. $X_1^T.X_3 \neq 0$ (scalar product).

In conclusion, we have shown that if the stored input vectors in the memory are orthogonal (i.e. pairwise), then Cosine = 1. Otherwise, when input vectors are correlated (non-orthogonal), the memory adds crosstalk (noise) to the original input pattern. (Refer to Abdi et. al (1999) for the changes happening in the formula for the recall of the learned pattern.)

$$\widehat{X_k} = W.X_k = \Sigma_i X_i.X_i^T.X_k = \Sigma_i (X_i^T.X_k).X_i = (X_k^T X_k)X_k + \Sigma_{i \neq k}(X_i^T X_k)X_i$$

The noise,$\Sigma_{i \neq k}(X_i^T X_k)X_i$, equals to zero when all the patterns in the input set are mutually orthogonal ($X_i^T.X_1 = 0$). However, if the input patterns are normalized (i.e. the length is 1, $\|X_i\| = 1$), the response of the memory is the input pattern.

3.1.1 Ability of memory to generalize the new patterns

Imagine we are looking at reconstructing vehicle 4 given the vehicle 1 and 3 stored in the memory.

$$\widehat{X_4} = W.X_4 = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$\cos(\widehat{X_4}, X_4) = 1$ (obtained for learned vehicle images) which shows perfect recall.

Reconstructing the fourth vehicle, having vehicle 1 and 2 in the memory, the quality of response is $\cos(\widehat{X_4}, X_4) = 0.33$ (obtained for the learned vehicle images).

As demonstrated, vehicle 4 can be better generalized by the first memory (vehicle image 1 and 3), than the second memory (1 and 2), i.e. $1 > 0.33$.

The above example clearly implies that auto-associative memory is more effective in reconstructing learned patterns than new pattern. This can be interpreted as a sign that the auto-associative memory has the capability to distinguish between new and learned patterns. We say that memory has the ability to recognize learned patterns if,

cos(Reconstructed and Learned patterns) > cos(Reconstructed and New patterns)

We can evaluate the quality of the estimate of a pattern by also computing the reconstruction error for the pattern. The formula for the reconstruction error for the i^{th} pattern is $e_i = 1 - cos(\widehat{X_i}, X_i)$; the difference between a perfect and actual performance of the network.

We can get the global error, by $\mathbf{e} = \Sigma e_i$ (sum of the reconstruction error of all stored

patterns)

According to Abdi (1999), to express the reconstruction error, we can use the reconstruction error function:
$\tau_i = \frac{1}{2}e_i^T.e_i = \frac{1}{2}(X_i - \widehat{X_i})^T(X_i - \widehat{X_i})$. The reconstruction error and reconstruction error function become the same when the input and output vectors are normalized. The global error function is:
$\tau = \Sigma\tau_i = \frac{1}{2}\Sigma(X_i - \widehat{X_i})^T.(X_i - \widehat{X_i})$

3.2 The Widrow-Hoff Learning Rule

So far we have learned that the Hebbian learning rule has its own draw backs, when learned patterns are not orthogonal. The performance can be improved by using the Widrow-Hoff algorithm.

Matrix Format: $W_{[k+1]} = W_{[k]} + \eta(X - W_{[k]}.X)X^T = W_{[k]} + \eta(X - \widehat{X_k})X^T =$

$$W_{[k]}^T + \eta EX^T = W_{[k]}^T + \Delta W_{[k]}$$

E is the error matrix; $\Delta W_{[k]}$ is the connection weight correction matrix at the k^{th} iteration with η as a constant.

In order to maximize the quality of reconstruction of the input patterns, this algorithm adjusts the connection weights between two neurons iteratively. It helps to have a smaller error term for a second presentation of the initial inputs, Paplinski

(2003).

Using the previous example (image retrieval of vehicle 1 and 3), we can see that when Widrow-Hoff rule is being used , non-orthogonality is no longer a problem in oppose to Hebbian rule which the non-orthogonality caused noise in the process of image retrieval.

We assume $\eta = 0.3$, $X_1 = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$, and $X_3 = \begin{bmatrix} +1 \\ -1 \\ +1 \\ -1 \end{bmatrix}$,

hence the matrix X is equal to $X = \begin{bmatrix} +1 & +1 \\ +1 & -1 \\ +1 & +1 \\ +1 & -1 \end{bmatrix}$.

Then, we initialize the weight matrix to zero and put all the connection weights in a $4 * 4$ matrix. The first iteration is as follows:

- $\widehat{X_0} = W_{[0]}.X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
- $\varepsilon_0 = X - \widehat{X_0} = \begin{bmatrix} +1 & +1 \\ +1 & -1 \\ +1 & +1 \\ +1 & -1 \end{bmatrix}$

- $\Delta_{W_{[0]}} = 0.3 \times \varepsilon_0 \times X^T = \begin{bmatrix} 0.6 & 0 & 0.6 & 0 \\ 0 & 0.6 & 0 & 0.6 \\ 0.6 & 0 & 0.6 & 0 \\ 0 & 0.6 & 0 & 0.6 \end{bmatrix}$

The second iteration:

- $W_1 = W_0 + \Delta_{W_0} = \begin{bmatrix} 0.6 & 0 & 0.6 & 0 \\ 0 & 0.6 & 0 & 0.6 \\ 0.6 & 0 & 0.6 & 0 \\ 0 & 0.6 & 0 & 0.6 \end{bmatrix}$
- $\widehat{X_1} = W_1.X = \begin{bmatrix} 1.2 & 1.2 \\ 1.2 & -1.2 \\ 1.2 & 1.2 \\ 1.2 & -1.2 \end{bmatrix}$
- $\varepsilon_1 = X - \widehat{X_1} = \begin{bmatrix} -0.2 & -0.2 \\ -0.2 & 0.2 \\ -0.2 & -0.2 \\ -0.2 & 0.2 \end{bmatrix}$
- $\Delta_{W_{[1]}} = 0.3 \times \varepsilon_1 \times X^T = \begin{bmatrix} -0.12 & 0 & -0.12 & 0 \\ 0 & -0.12 & 0 & -0.12 \\ -0.12 & 0 & -0.12 & 0 \\ 0 & -0.12 & 0 & -0.12 \end{bmatrix}$

The Third iteration:

- $W_{[2]} = W_{[1]} + \Delta_{W_{[1]}} = \begin{bmatrix} 0.48 & 0 & 0.48 & 0 \\ 0 & 0.48 & 0 & 0.48 \\ 0.48 & 0 & 0.48 & 0 \\ 0 & 0.48 & 0 & 0.48 \end{bmatrix}$
- $\widehat{X_2} = W_{[2]}.X = \begin{bmatrix} 0.96 & 0.96 \\ 0.96 & -0.96 \\ 0.96 & 0.96 \\ 0.96 & -0.96 \end{bmatrix}$
- $\varepsilon_2 = X - \widehat{X_2} = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & -0.4 \\ 0.4 & 0.4 \\ 0.4 & -0.4 \end{bmatrix}$
- $\Delta_{W_{[2]}} = 0.3 \times \varepsilon_2 \times X^T = \begin{bmatrix} 0.24 & 0 & 0.24 & 0 \\ 0 & 0.24 & 0 & 0.24 \\ 0.24 & 0 & 0.24 & 0 \\ 0 & 0.24 & 0 & 0.24 \end{bmatrix}$

The last iteration:

- $W_{[22]} = W_{[21]} + \Delta_{W_{[21]}} = \begin{bmatrix} 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \end{bmatrix}$
- $\widehat{X_{22}} = W_{[22]}.X = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}$

- $\varepsilon_{22} = x - \widehat{X_{22}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

- $\Delta_{W_{[22]}} = 0.3 \times \varepsilon_{22} \times X^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

We will continue these steps till we reach to $\Delta_W = 0$. As soon as we reach to this level, we can say that the Widrow-Hoff algorithm converged to the correct answer in $k + 1$ iterations.

Note: the number of iterations needed to reach convergence depends on the initial value chosen for the learning constant, i.e. η. If η is too small, then we need to conduct more iterations to reach the correct solution. However, a very large value could cause the memory to oscillate around the correct answer (Kohzadi, 1995).

Chapter 4

Linear Hetero-associative Memories (Neural Classifiers)

In auto-associative memories an input pattern was associated to itself, but in hetero-associative memories an input layer is connected to an output layer by a set of modifiable connection (i.e. there are always two layers, input and output layers.) The objective in a linear hetero-associative is to train the network to learn mappings between input-output pairs such that the memory produces the appropriate output in response to a given input pattern, Abdi (1999).

All the rules presented for the auto-associative memory can be applied to hetero-associative memories. The following is the general architecture of a hetero-associative memory:

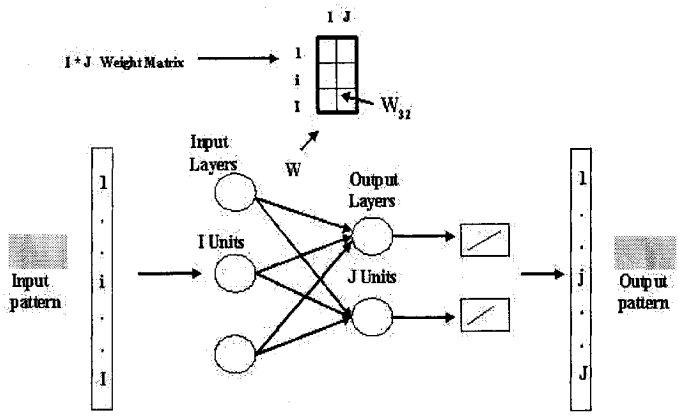


Figure 6: Hetero-associative memories

4.1 Applications of The Hebbian Rule

We can consider the Hebbian learning rule as a supervised learning if during learning we use the correct output. According to Abdi (1999, p.49), *The Hebbian learning rule sets the change of the connection weights to be proportional to the product of the input and the expected output.*

In Figure 6, input neurons is presented by an $I-dimensional$ vector called X_k (X_{I*1} matrix) with its corresponding output which is presented by a $J-dimensional$ vector called O_k (O_{K*1} matrix). Suppose γ represents the proportionality constant, and the connection weight matrix can be represented as $W = \gamma \Sigma_i X_i . O_i^T$ (Hebbian Learning rule) we assume,

- X_1^T = [+1 +1 +1] represents Mazda 323 (the target value of O_1^T = [1 1 -1])
- X_2^T = [-1 -1 -1] represents Toyota 4 Runner truck (the target value of O_2^T = [1

-1 1])

- X_3^T = [-1 +1 -1] represents Dodge pick-up (the target value of O_3^T = [1 − 1 − 1])
- X_4^T = [+1 +1 -1] represents Ford Focus (the target value of O_4^T = [-1 1 -1])

Recall that Hetero-associative memories produce output in response to a given input pattern. To find out the respective connection weight for each input, first we store the image-name pairs in the memory one by one:

First step, **Store the Vehicle-model pairs in the memory, one by one:**

Assuming $\gamma = 1$, we store the first vehicle and its related models in the memory by modifying the weights as following:

$$W_{[1]} = W_{[0]} + X_1 \times O_1^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} \times \begin{bmatrix} +1 & +1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$W_{[2]} = W_{[1]} + X_2 \times O_2^T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \times \begin{bmatrix} +1 & -1 & +1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 2 & -2 \\ 0 & 2 & -2 \\ 0 & 2 & -2 \end{bmatrix}$$

$$W_{[3]} = W_{[2]} + X_3 \times O_3^T = \begin{bmatrix} 0 & 2 & -2 \\ 0 & 2 & -2 \\ 0 & 2 & -2 \end{bmatrix} + \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} \times \begin{bmatrix} +1 & -1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 3 & -1 \\ +1 & 1 & -3 \\ -1 & 3 & -1 \end{bmatrix}$$

$$W_{[4]} = W_{[3]} + X_4 \times O_4^T = \begin{bmatrix} -1 & 3 & -1 \\ +1 & 1 & -3 \\ -1 & 3 & -1 \end{bmatrix} + \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix} \times \begin{bmatrix} -1 & +1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -2 & 4 & -2 \\ 0 & 2 & -4 \\ 0 & 2 & 0 \end{bmatrix}$$

As we explained in the previous section, we can check the quality of the response by the following formula:

$\widehat{O_i} = W^T X_i$ and $cos(\widehat{O_i}, O_i) = [(\widehat{O_i^T}).O_i]/[\|\widehat{O_i^T}\|.\|O_i\|]$,

Hence $cos(\widehat{O_4}, O_4) = [(\widehat{O_4^T}).O_4]/[\|\widehat{O_4^T}\|.\|O_4\|] = 12/(\sqrt{104}\sqrt{3}) = 0.68$

After comparing all of the responses to the vectors presenting the vehicle models (cosine comparison), we find that none of the models recalled the associated input perfectly. According to Abdi (1999), when the input and output vectors are non-orthogonal (i.e. l and r), the memory creates crosstalk and adds noise to the expected patterns:

$\widehat{O_l} = W^T X_l = \Sigma_r O_r X_r^T X_l = \Sigma_r (X_r^T X_l) O_r = (X_l^T X_l) O_l + \Sigma_{r \neq l} (X_r^T X_l) O_r$

The size of the noise $\Sigma_{r \neq l} (X_r^T X_l) O_r$ depends on the correlation between the input patterns (similarity); the larger the similarity between the input and the output patterns, the larger the crosstalk.

4.2 Applications of The Widrow-Hoff Rule

We can improve the performance of the memory by using an error correction rule instead of Hebbian rule. As we discussed before, the Widrow-Hoff learning rule adjusts the connection weights in order to minimize the error sum of squares between the expected response and the response of the memory. By using the error term, the difference between the answer of the memory and the expected output, the weight gets adjusted iteratively. This can be done in two different ways, either in a single stimulus learning mode or in a batch mode. The difference is that in the single mode, we calculate the error after each stimulus, but in the batch mode, we calculate it after the complete set of patterns. The connection weight matrix for a Hetero-associative Memory at iteration n+1 is expressed as (Abdi ,1999):

$W^T_{[K+1]} = W^T_{[K]} + \eta(O - W^T_{[K]}.X)X^T = W^T_{[K]} + \eta(O - \widehat{O_{[K]}})X^T = W^T_{[K]} + \eta EX^T$

$= W^T_{[K]} + \Delta W_{[K]}$

E is the error matrix; $\Delta W_{[K]}$ is the connection weight correction matrix at the K^{th} iteration with η as a constant. Now assume, we want to do an image retrieval from the memory for Car A and Truck B:

We assume input $X_1 = \begin{bmatrix} -1 \\ -1 \\ +1 \\ -1 \end{bmatrix}$ corresponds to the output value of

O[+1 -1 +1] for car A, and the input $x_2 = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}$ corresponds to

the output value of $O[-1-1+1]$ for Truck B, or collectively as

$$X = \begin{bmatrix} -1 & +1 \\ -1 & -1 \\ +1 & -1 \\ -1 & +1 \end{bmatrix}, \text{ and } O = \begin{bmatrix} +1 & -1 \\ -1 & -1 \\ +1 & +1 \end{bmatrix}$$

- $W_{[0]} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- $\widehat{O_0} = W_{[0]}^T.X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} -1 & +1 \\ -1 & -1 \\ +1 & -1 \\ -1 & +1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
- $\varepsilon_0 = O - \widehat{O_0} = \begin{bmatrix} +1 & -1 \\ -1 & -1 \\ +1 & +1 \end{bmatrix}$
- We set $\eta = 0.3$; We could choose any other number between 0 and 1. Choosing a very low or very high number may cause over fitting.
- $\Delta_{W_{[0]}} = 0.3 \times \varepsilon_0 \times X^T = \begin{bmatrix} -0.6 & 0 & 0.6 & 0.6 \\ 0 & 0.6 & 0 & 0 \\ 0 & -0.6 & 0 & 0 \end{bmatrix}$
- $W_{[1]}^T = W_{[0]}^T + \Delta_{W_{[0]}} = \begin{bmatrix} -0.6 & 0 & 0.6 & 0.6 \\ 0 & 0.6 & 0 & 0 \\ 0 & -0.6 & 0 & 0 \end{bmatrix}$

The second iteration:

- $W_{[1]} = \begin{bmatrix} -0.6 & 0 & 0 \\ 0 & 0.6 & -0.6 \\ 0.6 & 0 & 0 \\ 0.6 & 0 & 0 \end{bmatrix}$
- $\widehat{O_1} = W_{[1]}^T.X = \begin{bmatrix} -0.6 & 0 & 0.6 & 0.6 \\ 0 & 0.6 & 0 & 0 \\ 0 & -0.6 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} -1 & +1 \\ -1 & -1 \\ +1 & -1 \\ -1 & +1 \end{bmatrix} = \begin{bmatrix} 1.8 & -1.8 \\ -0.6 & -0.6 \\ 0.6 & 0.6 \end{bmatrix}$
- $\varepsilon_1 = O - \widehat{O_1} = \begin{bmatrix} -0.8 & 0.8 \\ -0.4 & -0.4 \\ 0.4 & 0.4 \end{bmatrix}$
- $\Delta_{W_{[1]}} = 0.3 \times \varepsilon_1 \times X^T = \begin{bmatrix} 0.48 & 0 & -0.48 & 0.48 \\ 0 & 0.24 & 0 & 0 \\ 0 & -0.24 & 0 & 0 \end{bmatrix}$
- $W_{[1]}^T = W_{[0]}^T + \Delta_{W_{[0]}} = \begin{bmatrix} -0.12 & 0 & 0.12 & -0.12 \\ 0 & 0.84 & 0 & 0 \\ 0 & -0.84 & 0 & 0 \end{bmatrix}$

We continue until we get $\Delta_{W_{[K]}} = 0$

Chapter 5

Backpropagation (BP) Algorithm

What we have learned so far helps us to solve *linear*[1] problems with Neural Networking models. In this section, we learn how backpropagation algorithm can help us use neural networks models in nonlinear problems. BP networks are made of multi-layer networks consisting of nonlinear units, where only the hidden layers (cells) need to be nonlinear units.

Backpropagation uses supervised learning like perceptrons and linear hetero-associator. The difference between backpropagation and other architects is that the BP uses the error made by the network as a basis to achieve the correct connection weights. The error in the network first gets converted to error signal by the output unit and then

[1] *"function $Z(B) = Z_0 + Z_iB(i)$ which is linear in the parameters B(1), B(2),...,B(n) to be one where each term in the function has at most one parameter, and each parameter appears only to the first degree, that is, to the first power. A linear equation results by setting Z(B) to zero. Bates and Watts (1986) consider the term "nonlinear" to imply that the gradient of the modeling function is not constant. We will consider that the parameters B are nonlinear if the equations which must be solved or functions which must be minimized in order to estimate their values cannot be reduced to a set of simultaneous linear equations "(Nash, 1987.)*

back propagated (backward move) to the hidden layers. The hidden layers use the error signal and then estimate their errors.

Figure 7, all connection weights proportionally get automatically updated after the overall estimated hidden layers.

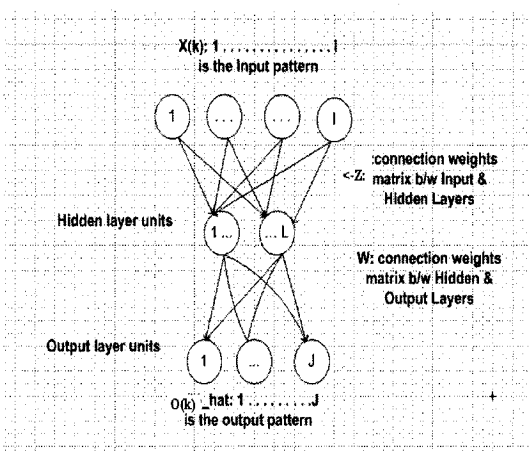


Figure 7: Back-propagation network

- x_k, $(k = 1, .., i, .., I)$, is the input units presented by a $I * 1$ matrix
- h_k, *hidden* $(k = 1, .., l, ., L)$, is the hidden units presented by a $L * 1$ matrix
- $\widehat{o}_k$, $(k = 1, .., j, ., J)$, is the predicted output/response unit stimulus presented by a $J * 1$ matrix
- Z_{I*L} is the connection weight matrix between the input and hidden layers
- W_{L*J} is the connection weight matrix between the hidden and output layers

- $f(x) = logist(x) = 1 \div (1 + \exp^{-x})$
  or $f(x) = \tanh(x) = (\exp^{x} - \exp^{-x}) \div (\exp^{x} + \exp^{-x})$ are the nonlinear transformation functions which transforms the activation level into a response. Note: $\tanh(x)$ or the hyperbolic tangent is usually used when the output response range is between -1 and $+1$.

When Widrow-Hoff learning rule is used, non-linear units like linear ones learn by iteratively correcting their connection weights. For nonlinear problems, the connection weights correction is proportional to the error, the value of inputs (i.e. linear) , and when the output of the cell is near asymptotic values (i.e. the asymptotic values are 0 and 1 for the logistic function), the degree of correction must be very small. The degree of correction is very large when the output of the cell is in the middle of the logistic, i.e. 0.5 for the logistic.

To make the connection weight correction proportional to the *rate of change* of the transfer function, it is much convenient to make it proportional to the derivatives of the function. we calculate the correction for a connection weight from input cell i to hidden cell l:

$\Delta_{Z_{i,l}}$ = (activation of cell i or x_i) $*$ (error signal past to the hidden cell "l" from the output cell) $= x_i * [f(\acute{a}_l) \times \hat{e}_l]$

so the derivative for the above transfer functions shall be as follows:

$f'(x) = \frac{dlogist(x)}{dx} = logist(x)[1 - logist(x)] = f(x)[1 - f(x)]$

$f'(x) = \frac{dtanh(x)}{dx} = \frac{4}{(exp^x + exp^{-x})^2}$

Now, we look at an example of BP algorithm with non-threshold cells from Abdi (1999, p.75-83).

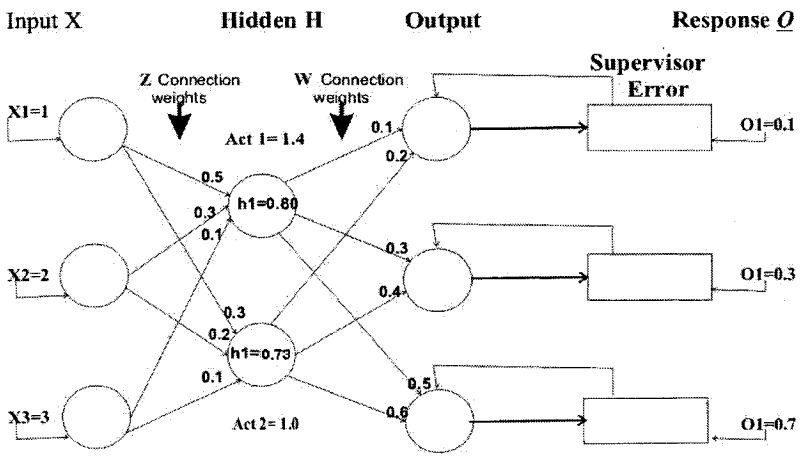
In the first part, we use the input layer and generate a normal feed-forward flow of the activation to the output layer passed through the hidden layer. We assume:

- $Z_{[K+1]} = Z_{[K]} + \eta h_l \delta^T_{hidden,l} = Z_{[K]} + \Delta_{Z_{[K]}}$;K is the iteration number
- There are three input cells $X_i = x_1, x_2, x_3$ or $X = [1\ 2\ 3]^T$.
- There are two hidden cells and the Z_{3*2} is the connection weight matrix between the input and hidden layers, $Z_{3*2} = \begin{bmatrix} .5 & .3 \\ .3 & .2 \\ .1 & .1 \end{bmatrix}$
- The hidden cell response is calculated by $Hidden_{[K]} = f(Z^T_{3*2} X_{[K]})$
- There are three output cells.
- The W_{2*3} is the connection weight matrix between the hidden and output layers. $W_{2*3} = \begin{bmatrix} .1 & .3 & .5 \\ .2 & .4 & .6 \end{bmatrix}$
- We consider the output response vector as $\hat{O_{[K]}} = f(W^T_{2*3} Hidden_{[K]})$, and $\eta = 1$.

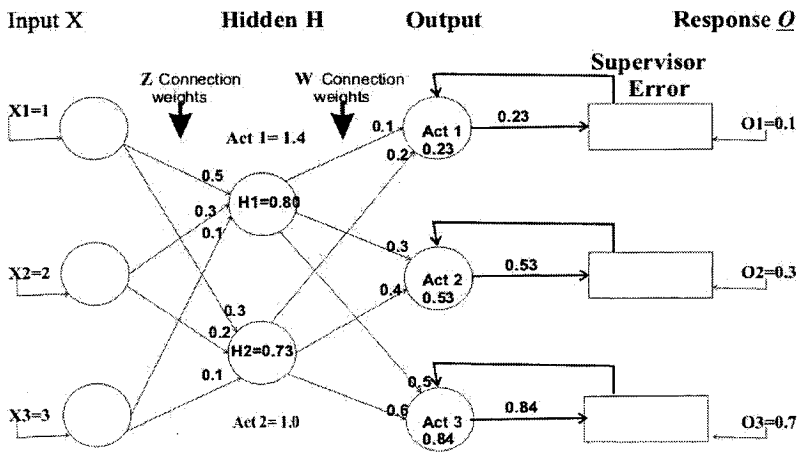
We start the process by calculating the level of activation of the hidden units, (Note: All the numbers have been rounded)

- Activation Level= $\begin{bmatrix} .5 & .3 & .1 \\ .3 & .2 & .1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1.4 \\ 1.0 \end{bmatrix}$

- Hidden $= f(\begin{bmatrix} 1.4 \\ 1.0 \end{bmatrix}) = \begin{bmatrix} 1 \div (1 + \exp^{-1.4}) \\ 1 \div (1 + \exp^{-1.0}) \end{bmatrix} = \begin{bmatrix} 0.80 \\ 0.73 \end{bmatrix}$

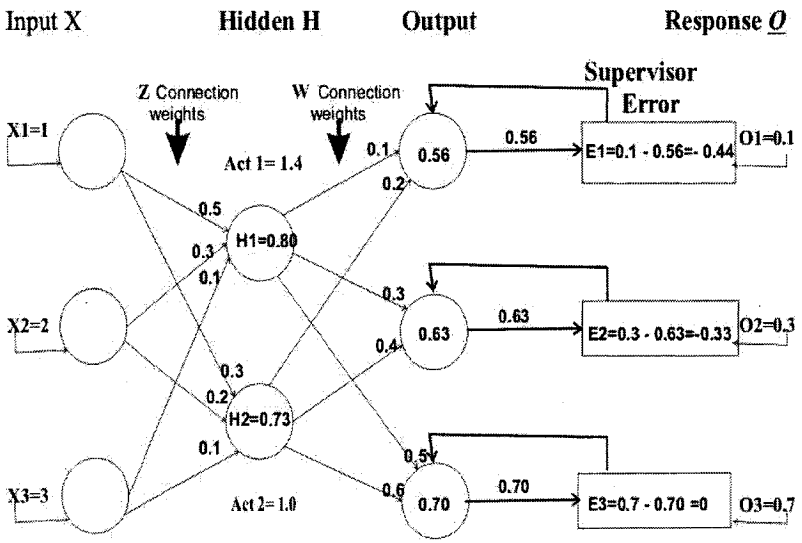


- Output Activation Level$= \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix} \times \begin{bmatrix} 0.80 \\ 0.73 \end{bmatrix} = \begin{bmatrix} 0.23 \\ 0.53 \\ 0.84 \end{bmatrix}$



- $\widehat{O} = f(\begin{bmatrix} 0.23 \\ 0.53 \\ 0.84 \end{bmatrix}) = \begin{bmatrix} 1 \div (1 + \exp^{-0.23}) \\ 1 \div (1 + \exp^{-0.53}) \\ 1 \div (1 + \exp^{-0.84}) \end{bmatrix} = \begin{bmatrix} 0.56 \\ 0.63 \\ 0.70 \end{bmatrix}$

- $Error = O - \widehat{O} = \begin{bmatrix} 0.1 \\ 0.3 \\ 0.7 \end{bmatrix} - \begin{bmatrix} 0.56 \\ 0.63 \\ 0.70 \end{bmatrix} = \begin{bmatrix} -0.44 \\ -0.33 \\ 0.0 \end{bmatrix}$



After this process, the network can start the learning process by first calculating the error signal. To compute error signal, we must first compute the derivative of the output unit responses (i.e. $\bigotimes$ represents elementwise product of vectors.)

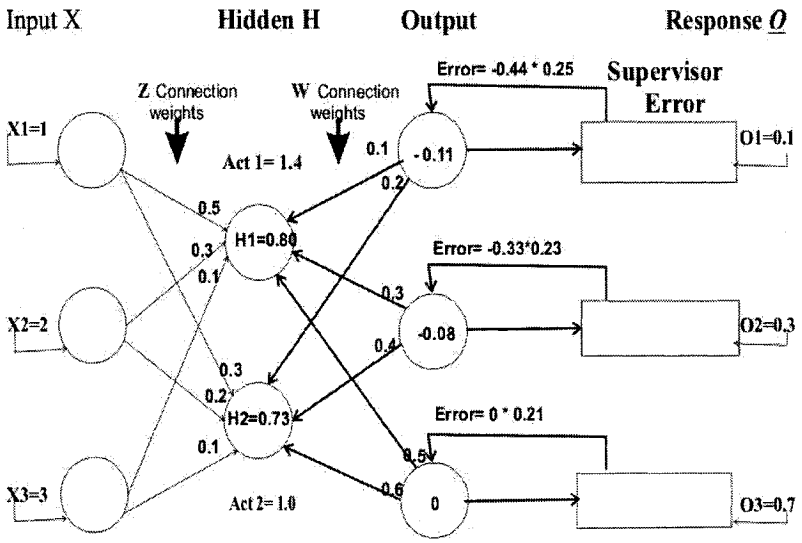
- $\acute{f}(a) = \widehat{O} \bigotimes (1 - \widehat{O}) =$

$$= \begin{bmatrix} 0.56 \\ 0.63 \\ 0.7 \end{bmatrix} \bigotimes \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.56 \\ 0.63 \\ 0.70 \end{bmatrix} \right) = \begin{bmatrix} 0.56 \\ 0.63 \\ 0.70 \end{bmatrix} \bigotimes \begin{bmatrix} 0.44 \\ 0.37 \\ 0.30 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.23 \\ 0.21 \end{bmatrix}$$

Next step is the backpropagation process. It starts by computing the output errors applied to the hidden layer.

- $\delta_{output} = \acute{f}(a) \bigotimes Error = \widehat{O} \bigotimes (1 - \widehat{O}) \bigotimes (O - \widehat{O}) =$

$$= \begin{bmatrix} 0.25 \\ 0.23 \\ 0.21 \end{bmatrix} \bigotimes \begin{bmatrix} -0.44 \\ -0.33 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -0.11 \\ -0.08 \\ 0.0 \end{bmatrix}$$



- $\widehat{error} = W \times \delta_{output} = \acute{f}(a) \bigotimes Error = \widehat{O} \bigotimes (1 - \widehat{O}) \bigotimes (O - \widehat{O}) =$

$$= \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{bmatrix} \times \begin{bmatrix} -0.11 \\ -0.08 \\ 0.00 \end{bmatrix} = \begin{bmatrix} -0.03 \\ -0.05 \end{bmatrix}$$

The hidden layer's error signal is calculated in the same manner. Note: When the error is supplied by the supervisor, we replace it with the estimation of the hidden units' error.

- $\acute{f}(b) = Hidden \bigotimes (1 - Hidden) = \begin{bmatrix} 0.80 \\ 0.73 \end{bmatrix} \bigotimes (\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.80 \\ 0.73 \end{bmatrix}) = \begin{bmatrix} 0.80 \\ 0.73 \end{bmatrix} \bigotimes \begin{bmatrix} 0.198 \\ 0.269 \end{bmatrix} =$

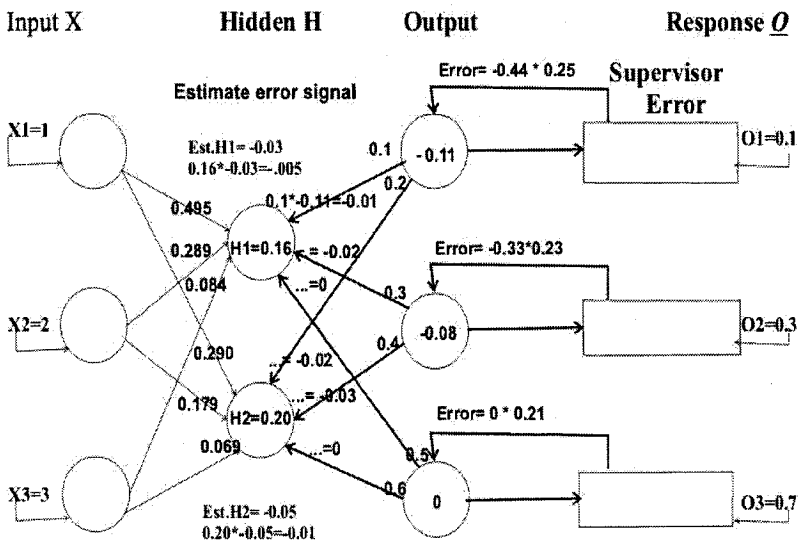
$$\begin{bmatrix} 0.16 \\ 0.20 \end{bmatrix}$$

- $\delta_{hidden} = \acute{f}(b) \bigotimes error = hidden \bigotimes (1 - hidden) \bigotimes (W \times \delta_{output}) = \begin{bmatrix} 0.16 \\ 0.20 \end{bmatrix} \bigotimes \begin{bmatrix} -0.03 \\ -0.05 \end{bmatrix} = \begin{bmatrix} -0.005 \\ -0.010 \end{bmatrix}$

- $Z_{[K+1]} + \Delta_Z = Z + \eta x \delta^T_{hidden} = \begin{bmatrix} 0.5 & 0.3 \\ 0.3 & 0.2 \\ 0.1 & 0.1 \end{bmatrix} + \left(\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \times \begin{bmatrix} -0.005 & -0.010 \end{bmatrix} \right) =$

$$\begin{bmatrix} 0.5 & 0.3 \\ 0.3 & 0.2 \\ 0.1 & 0.1 \end{bmatrix} + \begin{bmatrix} -0.005 & -0.010 \\ -0.011 & -0.021 \\ -0.016 & -0.031 \end{bmatrix} = \begin{bmatrix} 0.49 & 0.29 \\ 0.29 & 0.18 \\ 0.08 & 0.07 \end{bmatrix}$$

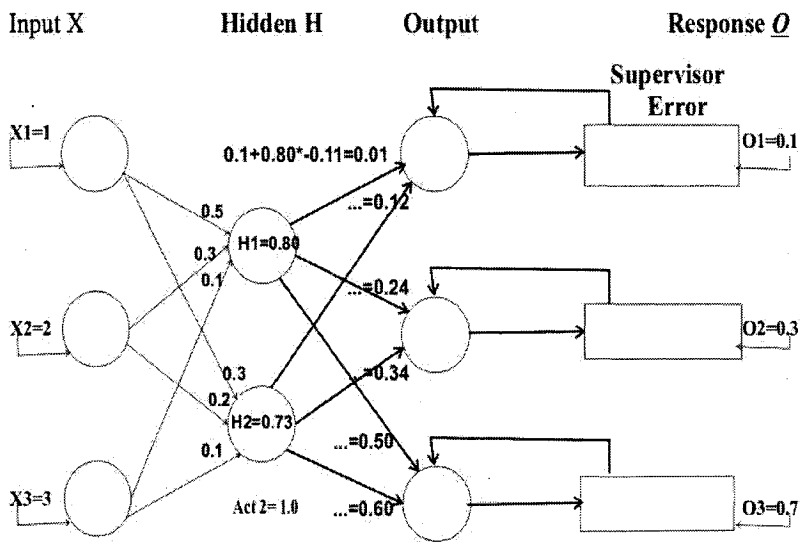
(Note: As we previously assumed, η is equal to 1 in this example.)



Now, we can update our output weights ($W_{[K+1]} = W_{[K]} + \Delta_W$), and start the iteration again.(Note: the redirection of the arrows from the above figure.)

- $\Delta_w = \eta \times hidden \delta^T_{output} = 1 \times \begin{bmatrix} 0.80 \\ 0.73 \end{bmatrix} \times \begin{bmatrix} -0.11 & -0.08 & 0.0003 \end{bmatrix} = \begin{bmatrix} -0.09 & -0.06 & 0.0(\\ -0.08 & -0.06 & 0.0(\end{bmatrix}$

- $W_{[k+1]} = W_{[k]} + \Delta_W = \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{bmatrix} + \begin{bmatrix} -0.09 & -0.06 & 0.0003 \\ -0.08 & -0.06 & 0.0002 \end{bmatrix} = \begin{bmatrix} 0.01 & 0.24 & 0.5 \\ 0.12 & 0.34 & 0.6 \end{bmatrix}$.



After we got the new weight matrix, we continue to repeat the same steps until we get the desired perfect output.

Neural Networking Library of *R*

The neural networking package of *R* which is called *nnet Library* is based on the single-hidden layer, feed-forward Backpropagation model which is the most commonly used amongst researchers. *nnet Library* of *R* conducts the exact same BP process as shown in this chapter to generate the desired forecasting outputs. In the next chapter, we are going to use the neural networking library of *R* as well as other useful libraries to conduct a comparative analysis between the results obtained using neural networks and ARIMA models.

Chapter 6

Time Series Forecasting using Neural Networks

In this section, we look at time series forecasting using neural networks. A comparison will be made between the final output of each model and another relevant forecasting method, i.e. ARIMA. In order to find an effective neural network forecasting model, we need to take 3 steps. First, we must detect the appropriate number of input patterns. Next, we estimate the number of hidden nodes in the hidden layer. Finally, we must compare the result of our networks with other forecasting models to find out the validity of our model.

6.1 Time Series Analysis

The network that we are working on in our time series analysis is a feedforward neural networks which is composed of input neurons, hidden nodes (one hidden layer), and one output layer. As we mentioned in the previous chapters, we use adaptive learning algorithms such as Widrow-Hoff back-propagation algorithm to adjust weights until we reach our desired relationship between the input and output layer of the networks.

(Note: Because of the intensity of the calculations (very time consuming), we will conduct error minimization using R statistical software.)

According to *Lin et al.* (1995), we can describe the neural network forecaster as:

$$X_{n+1} = NN(X_n, X_{n-1},...X_{n-k}, e_n, e_{n-1},..., e_{n-l})$$

To determine the performance of the network forecaster we need to know the input data, original observations (X), and residuals (e).

Detection of input patterns may be done by Autocorrelation Function (ACF), and Partial Autocorrelation Function (PACF) analysis as a rule of thumb introduced by *Lin* (1995). The main thing is to ensure that the series has been rendered stationnary. Hence, if we detect any particular trend in the data, we use differencing to remove the trend. We continue this untill the series become stationnary. In one-step forecasting, ACF is used to determine the lags of residuals for short term forecasting (Lin, 1995). The next step to detect the input patterns is to calculate PACF. The result of PACF analysis reveals how X_i is autocorrelated to X_{i+k}. We choose those partial autocorrelation coefficients which are considerably different than the other coefficients. The number of inputs needed for the network forecaster is basically the largest lag between any two of the coefficients.

According to Baum E. B. and Haussler D.(1983), a rule of thumb to determine the number of hidden neurons is by the following formula:

$$HiddenNeurons \leq \frac{(Num_{TrainingExample}) \times (Error_{Tolerance})}{(Num_{datapoints}) + (Num_{outputneurons})}$$

- $Num_{TrainingExample}$: Number of Observations × Number of data points
- $Error_{Tolerance}$: Error Tolerance
- $Num_{datapoints}$: Number of data points per training example (i.e. input numbers)
- $Num_{outputneurons}$: Number of output neurons

When we look at one-step ahead forecasting, Figure 8, residuals also considered to be a part of the input values. However, when we look at the multi-step ahead forecasting (long-term prediction), the actual future data may not exist, hence residuals are no longer considered to be available. So, we remove the *feedback* loop from the output, as shown in the following graph.

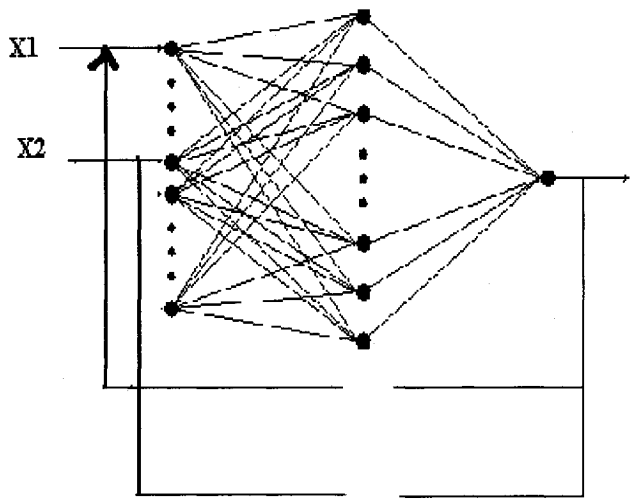


Figure 8: neural network forecaster for one-step ahead forecasting (Lin, 1995).

To give a simple example of using the *Baum-Hassler rule*, we look at the AirPassengers example from the following section.

- Number of Observations: 144
- suppose the $Error_{Tolerance}$: 0.03
- $Num_{datapoints}$: we try different values to fit the best network, refer to Empirical Analysis Section.
- $Num_{outputneurons}$: 1

$HiddenNeurons \leq \frac{(144)\times Num_{datapoints}\times(0.03)}{(Num_{datapoints})+1}$,we have
$Hidden \leq \frac{144\times Num_{datapoints}\times(0.03)}{(Num_{datapoints}+1)}$

So the number of neurons in hidden layers must be 2,3,4 at most for the input numbers (data points) of 2,10,20 respectively. Here is the number of hidden neurons as a starting point to build different NN networking models. As we are going to show in the following sections, we also try other numbers for input and hidden neurons to justify our selected networks. The results have shown that in order to find both the proper number of input and hidden neurons, we need to use trial and error method. Previously mentioned rules work only as a starting point for us.

We conduct time series prediction by feeding our desired model a series of input data X_1, ..., X_n, and forecast $X_{n+1}...X_{n+k}$ (k a positive integer). In this section, the main comparison has been done between ARIMA and neural networks models. However, we have also looked at two simpler forecasting models, Moving Average and Holt-Winter's. Although Holt-Winter's method showed some good results in one case (i.e.

Canadian Road Fatalities data set), but it failed in other trials. Hence, we concentrated on ARIMA and neural networks as two main comparative methods. Since *R* is our main focus, we use the ready packages and functions in the software. For this part, we need to load "Stat" library or "ts" library (in older versions of *R*.) Note: we also need to load the *nnet*, *base*, *stats*, *MASS*, *ade4*, *datasets*, *splines*, *tseries*, and *forecast* libraries from *R* to conduct the necessary analysis.

In the following examples, we make our comparison between ARIMA and neural networking models by dividing our analysis to two parts of forecasting Seasonal and Non-seasonal data.

6.1.1 Non-Seasonal Data Set

In forecasting analysis of non-seasonal data , we look at 13 data sets in total. In this section, we look at only 1 set of non-seasonal data, but the remaining 12 sets will be discussed in the next chapter.

Looking at *USeconomic* data set, Luetkepohl, H. (1991),available in the *R* directory of *tseries* library, we get *M1*, the seasonally adjusted real money, and *GNP* in 1982 U.S. Dollars, as well as, *rs* which is the discount rate on 91-Day treasury bills, and *r1* which is the yield on the long-term treasury bonds. *USeconomic* contains the logarithm of *M1*,and *GNP*, as well as the real values of *rs* and *r1*. Our focus, in the next section, will be to find an appropriate forecasting model for *rs*, *Discount rate on 91-day treasury bills.*

6.1.2 ARIMA Modeling (Non-Seasonal)

In ARIMA modeling, we look at both AR(p), autoregressive process of order p, as well as MA(q), moving average process of order q. An ARIMA (p,d,q) process which stands for the integration of AR and MA is an ARIMA model in which the d^{th} order of differencing is an ARIMA(p,q) process, Venables, W. N. and Ripley (2000). We can define it as

$X_n = AR(p) + MA(q)$ which can also be defined as

$X_n = \sum_1^p \alpha_i X_{n-i} + \sum_0^q b_k \varepsilon_{n-k}$

AR(p) can also be expressed as:

$X_n = \mu + \alpha_1(X_{n-1} - \mu) + ... + \alpha_p(X_{n-p} - \mu) + \varepsilon_n, for n = 1, ...$

MA(p) can also be expressed as:

$X_n = \mu_n + \varepsilon_n + b_1\varepsilon_{n-1} + ... + b_q\varepsilon_{n-q}$

Data Description:

Figure 9 shows an unsmoothed scatterplot of the discount rate on 91-day treasury bills against time, *rs*. As you can see, there is an upward movement at the beginning of the series. Usually, time-series analysis divulges a sequential dependence, John Maindonald and John Braun (2003). This dependence creates a challenging task of finding the right model.

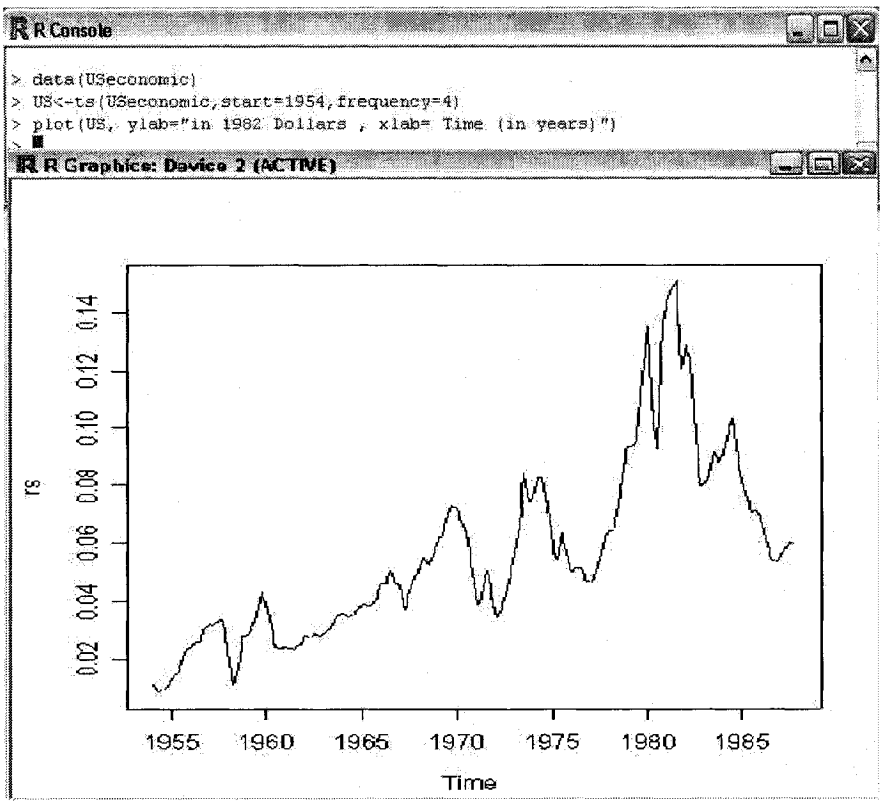


Figure 9: Scatter plot of *the discount rate on 91-day treasury bills against tim, rs.*

Besides the scatterplot, we can also look at the lag plots, Figure 10. lag 1 plot indicates that we are plotting x_k against x_{k-1} for all $k = 2, 3, ..., n$. Similarly, lag 2 plot shows x_k against x_{k-2} for $k = 3, ..., n$, and so on.

Figure 11 shows that the first 4 lag plots for *rs* data in *tseries library* of *R.* "If we look at the lag 1 plot of *rs*, the scatter of points around a straight line suggests that the dependence between consecutive observations is linear", John Maindonald (2003). According to the same graph, as the lag number increases, we can observe a more scattered plot. Hence, the lag 4, successfully suggests that the separated points are

less dependent.

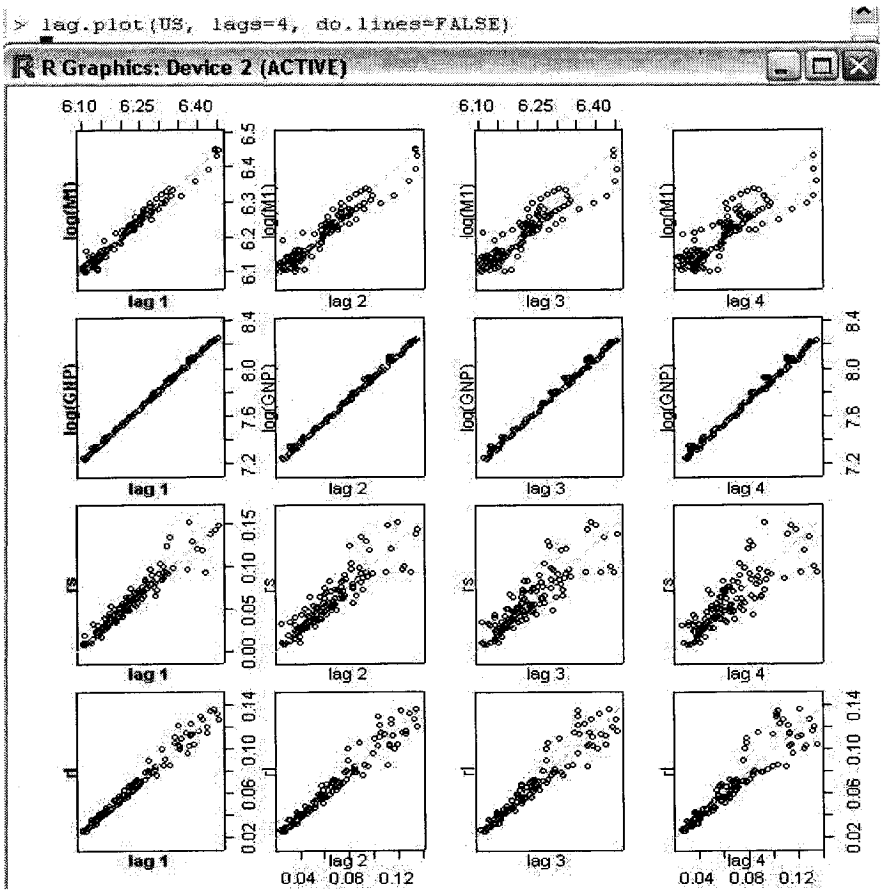


Figure 10: lag plot for all the variables of *USeconomic* data set. ($R_{GUI} - output$)

If we look at the different plots (i.e. Scatter, Lag, and ACF plots) of *rs* data set separately, Figure 12, we can observe that there is a positive correlation (autocorrelation), according to the lag plot.

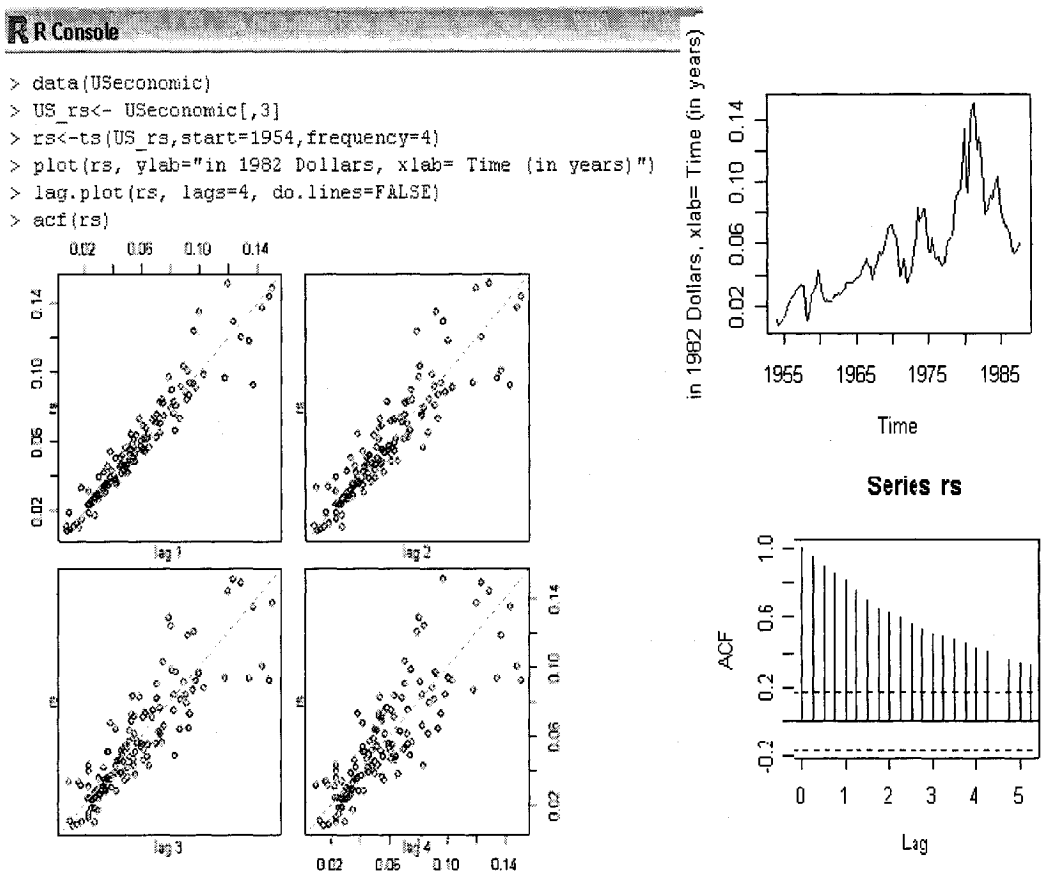


Figure 11: Scatterplot, lag plot, and ACF plot of *rs* ($R_{GUI} - output$)

6.1.3 Data Analysis

rs is a quarterly data set. Since it is quarterly, the seasonal length is fixed at 4. According to ACF, Figure 12, the data declines rather slowly which suggests that our data has a trend. Since there is a clear trend in our data, *rs*, the series is not stationary, and we must eliminate the trend before continuing the ARIMA modeling process.

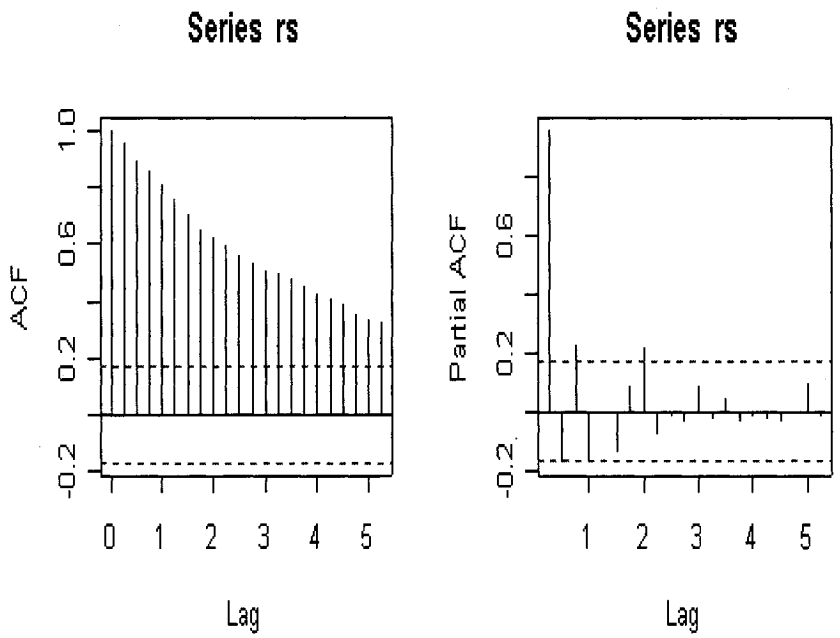


Figure 12: The ACF, and PACF graphs of *rs* ($R_{GUI} - output$)

Standard practice to stationnarize a series with a trend , Figure 13, is to take the series first differences (Nash, 2001.)

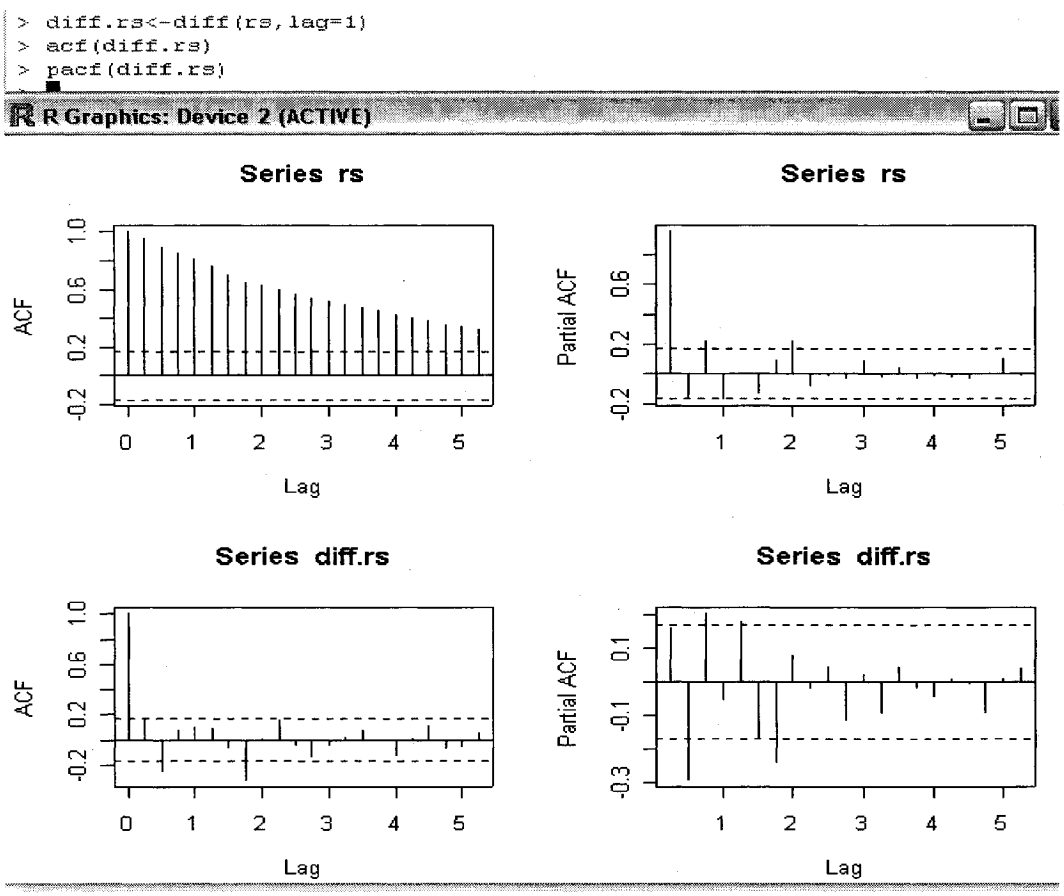


Figure 13: Non-seasonal first difference of *rs* ($R_{GUI} - output$)

ar function in *R* is a wrapper for the autocorrelation estimate function, *yw*, (*ar.yw*), and other functions such as *ar.burg, ar.ols, and ar.mle.* In *R*, order selection is conducted by AIC (Akaike's Information Criterion). AIC is designed to decline by the increase of the estimated predictive power, John Maindonald (2003).

- In ar.mle function, AIC is calculated such that the estimate of variance were the Maximum Likelihood Estimation (MLE), disregarding the determinant term from the LH (likelihood).

- In autocorrelation estimate function, ar.yw, the variance matrix of the innovations is calculated using the fitted coefficients and the auto-covariance of input values, x.

Going back to AR(p) formula, we first try to estimate α. We can either use the maximum liklihood estimator, or the autocorrelation at lag 1, Figure 14.

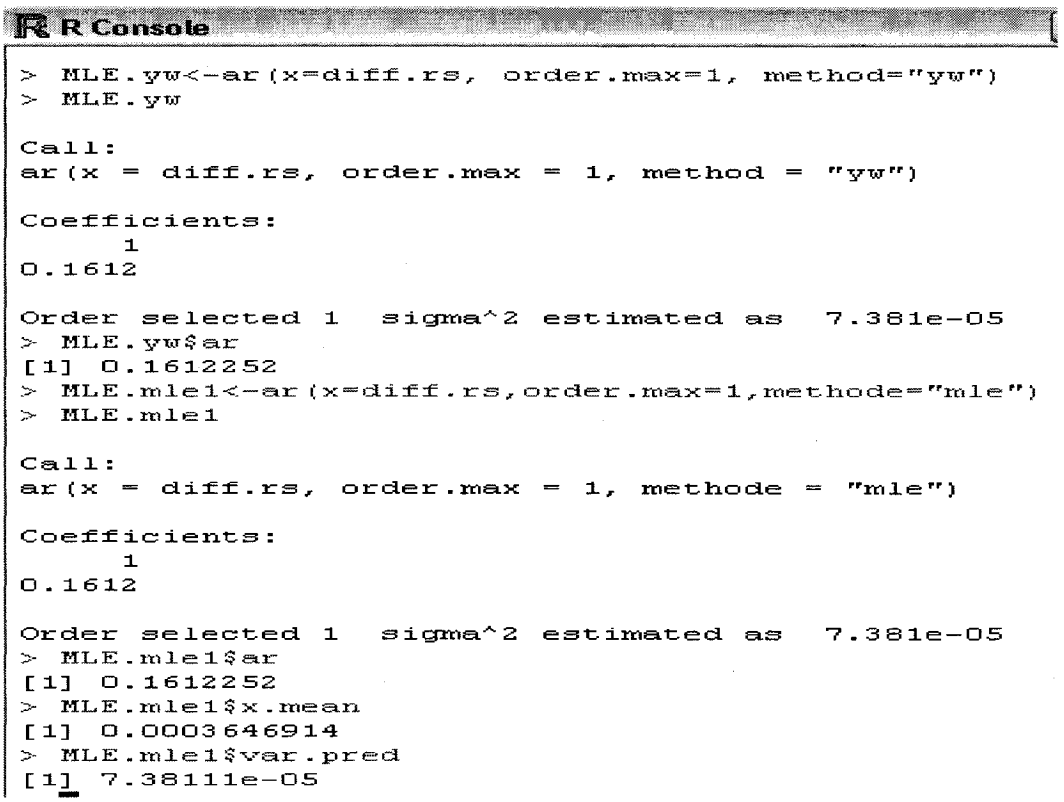


Figure 14: Order selection ($R_{GUI} - output$)

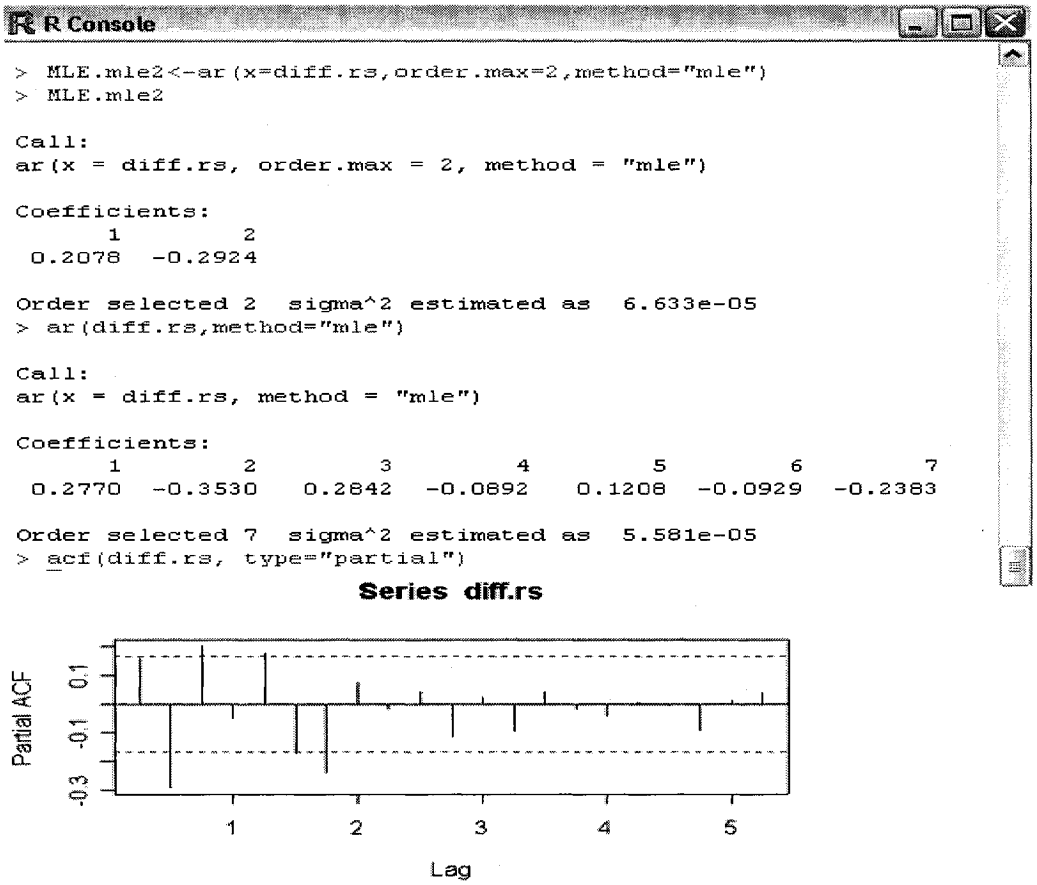


Figure 15: Order Selection and Partial ACF of *diff.rs* (R_{GUI} – *output*)

As shown above, we first fit a AR(1) process, after removing the mean which gives us:

$$X_n = 0.1612X_{n-1} + \varepsilon_n \text{ with } \sigma^2 = (7.38111)10^{-05}.$$

In Figure 14, we computed the autocorrelation estimate (i.e. MLE.yw using *ar* function) considering AR(1), i.e. order.max=1. Then, we conducted the autocorrelation

estimate of alpha by *MLE.yw* §*ar* function. In Figure 15, *R* has computed the Maximum Likelihood estimate (MLE.mle).

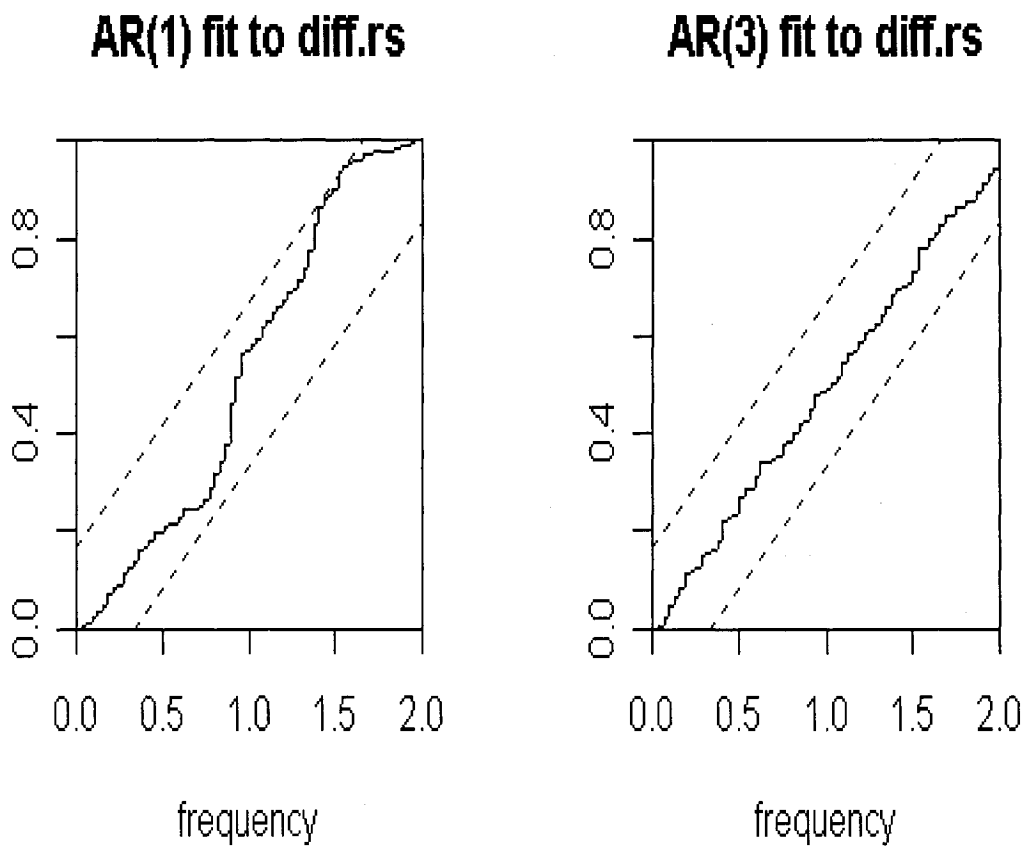


Figure 16: Cumulative Peridogram Plots for residuals of AR models fitted to *diff.rs* ($R_{GUI} - output$)

6.1.4 Finding The Right ARIMA Model For *rs* Data Set

Our model selection criteria is AIC (in R), but we also may look at BIC, and MSE values.

- AIC (Akaike's Information Criterion) which can be calculated as
  AIC(P) $= -2ln(L) + 2P$; where P is the number of parameters in the model and L is the likelihood function.

- BIC (Schwartz's Bayesian Criterion) which is quite similar to AIC:
  BIC(P) $= -2ln(L) + P.ln(n)$; n is the number of observations

We can try fitting different AR's and compare their AIC, σ^2, BIC,and MSE values. To do this, we can use *arima* function of R, which fits the model by ML (Maximum Likelihood) and does not include a mean.

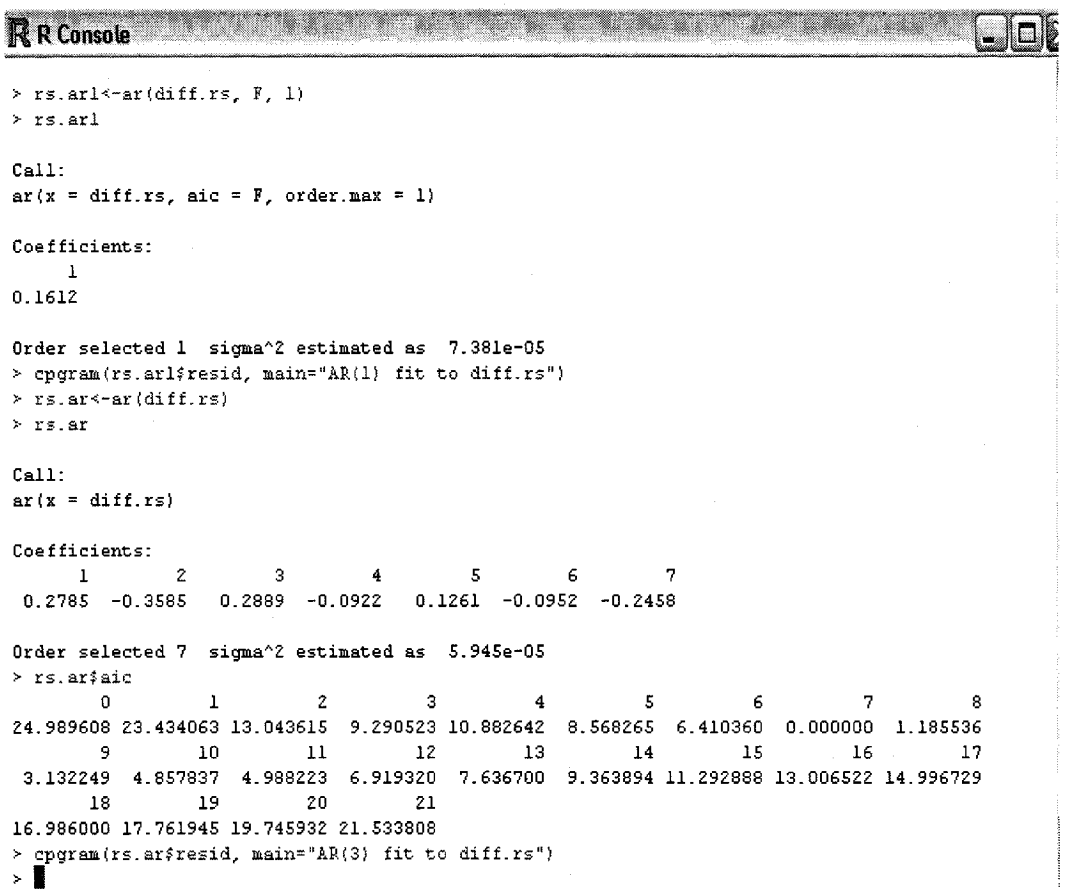


Figure 17: AR(1) and AR(3) models fitted to *diff.rs* ($R_{GUI} - output$)

According to the Figure 13, the ACF of the differenced series has declined sharply. If we look at the ACF diagram, we can see that the autocorrelations are not different from zero after lag 1 or 2 which is the indication of a MA(1) or MA(2) model that would best describe the variable. Similarly, after examining the PACF of series *rs*, we can observe that the partial autocorrelations are not different from zero after lag 1 or 2. This would indicate that an AR(1) or AR(2) model would best describe this variable. Next, we try different ARIMA models, and base our decision on the AIC criterion to find the best fit.

According to the results shown by the *R* GUI (Figure 18-21), between all the models we have tried so far, the ARIMA$(1,0,0)$ looks the best choice because of the lowest AIC value, BIC, and MSE (Mean Square Error) value. Figure 19 indicates that AR(1) has removed all the correlations, according to the *P-value* for the Box-Pierce (1970), i.e. P-value for Goodness of Fit Statistics.

Comparing AR(1,0,0), and the best ARIMA fit obtained by *best.arima* function of *R*, Figure 22, AR(1,0,0) seems a more reasonable model for our forecast.

```
> rs.arima1<-arima(rs,order=c(1,0,0),n.cond=3)
> rs.arima1
Series: rs
ARIMA(1,0,0)(0,0,0)[4] model

Coefficients:
         ar1  intercept
      0.9613     0.0506
s.e.  0.0218     0.0164

sigma^2 estimated as 7.331e-05:  log likelihood = 453.15,  aic = -900.31
> tsdiag(rs.arima1)
>
```

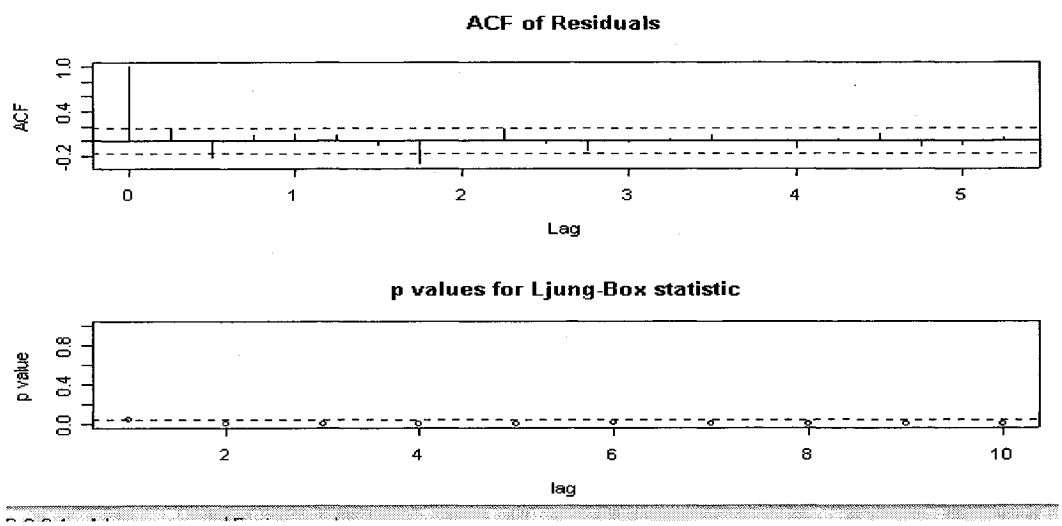


Figure 18: ARIMA(1,0,0) Model ($R_{GUI} - output$)

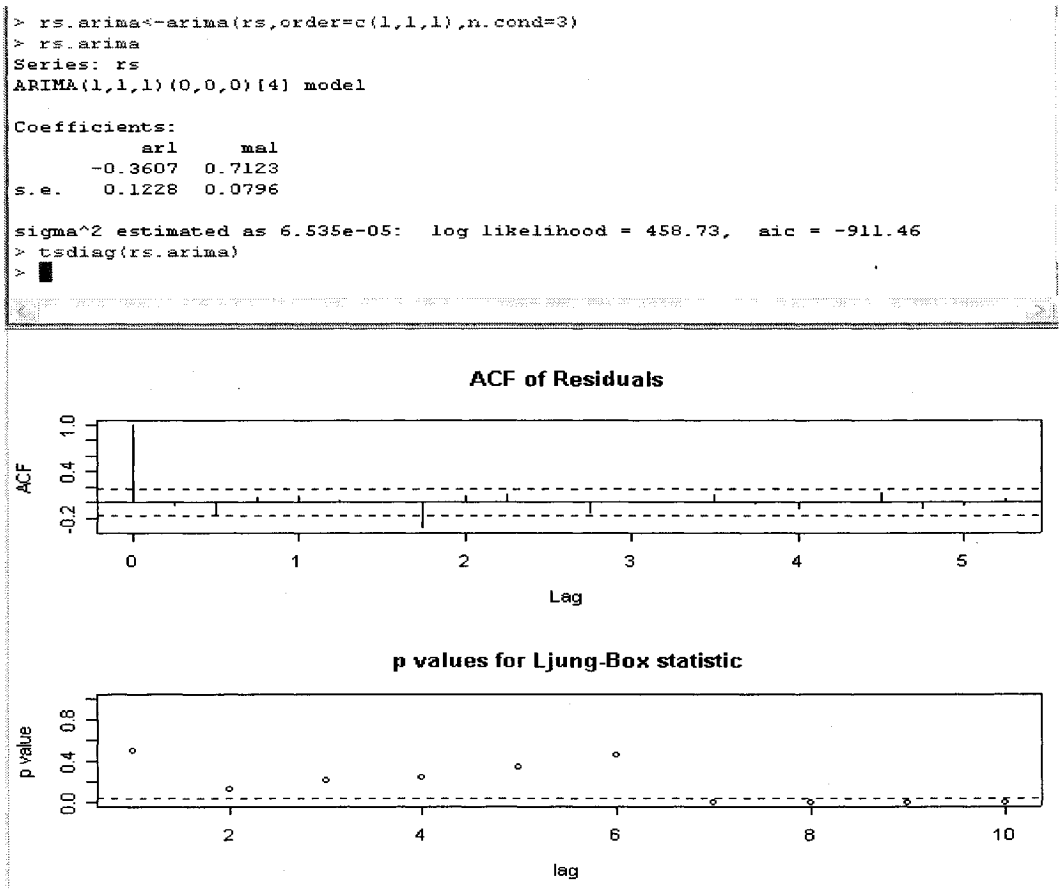


Figure 19: ARIMA(1,1,1) Model ($R_{GUI} - output$)

```
> rs.arima<-arima(rs,order=c(2,1,1),n.cond=3)
> rs.arima
Series: rs
ARIMA(2,1,1)(0,0,0)[4] model

Coefficients:
          ar1      ar2      ma1
      -0.2850  -0.2282  0.5773
s.e.   0.1347   0.0931  0.1178

sigma^2 estimated as 6.275e-05:  log likelihood = 461.41,  aic = -914.82
> tsdiag(rs.arima)
>
```

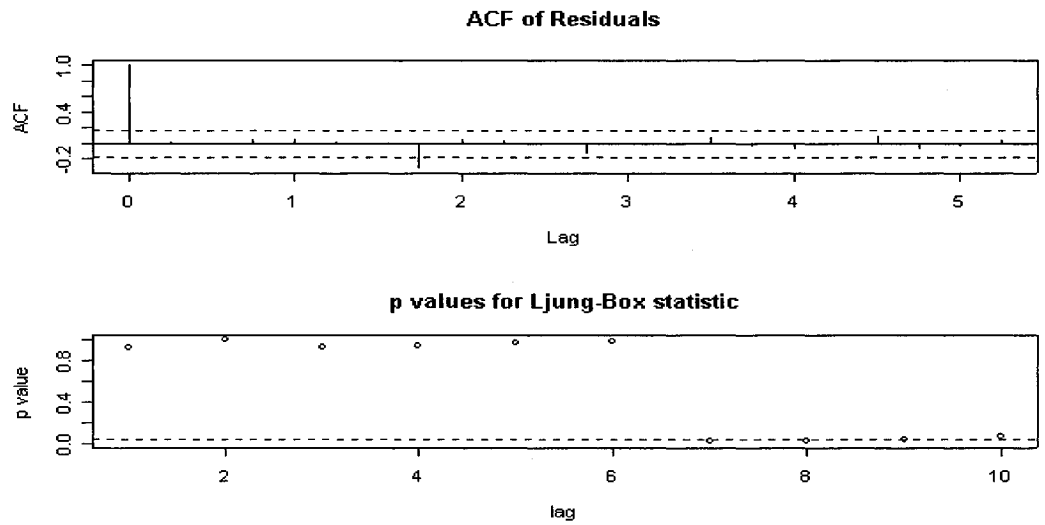


Figure 20: ARIMA(2,1,1) Model ($R_{GUI} - output$)

```
> rs.arima<-arima(rs,order=c(2,0,0),n.cond=3)
> rs.arima
Series: rs
ARIMA(2,0,0)(0,0,0)[4] model

Coefficients:
          ar1      ar2  intercept
      1.1389  -0.1850     0.0520
s.e.  0.0837   0.0843     0.0141

sigma^2 estimated as 7.077e-05:  log likelihood = 455.51,  aic = -903.03
> tsdiag(rs.arima)
>
```

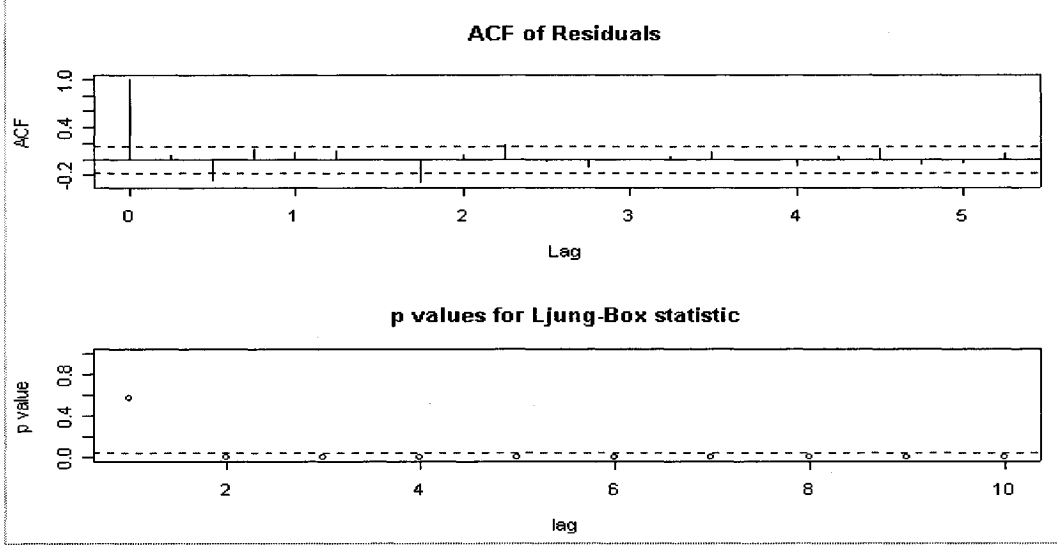


Figure 21: ARIMA(2,0,0) Model ($R_{GUI} - output$)

```
> fit<-best.arima(rs,d=1)
> fit
Series: rs
ARIMA(1,1,4)(0,0,0)[4] model

Coefficients:
          ar1     ma1      ma2      ma3     ma4
      -0.6681  0.9973  -0.0993  -0.1008  0.2041
s.e.   0.2367  0.2535   0.1528   0.1702  0.1258

sigma^2 estimated as 5.956e-05:  log likelihood = 464.7,  aic = -917.4
> forecast1<-forecast(fit,h=8)
> forecast1
        Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
1988 Q1     0.05878697 0.04889669 0.06867725 0.04366110 0.07391284
1988 Q2     0.05991802 0.04346752 0.07636852 0.03475916 0.08507688
1988 Q3     0.05992162 0.04067610 0.07916715 0.03048814 0.08935511
1988 Q4     0.05970443 0.03748700 0.08192186 0.02572580 0.09368306
1989 Q1     0.05984955 0.03441755 0.08528154 0.02095466 0.09874443
1989 Q2     0.05975259 0.03183148 0.08767370 0.01705093 0.10245424
1989 Q3     0.05981737 0.02938976 0.09024498 0.01328235 0.10635238
1989 Q4     0.05977409 0.02716990 0.09237827 0.00991029 0.10963788
> plot(forecast1,ylab="The Discount Rate on 91-day Treasury Bills")
```

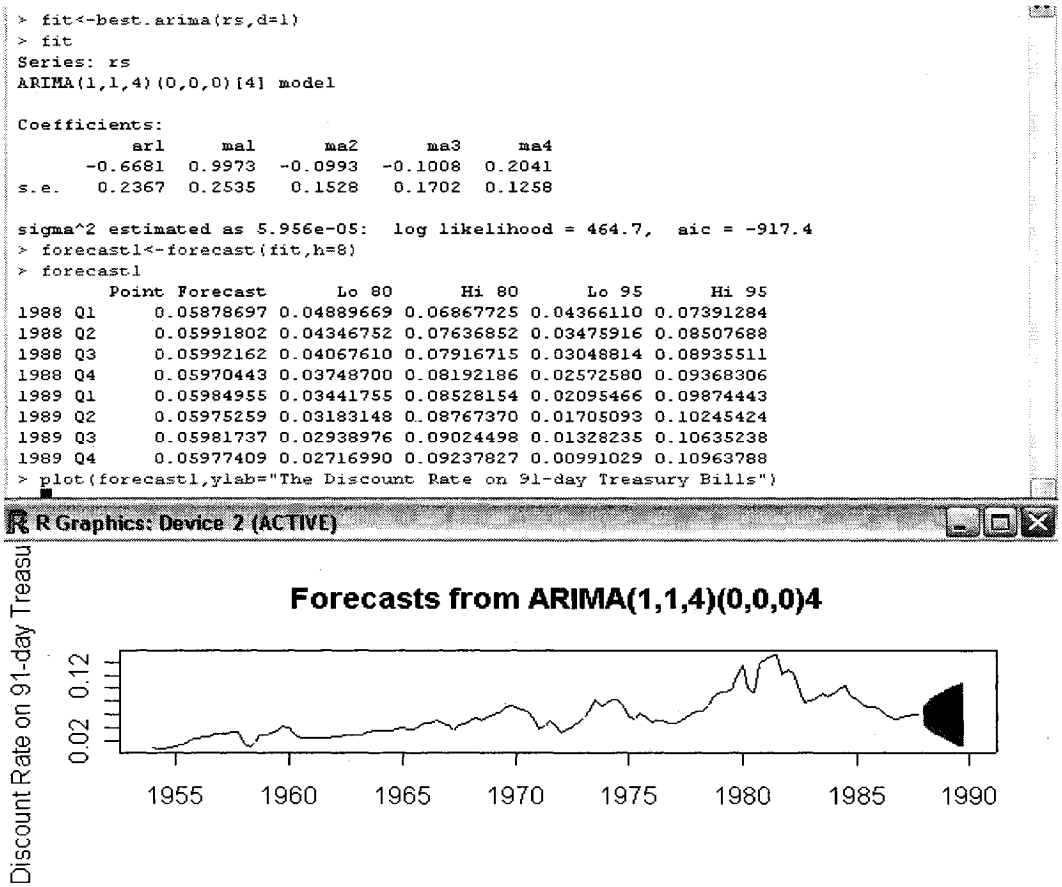


Figure 22: Best fit obtained by *best.arima* built-in function of R, ARIMA(1,1,4)

$(R_{GUI} - output)$

Figure 23 and Figure 24 show the n-steps ahead (i.e. 4 quarters) forecast of data using ARIMA(1,0,0) model. When we used R to calculate ARIMA, we can use either the *forecast* function ,as shown in Figure 23, or we can use the *predict* function (more descriptive) as shown in Figure 24. Both functions create very similar results.

```
> rs.arima<-arima(rs,order=c(1,0,0),n.cond=3)
> forecast1<-forecast(rs.arima,h=8)
> forecast1
         Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
1988 Q1      0.05966668 0.04869406 0.07063929 0.04288551 0.07644785
1988 Q2      0.05931420 0.04409366 0.07453475 0.03603639 0.08259201
1988 Q3      0.05897536 0.04068623 0.07726449 0.03100454 0.08694617
1988 Q4      0.05864962 0.03792473 0.07937450 0.02695364 0.09034559
1989 Q1      0.05833647 0.03559132 0.08108162 0.02355077 0.09312217
1989 Q2      0.05803544 0.03357114 0.08249973 0.02062052 0.09545035
1989 Q3      0.05774604 0.03179404 0.08369804 0.01805588 0.09743620
1989 Q4      0.05746784 0.03021308 0.08472260 0.01578528 0.09915040
> plot(forecast1,ylab="Discount Rate on 91-day Treasury Bills")
```

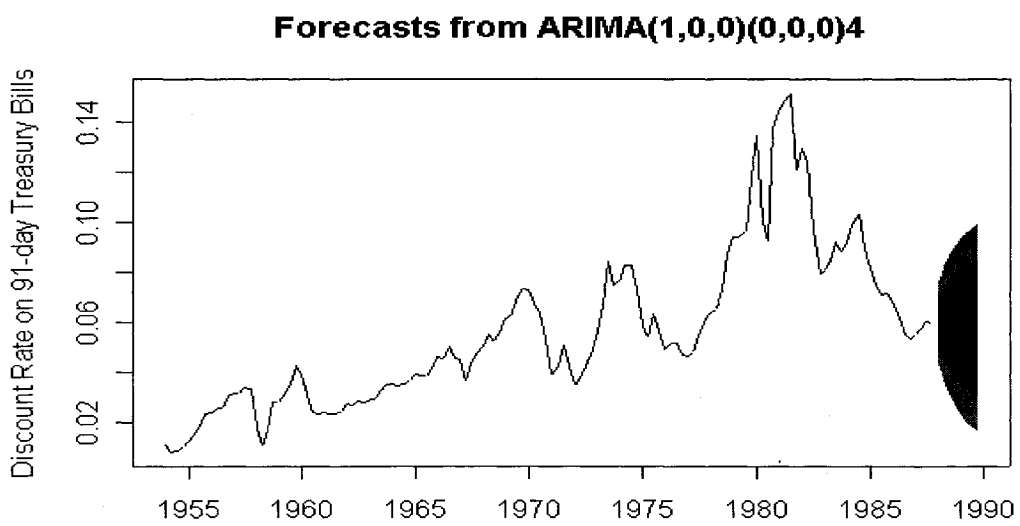


Figure 23: Forecasts for the next 8 quarters using ***forecast*** function

$(R_{GUI} - output)$

```
> TVsets<- as.matrix(c(rep(0,128),rep(1,8)))
> rs.fit<-arima(rs, order=c(1,0,0),xreg=TVsets)
> TVsets<- as.matrix(c(rep(0,127),rep(1,8)))
> TVsets<- as.matrix(c(rep(0,128),rep(1,8)))
> rs.fit<-arima(rs, order=c(1,0,0),xreg=TVsets)
> print(rs.fit)
Series: rs
ARIMA(1,0,0)(0,0,0)[4] model

Regression variables fitted:
   0
   0
   0
   . . .
   1
   1
   1

Coefficients:
         ar1  intercept     xreg
      0.9615     0.0509  -0.0020
s.e.  0.0218     0.0165   0.0086

sigma^2 estimated as 7.327e-05:  log likelihood = 453.18,  aic = -898.36
> predict(rs.fit,n.ahead=8,newxreg=as.matrix(rep(1,8)))
$pred
           Qtr1       Qtr2       Qtr3       Qtr4
1988 0.05960233 0.05918792 0.05878947 0.05840637
1989 0.05803801 0.05768384 0.05734331 0.05701589

$se
           Qtr1       Qtr2       Qtr3       Qtr4
1988 0.00856005 0.01187495 0.01427018 0.01617193
1989 0.01774971 0.01909268 0.02025516 0.02127340
```

Figure 24: Forecasts for the next 8 quarters using ***predict*** function ($R_{GUI} - output$)

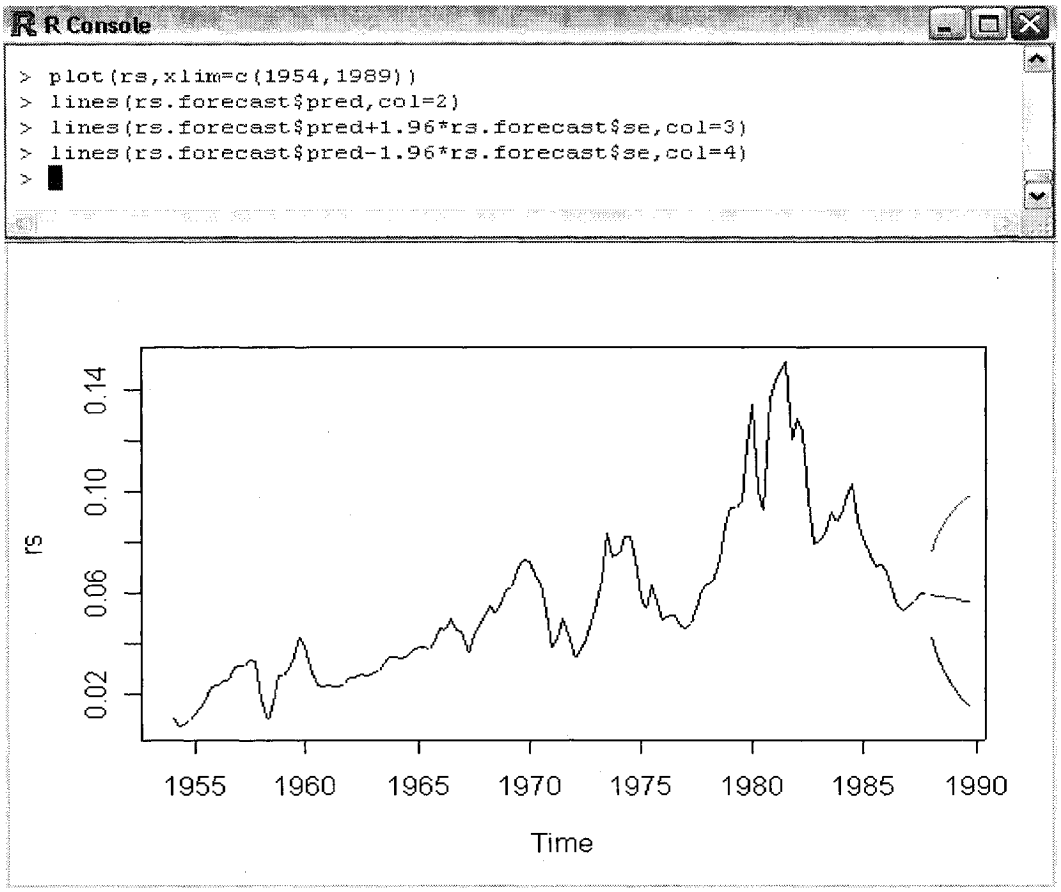


Figure 25: Plot of the forecasted values, ARIMA(1,0,0)(R_{GUI} – *output*)

6.1.5 Neural Networks Modeling (Non-Seasonal)

To forecast using neural networks, we use the commonly used network architect which is the Feed-forward single hidden layer network which uses Back-propagation training algorithm to estimate the weights (Lee, 1999). We first divide the data into training and validations sets. **Training**: The network is built based on the training data set. **Forecasting**: Based on the accuracy of out-of-sample forecasts, we select the "best"

network. The following R source code conducts a univariate forecasting modeling of the discount rate on the US 91-day treasury bills ("rs"). The source code is as follows:

```
> train <- function(newdata, validation,perceptron, hidden )
+ {set.seed(7777)
+ x<-as.matrix(newdata)
+ r<-nrow(x)
+ rval<-(r - validation)
+ XperceptronVars<- as.matrix(x[perceptron:(rval - 1)])
+ perceptronPlus<-perceptron + 1
+ Youtput<-x[perceptronPlus:rval]
+ valid<-as.matrix(x[rval:(r - 1)])
+ for(iter in 1:(perceptron - 1))
+ {
+ P<-perceptron - iter
+ R<-rval - iter
+ Xperceptron<- cbind(XperceptronVars, x[P:(R - 1)])
+ XperceptronVars<-as.matrix(Xperceptron)
+ Valid1<-cbind(valid, x[rval:(r - 1)])
+ valid<-as.matrix(Valid1)
+ }
+ sample<-nrow(XperceptronVars)
+ nn<-nnet(XperceptronVars,Youtput,size=hidden,
+ MaxNWts =100000,maxit=10000,linout=T,skip=T, decay=1e-2,reltol=1e-7,abstol=1e-7,range=1.0)
+ proutput<- predict(nn,XperceptronVars)
+ residualSS<-sum((Youtput - proutput)^2)
+ print(paste("Residual sum of squares:", residualSS))
+ Oh<-predict(nn,valid)
+ fcast<-x[(rval + 1):r]
+ ab<-sum(abs(proutput - Youtput)/ proutput) * 100
+ MAPE<- ab / sample
+ F_residualSS<-sum((fcast - Oh)^2)
+ print(paste("sum of squares of Forecasted errors:", F_residualSS))
+ print(paste("Mean Absolute Percent Error (MAPE):", MAPE))
+ print(paste("RMSE:",sqrt(residualSS / sample)))
+ print(paste("Forecasted RMSE:",sqrt(F_residualSS / validation)))
+ print("Network Architecture:")
+ print(nn)}
```

Figure 26: Sequential NN Training Source Code ($R_{GUI} - output$)

6.1.6 Multi-step ahead forecasting procedure for Non-Seasonal data

We now extend our procedure to take multiple steps ahead forecast.

1. Build the network based on the training set (i.e. 128 observations out of 136), Figure 27

2. Forecast 8 periods (within sample), and use the validation set (Sample 129 to 136) to compare the results, Figure 28. The comparison can be made by calculating the Root Mean Square Errors (Root MSE), and compare different networks to find the best possible model, Table 1. At this stage, we create different models by changing the value of input nodes, hidden nodes, maximum iterations,and decay function and compare their results.

3. Use the multi-step ahead forecasting algorithm for rs and forecast 8 periods ahead, by *pr(rs, 8, 3)* function, i.e. p stands for the number of input nodes which is only *three* in this example.

The summary of trained network in our example ($rs.nn$) reveals that we have a $3-4-1$ ($3-inputs$,$4-hiddennodes$, and $1-output$) network with 21 weights which creates the following forecast and hidden functions:

- $Forecast(i) = -1.17 - 0.68 * logistic(h1) - 0.68 * logistic(h2) - 0.68 * logistic(h3) - 0.68 * logistic(h4)$; the reason behind the logistics is the fact of non-linearities in the hidden units.
- $hidden1 = 0.19 - 0.10 * (i1) - 0.10 * (i2) - 0.10 * (i3)$
- $hidden2 = 0.19 - 0.10 * (i1) - 0.10 * (i2) - 0.10 * (i3)$
- $hidden3 = 0.19 - 0.10 * (i1) - 0.10 * (i2) - 0.10 * (i3)$
- $hidden4 = 0.19 - 0.10 * (i1) - 0.10 * (i2) - 0.10 * (i3)$

```
> x<-as.matrix(rs)
> # select the output: r(t)
> Youtput<-x[4:128]
> # obtain the input variables : r(t-1), r(t-2),and r(t-3)
> XinputVars<- cbind(x[3:127],x[2:126],x[1:125])
> #build a 3-4-1 network with skip layer connections and linear output.
> rs.nn<-nnet(XinputVars,Youtput,size=4,maxit=1000,decay=1e-2,reltol=1e-7,abstol=1e-7,range=1.0)
# weights:  21
initial  value 11.293926
iter  10 value 0.828521
iter  20 value 0.140968
iter  30 value 0.140483
iter  40 value 0.140472
iter  40 value 0.140472
iter  40 value 0.140472
final  value 0.140472
converged
> summary(rs.nn)
a 3-4-1 network with 21 weights
options were - decay=0.01
 b->h1 i1->h1 i2->h1 i3->h1
  0.19  -0.10  -0.10  -0.10
 b->h2 i1->h2 i2->h2 i3->h2
  0.19  -0.10  -0.10  -0.10
 b->h3 i1->h3 i2->h3 i3->h3
  0.20  -0.10  -0.10  -0.10
 b->h4 i1->h4 i2->h4 i3->h4
  0.20  -0.10  -0.10  -0.10
 b->o h1->o h2->o h3->o h4->o
-1.17 -0.68 -0.68 -0.68 -0.68
> #compute the residual sum of squares
> residualSS<-sum((Youtput-predict(rs.nn,XinputVars))^2)
> print(residualSS)
[1] 0.1056120
> eigen(nnetHess(rs.nn,XinputVars,Youtput),T)$values
 [1] 2.24939012 0.02783739 0.02783727 0.02783700 0.02641982 0.02007171
 [7] 0.02000091 0.02000021 0.01999997 0.01999997 0.01999997 0.01999987
[13] 0.01999987 0.01999987 0.01999277 0.01999199 0.01999161 0.01588813
[19] 0.01275439 0.01275384 0.01275290
```

Figure 27: Build the 3-4-1 network for rs with 21 weights. ($R_{GUI} - output$)

```
> train(rs,8,12,6)
# weights:  97
initial  value 3.758888
iter  10 value 0.064837
iter  20 value 0.015473
iter  30 value 0.015111
iter  40 value 0.015105
iter  50 value 0.015104
iter  60 value 0.015101
iter  70 value 0.015099
final  value 0.015099
converged
[1] "Residual sum of squares: 0.0105889988443687"
[1] "sum of squares of Forecasted errors: 0.000119885811384029"
[1] "Mean Absolute Percent Error (MAPE): 11.1843547557888"
[1] "RMSE: 0.0095542908055115"
[1] "Forecasted RMSE: 0.0038711401967642"
[1] "Network Architecture:"
a 12-6-1 network with 97 weights
options were - skip-layer connections  linear output units  decay=0.01
```

Figure 28: The results for a 12-6-1 network for *rs* with 97 weights. ($R_{GUI} - output$)

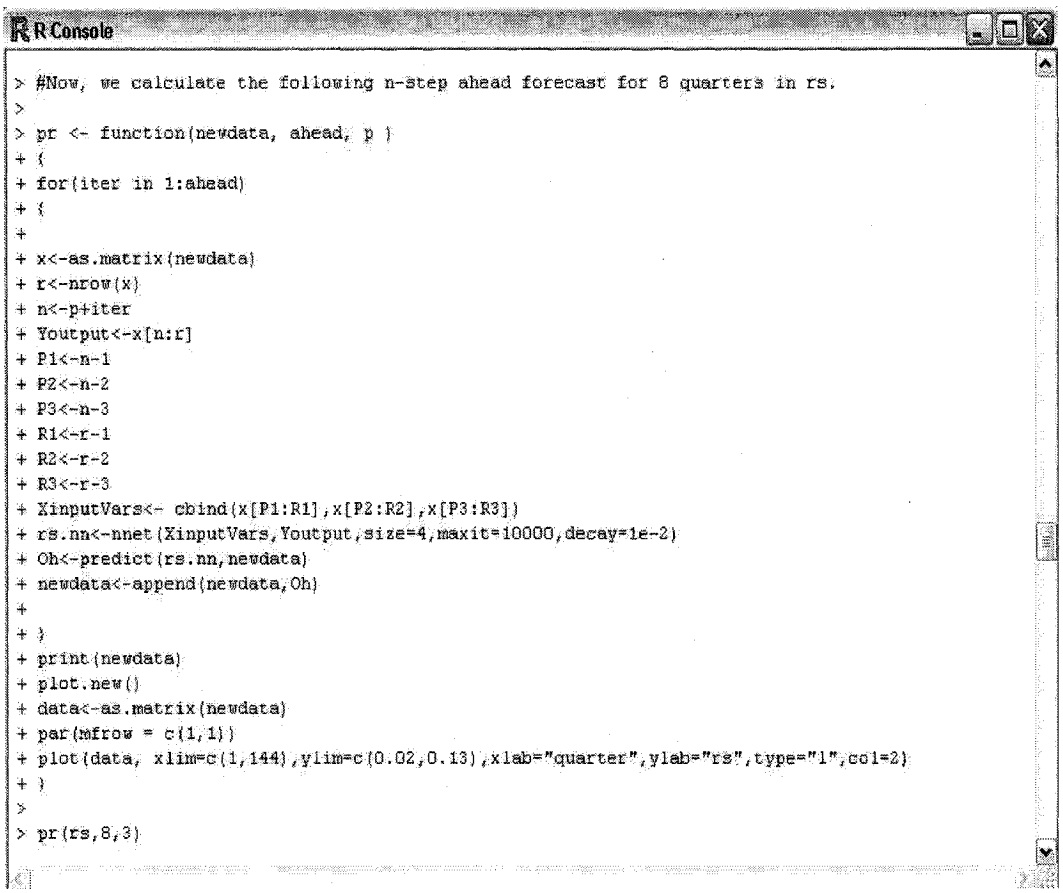


Figure 29: Multi-step ahead forecasting algorithm. ($R_{GUI} - output$)

6.1.7 Seasonal Data

Here, we look at 3 seasonal data sets, Air Passengers, Australian Beer, and Canadian Road Fatalities.

Our first seasonal data set is the famous *AirPassengers* data set from *R* directory which is the classic Box and Jenkins airline data, Figure 30. It contains the monthly totals of international airline passengers from 1949 to 1960. Our model selection

criteria for all NN models are MAPE (Mean Absolute Percentage Error), MSE, or RMSE. The main criterion is the lower value for the Forecasted RMSE, since our focus in this paper is forecasting. A model with low root mean square error for out-of-sample observations provides better results in multi-step ahead forecasting.

6.1.8 ARIMA Modeling (Seasonal)

It is not clear whether the seasonally differenced series is stationary, so we use the *adf.test*(AP.diff) command to do an augmented Dickey-Fuller test, Figure 31. The test is significant, i.e. Dickey-Fuller = -0.37907 and p-value = 0.02164, so we do no further differencing. Now, we fit the model as we did for the non-seasonal data,i.e. we try different models and compare their AIC values. According to the results shown on Table 2, ARIMA(0,1,1)(2,1,2) is the best model with the lowest AIC value.

```
> plot(stlAP<-stl(log10(AirPassengers), s.window=21))
> AP.stl<-stl(AirPassengers, "periodic")
> ds.AP<-AP.stl$time.series[,"trend"] + AP.stl$time.series[,"rema$
> TVsets<- as.matrix(c(rep(0,129),rep(1,15)))#Generate Separate T$
>
```

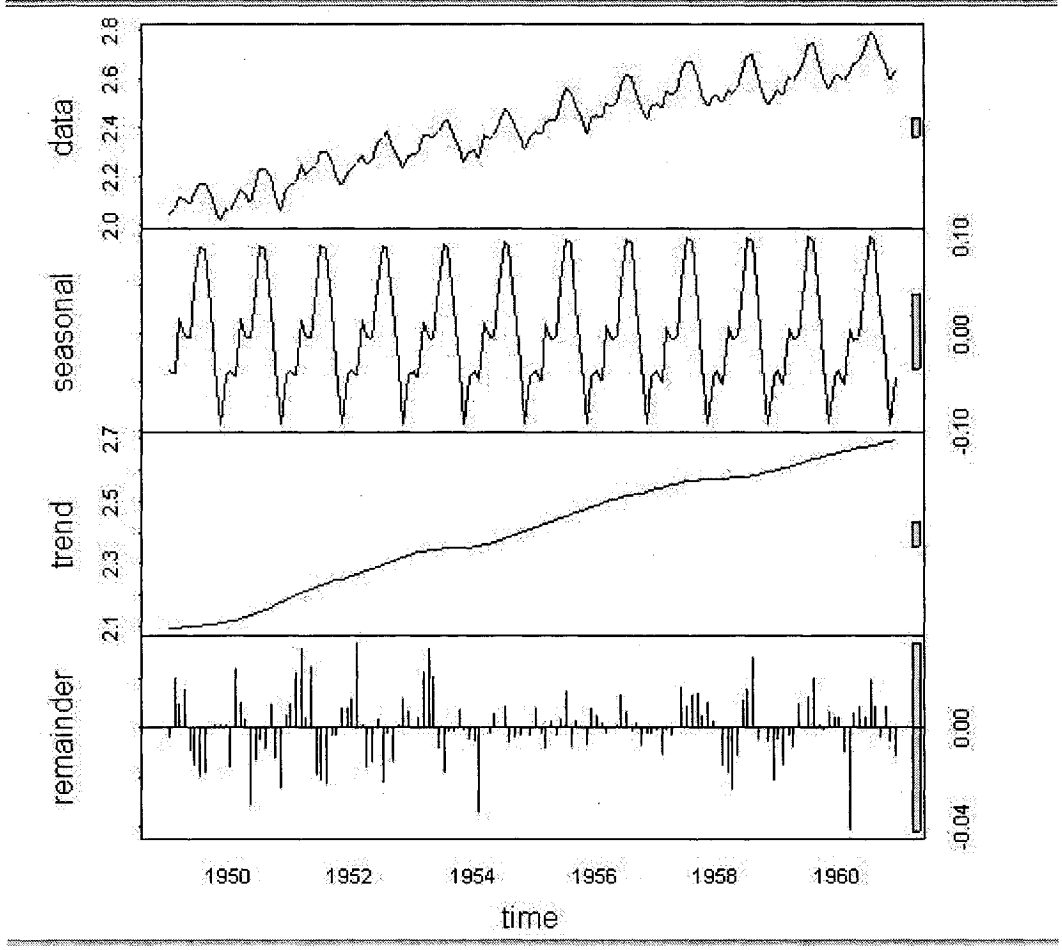


Figure 30: *AirPassengers* data set. ($R_{GUI} - output$)

```
> plot.new()
> adf.test(rs.diff)

        Augmented Dickey-Fuller Test

data:  rs.diff
Dickey-Fuller = -3.7907, Lag order = 5, p-value = 0.02164
alternative hypothesis: stationary

> AP.diff <- diff(AirPassengers,12)
> plot(AP.diff)
> par(mfrow = c(1,2))
> acf(AP.diff,lag.max = 40)
> pacf(AP.diff,lag.max = 40)
```

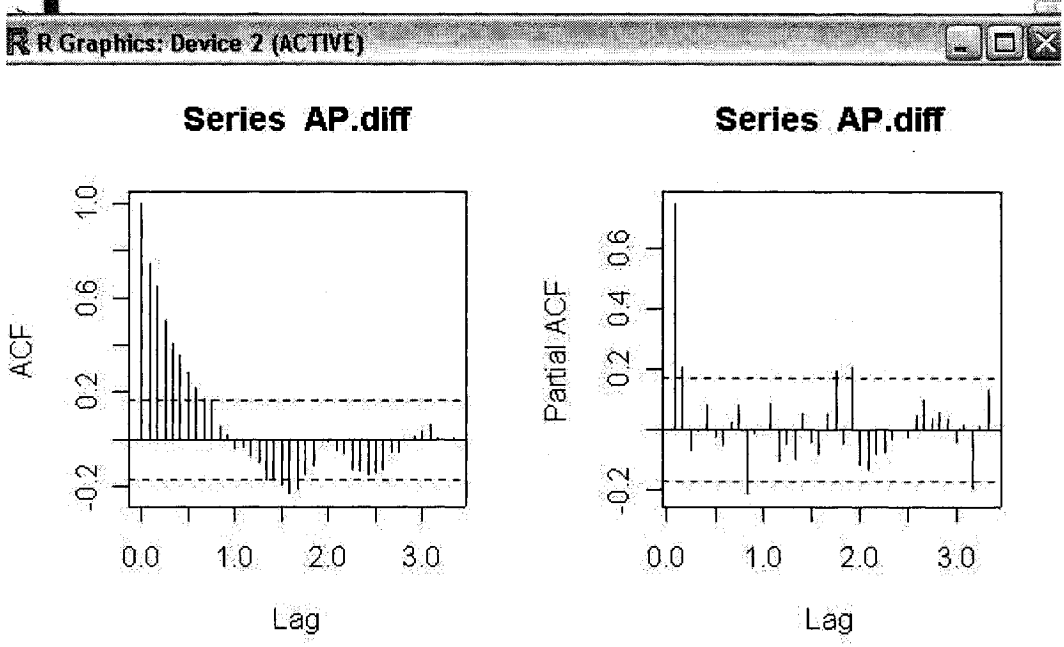


Figure 31: Dickey-Fuller test, ACF and PACF. ($R_{GUI} - output$)

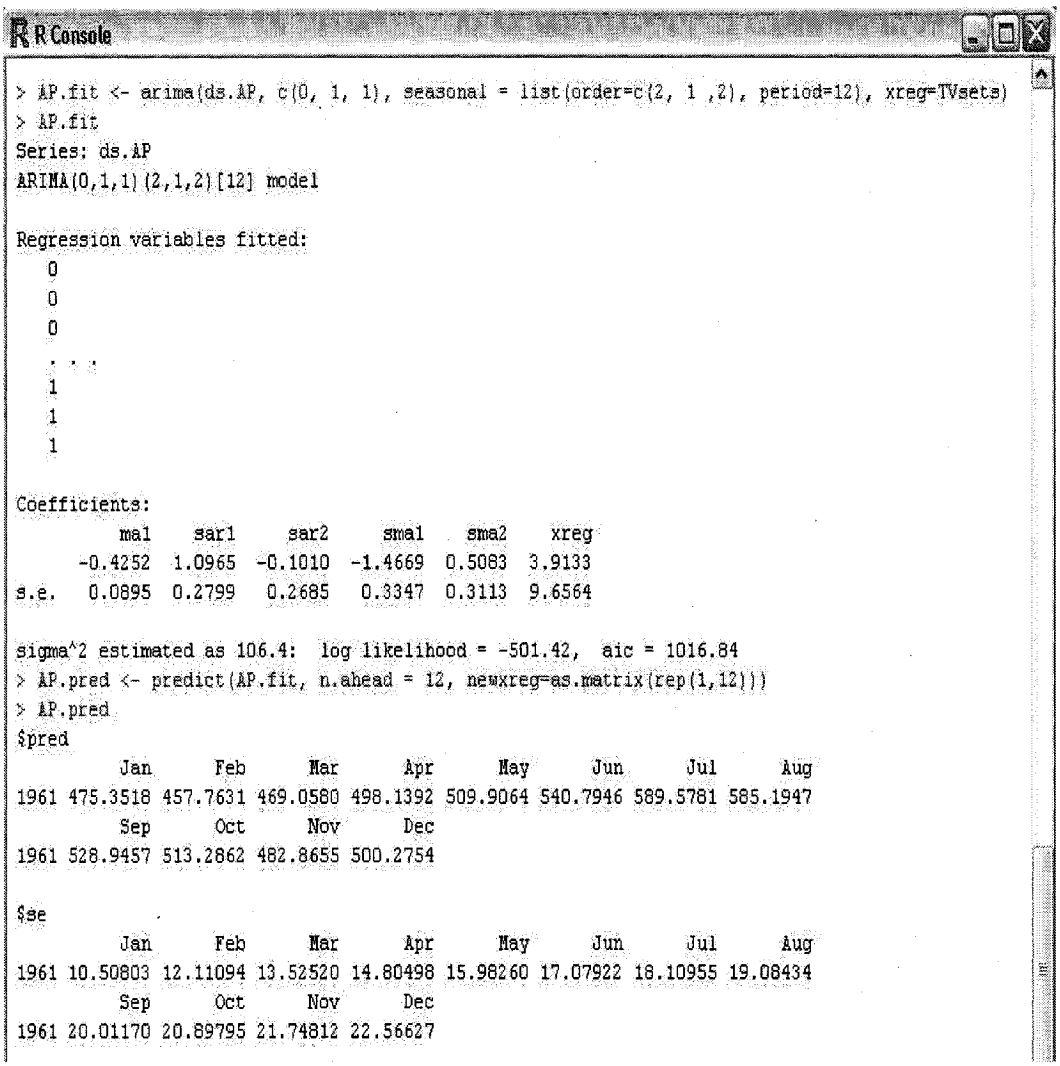


Figure 32: ARIMA (0,1,1)(2,1,2) model. ($R_{GUI} - output$)

```
> plot(AirPassengers, xlim=c(1949,1962), ylim =c(100,700))
> Low <- AP.pred$pred - 1.96 * AP.pred$se
> High <- AP.pred$pred + 1.96 * AP.pred$se
> lines(AP.pred$pred,col = 2)
> lines(Low,col=3)
> lines(High,col=4)
>
```

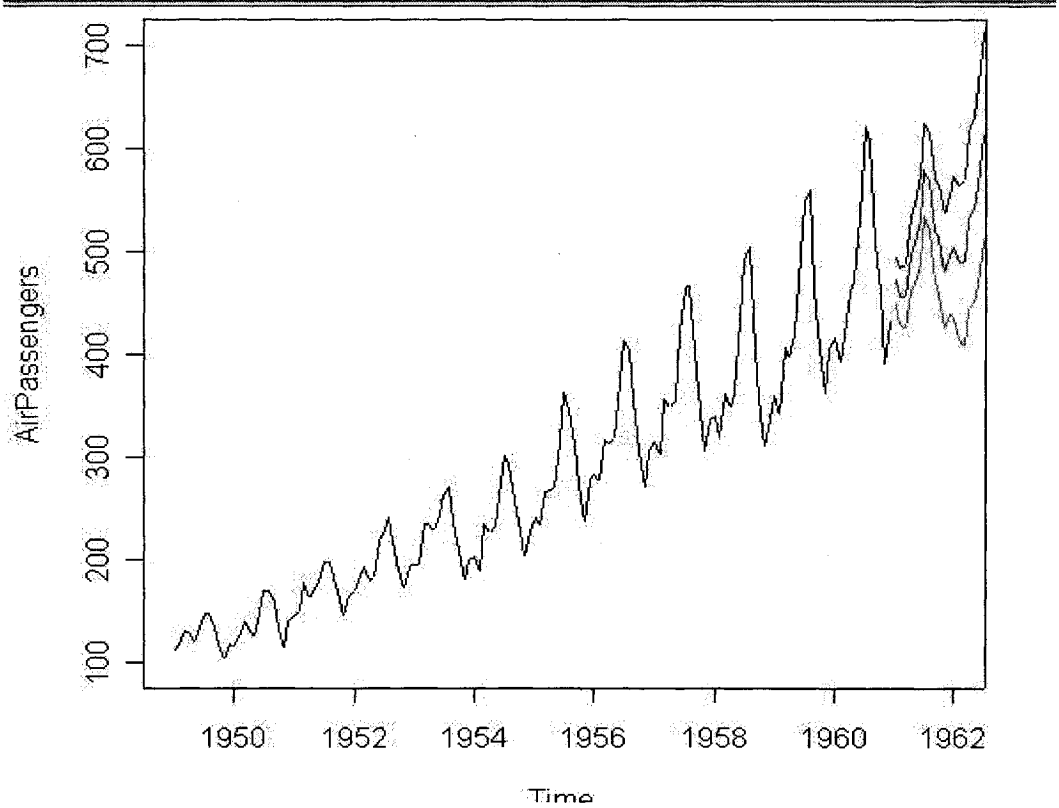


Figure 33: The graph of forecasted values (including the high and low)

6.1.9 Neural Networks Modeling (Seasonal)

When modeling seasonal data, we can use observations from specific lags (i.e. 1 and 7 for weekly, 1 and 12 for monthly) to train our network. However, this does not mean that we cannot train our network using sequential data input, as in the case

of non-seasonal models. As we are going to explain in this section, in many cases, using specific lags for training gives unsatisfactory results compared to sequential input training algorithm, refer to Appendix 6 and 7. (Note: for a 10-2-1 network, we create a matrix of $10 \times (N - 11)$ (N is the length of input data set) for our input variables. The first column contains the data from 1 to N-11, ..., and the 10^{th} column contains data from 10 to N-1, respectively, the desired output matrix is from 11 to N.)

When our input neurons are the sequential data from the same set, we are not treating each observation differently. This means that there will not be any difference between seasonal and non-seasonal network training. Neural networks treat both data sets the same way. Hence, when sequential data training has been used for seasonal sets, we may follow the same steps as it is explained in the forecasting of non-seasonal data.

Figure 34, *R-GUI* snap-shot, explains the process of forecasting *AirPassengers* data set using neural networks. We first generate separate Training and Validation sets (sub-samples), Appendix 6. We train the data based on the 128 samples, and build a 3-2-1 network; as noted before, this simple model is presented for to make the explanation of the forecasting process easier. Then, we forecast the next 16 periods using the trained network. Figure 37, compares the results of predicted values with the validation set.

```
> AP<-AP.stl$time.series[,"trend"] + AP.stl$time.series[,"remainder"]
> x<-as.matrix(AP)
> # select the output: r(t)
> Youtput<-x[4:128]
> #Generate Separate Test Set
> # obtain the input variables: r(t-1), r(t-2), and r(t-3)
> XinputVars<- cbind(x[3:127],x[2:126],x[1:125])
> # build a 3-2-1 network with skip layer connections and linear output.
> ap.nn<-nnet(XinputVars,Youtput,size=2,linout=T,skip=T,maxit=10000,decay=1e-2,
+ reltol=1e-7,abstol=1e-7,range=1.0)
# weights:  14
initial  value 246182.315736
iter  10 value 26834.647906
iter  20 value 26354.791380
iter  30 value 26182.363382
iter  40 value 24973.753618
iter  50 value 24772.763057
iter  60 value 24716.088280
iter  70 value 24690.420186
iter  80 value 24673.885419
iter  90 value 24623.050806
final  value 24622.594112
converged
> # print the summary results of the network
> summary(ap.nn)
a 3-2-1 network with 14 weights
options were - skip-layer connections  linear output units  decay=0.01
 b->h1 i1->h1 i2->h1 i3->h1
 -0.45   1.17  -3.68   2.90
 b->h2 i1->h2 i2->h2 i3->h2
 20.91   2.16   0.45  -2.75
  b->o  h1->o  h2->o  i1->o  i2->o  i3->o
 15.02  -7.93 -11.68   1.38  -0.19  -0.19
> # compute, then print, the residual sum of squares.
> residualSS<-sum((Youtput-predict(ap.nn,XinputVars))^2)
> print(residualSS)
[1] 24613.60
```

Figure 34: Build a 3-2-1 network ($R_{GUI} - output$)

```
> # setup the input variables in the forecsting subsample
> ap.p<-cbind(x[128:143],x[127:142],x[126:141])
> eigen(nnetHess(ap.nn,XinputVars,Youtput),T)$values
 [1] 5.727735e+07 1.730838e+07 6.344029e+04 1.540380e+04 5.176764e+03 4.658417e+03
 [7] 1.104682e+02 6.038808e+01 1.882834e+01 4.664797e+00 2.722052e-01 2.081996e-01
[13] 8.224462e-02 1.998873e-02
> # setup the input variables in the forecsting subsample
> ap.p<-cbind(x[128:143],x[127:142],x[126:141])
> # compute the forecasts
> Oh<-predict(ap.nn,ap.p)
> Oh
          [,1]
 [1,] 497.0856
 [2,] 436.6778
 [3,] 418.9235
 [4,] 418.9206
 [5,] 447.5783
 [6,] 442.0811
 [7,] 427.0413
 [8,] 423.3487
 [9,] 480.5583
[10,] 483.8099
[11,] 507.7079
[12,] 568.8831
[13,] 535.7668
[14,] 475.7852
[15,] 475.8142
[16,] 438.6619
> # select the observed returns in the forecasting subsample
> forecast<-x[129:144]
> # compute, then print, the sum of squares of forecast errors
> F_residualSS<-sum((forecast-Oh)^2)
> print(F_residualSS)
[1] 12235.77
```

Figure 35: Residuals ($R_{GUI} - output$)

6.1.10 Additional Time Series Data

The last two data set that we are going to look at are the Australian monthly beer production (January 1991 up to August 1995, 56 observations in total), and the Canadian monthly road casualties data (January 1975 up to June 1997, 270 observations

in total (Nash, 2001.)) The Figure 36 up to 40 are the analysis of the lags, trend and seasonality of the data.

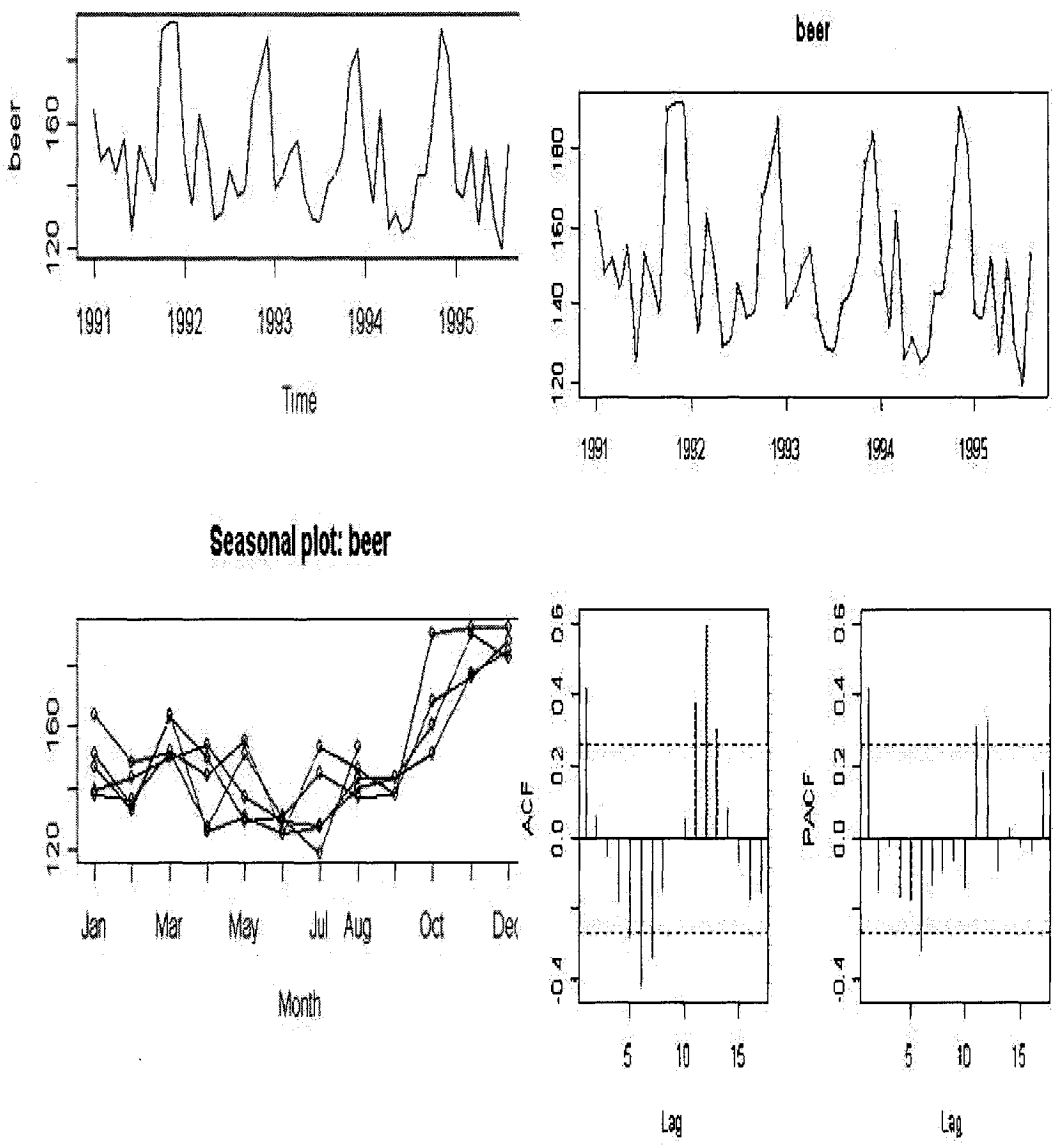


Figure 36: *Australian Beer*, lags analysis. ($R_{GUI} - output$)

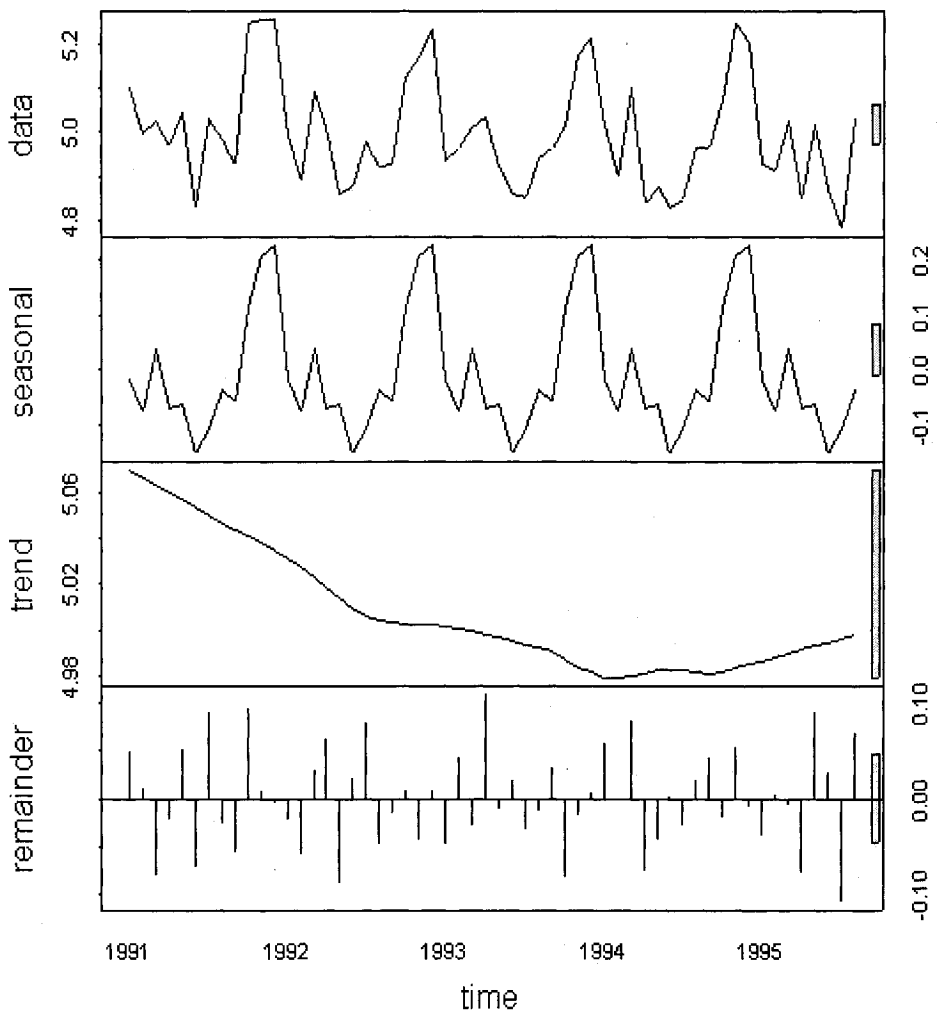


Figure 37: *Australian Beer*, trend and seasonality analysis. ($R_{GUI} - output$)

As can be seen from the above graphs, Australian monthly beer production has a downward trend and it is highly seasonal. Study of ACF and PACF graphs reveals the possible ARIMA models, as well as the neural networks models.

6.1.11 Holt-Winter's Method

During our time series analysis, we have realized that sometimes simpler forecasting models give satisfactory forecasting results. In Canadian Road Fatalities data analysis section, besides neural networks and ARIMA models, we also considered Holt-Winter's method (HW). It is an extension of exponential smoothing which takes into account a possible linear trend and additive or multiplicative seasonality.

- Notations: L for *Length of Seasonality*, T_t for *Trend component at time t*, X_i for i^{th} *seasonal term and factor*, $i = 1...L$, y_t for *seasonally-adjusted yt series*, e_t for *Random error at time t*, S_t for *Series Level*, and D_t for *Trend Factor*
- Multiplicative process: $Y = T \times X \times e$
- Additive process: $Y = T + X + e$
- According to figure 38-40, initial values (multiplicative process) for HW model achieved from R are $\alpha = 0.1989$, $\beta = 0$, and $\gamma = 0.4698$
- Series Level: $S_t = \alpha(y \div X_{t-L}) + (1 - \alpha)[S_{t-1} + D_{t-1}]; 0 < \alpha < 1$
- Seasonal Factor: $X_t = \beta(y \div S_t) + (1 - \beta)X_{t-L}; 0 < \beta < 1$
- Trend Factor: $D_t = \gamma(S_t - S_{t-1}) + (1 - \gamma)D_{t-1}; 0 < \gamma < 1$

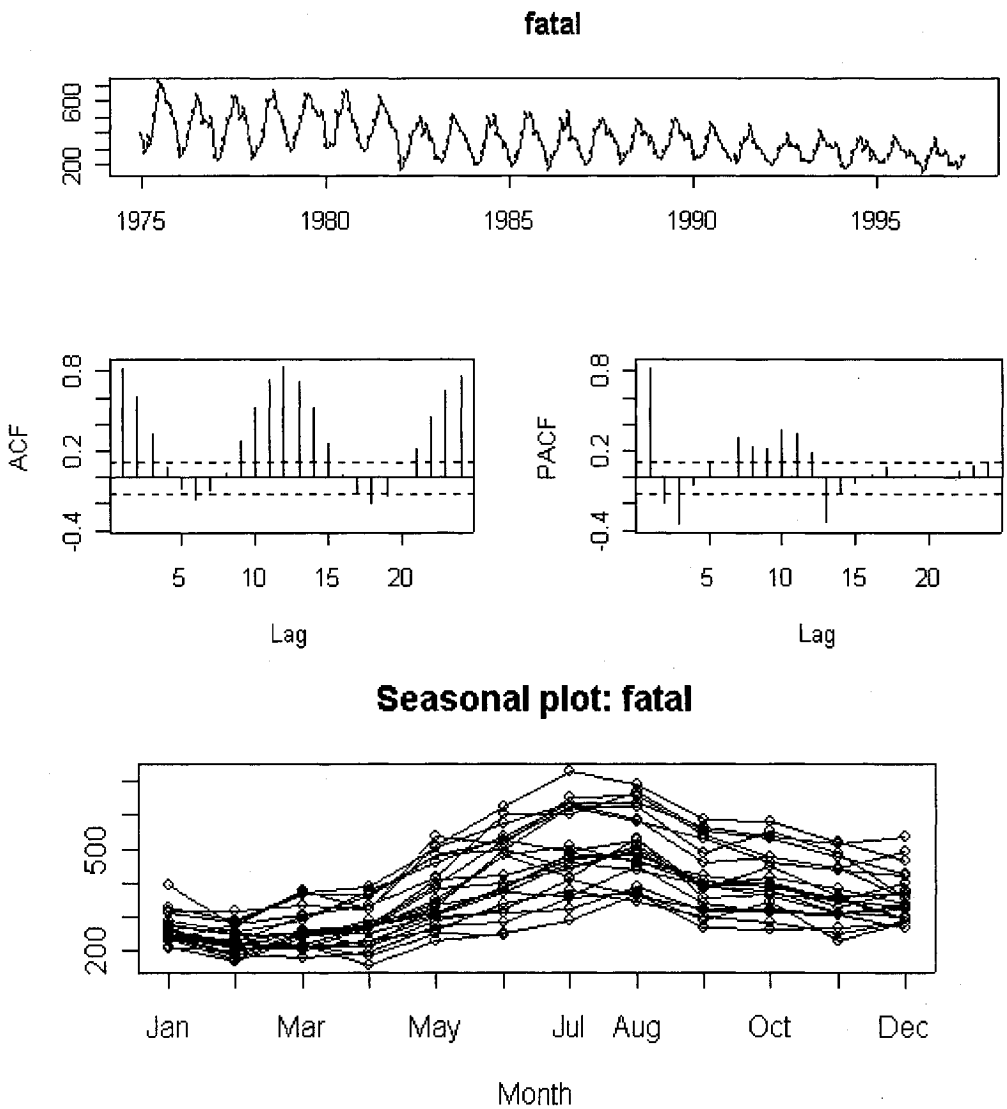
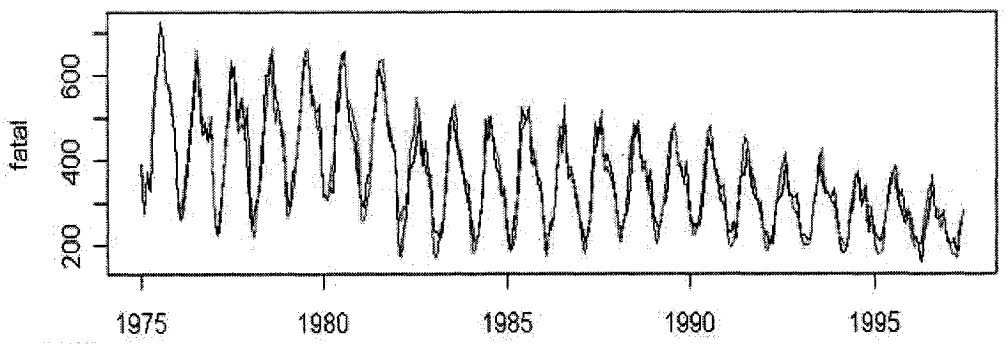


Figure 38: *Canadian Road Fatalities Graph*, trend and seasonality analysis.

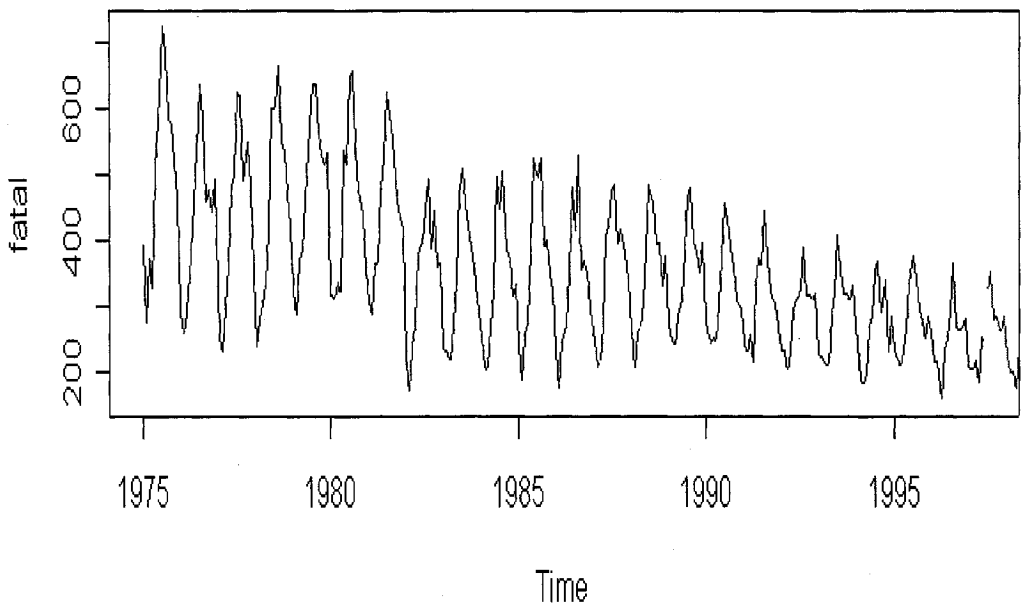
$(R_{GUI} - output)$




```
> HoltWinters(fatal)
Holt-Winters exponential smoothing without trend and with additive seasonal component.

Call:
 HoltWinters(x = fatal)

Smoothing parameters:
 alpha: 0.1989789
 beta : 0
 gamma: 0.4698582

Coefficients:
           [,1]
a    290.002544
s1    36.472124
s2    64.579528
s3    -7.941756
s4    -4.525299
s5   -29.488248
s6    -4.947919
s7   -65.386759
s8   -89.686143
s9   -89.237739
s10 -116.374746
s11  -49.734309
s12  -27.977674
```

Figure 39: Fitting Holt-Winter's model to *Canadian Road Fatalities* data.

$(R_{GUI} - output)$




```
> predict(hw, n.ahead = 12) #12 periods ahead
         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep
1997                                                        326.4747 354.5821 282.0608
1998 224.6158 200.3164 200.7648 173.6278 240.2682 262.0249
         Oct      Nov      Dec
1997 285.4772 260.5143 285.0546
1998
```

Figure 40: Forecasting 12-step ahead using Holt-Winter's method, *Canadian Road Fatalities.* ($R_{GUI} - output$)

In the Empirical Evaluation section, we will discuss the best forecasting model out of all 3 models.

6.2 Empirical Evaluation

We have conducted a comparison between the models of ARIMA and neural networks. We also considered Holt-Winter's method for one data set, *Canadian Road Fatalities.* Table 1 to 4 show the results of the seasonal and non-seasonal forecasting using all the models. We have used 10000 iterations to train the network which is enough for the network to converge. For each method, we have developed different models of forecasting. For modeling neural networks using seasonal data, we also considered specific lags training as well as sequential data training. In neural networks we have used a combination of 2 to 70 input neurons with a single hidden layer; note that single-hidden layer is the most commonly used between researchers.

As in the case of many forecasting methods, preprocessing the input data can improve the results very significantly. According to Atiya (1999), in neural networks, input and output preprocessing means extracting components from the input and their transformation to the output in a way that helps network to derive valuable information from the input variables and relate them to the required output. Preprocessing is a skill, and there is no single rule or set of standard steps. In this thesis, the de-trending and de-seasonality method has been used where ever they were applicable in our time series analysis. We used an error measurement, Root Mean Squared Error (RMSE), as our main comparison parameters. It is defined as follows:

$$MSE = \sum_1^n (X_t - X_{\hat{t}})^2/n, \text{ and } RMSE = \sqrt{MSE}$$

We also look at MAPE (Mean Absolute Percentage Error) when comparing some of

the models. MAPE is calculated as:

$MAPE = \Sigma_{t=1}^{T} \frac{|PE_t|}{T}$, in which PE stands for $\frac{(x_t - f_t)}{x_t} \times 100$;

- T is the total number of observations
- x_t is the observation
- f_t is the forecast

As we mentioned before, we have used a single layered network, using the *R* "nnet" training NN function, which uses the back-propagation method for error minimization purpose. From all the analysis has been done so far, we can observe a very high correlation between the training error and the testing error (validation). This means that there is relatively a good generalization by choosing an input sets that provides the lowest training error possible which eventually resulted a low testing error (i.e. F-RMSE: Root Mean Square Error of the Forecasted Values). Notations used in the Tables:

- MAPE, mean absolute percentage error
- FMAPE, forecasted mean absolute percentage error
- RMSE, root mean square error
- FRMSE, forecasted root mean square error

US Economy									
Neural Networks	N	Lags	MAPE	FMAPE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-4-1)	136	NO	40.41	12.8369	0.0095	0.0037	8	3	4
(3-2-1)	136	NO	40.4044	12.8376	0.0095	0.0037	8	3	2
(12-2-1)	136	NO	38.9411	12.7078	0.0095	0.003	8	12	2
(35-30-1)	136	NO	36.1589	10.93	0.0096	0.0078	8	35	30
(40-20-1)	136	NO	35.75	10.1736	0.0096	0.0117	8	40	20
(70-20-1)	136	NO	44.1852	8.8613	0.0091	0.0378	8	70	20
Arima	N	log likelihood	sigma^2	AIC	BIC	FMSE	RMSE	F-RMSE	Validation
(1,1,1)	136	458.73	0.0000653	-911.46	-893.9840707	0.00044	0.00808	0.02090	8
(2,1,1)	136	461.41	0.0000627	-914.82	-897.3440707	0.00032	0.00792	0.01800	8
(1,0,0)	136	453.15	0.0000733	-900.31	-882.8340707	0.00028	0.00856	0.01665	8
(2,0,0)	136	455.51	0.0000707	-903.03	-885.5540707	0.00034	0.00841	0.01856	8
(1,1,4)	136	464.7	0.0000595	-917.4	-899.9240707	0.00033	0.00771	0.01819	8

Table 1: Results of *rs* data set

AirPassengers									
Neural Networks	N	Lags	MAPE	FMAPE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(2-2-1)	144	(1,12)	44.6322784	41.084757	119.1771	220.048757	12	2	2
(3-2-1)	144	(1,2,12)	13.1485496	98.887879	120.158	704.957214	12	3	2
(3-2-1)	144	(1,12,13)	45.2303842	40.83470659	11.74514	216.699059	12	3	2
(3-3-1)	144	NO	43.7162	8.938	29.03	52.8335	12	3	3
(6-3-1)	144	NO	44.1538	4.3982	14.97	64.85	12	6	3
(12-3-1)	144	NO	41.79	4.0434	12.888	61.213	12	12	3
Arima	N	log likelihood	sigma^2	AIC	BIC	FMSE	RMSE	F-RMSE	Validation
(1,1,1)(1,0,1)	144	-506.6	132.4	1023.2	1041.01888	293.02592	11.50652	17.11800	12
(0,1,1)(0,1,1)	144	-507.21	134.8	1022.42	1040.23888	486.64360	11.61034	22.06000	12
(1,1,1)(2,1,2)	144	-501.33	111.3	1018.67	1036.48888	353.77848	10.54988	18.80900	12
(0,1,1)(2,1,2)	144	-500.63	96.11	1015.26	1033.07888	322.20250	9.80357	17.95000	12

Table 2: Results of *AirPassengers* data set

				Australian Beer					
Neural Networks	N	Lags	MAPE	FMAPE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(2-2-1)	56	(1,12)	9.0337567	11.3258076	20.3842	21.3125979	12	2	2
(2-1-1)	56	(1,12)	9.03375672	16.67748407	19.3106	38.6740255	12	2	1
(3-2-1)	56	(1,12,13)	9.17949516	11.32257156	21.3183	21.3144802	12	3	2
(3-3-1)	56	NO	14.19	9.5349	17.42	19.0116	12	3	3
(6-3-1)	56	NO	14.3997	4.3	9.074	25.76643	12	6	3
(12-3-1)	56	NO	14.4500	6.4707	0.4496	29.098	12	12	3
(18-3-1)	56	NO	14.766	0.0113	0.0267	22.063	12	18	3
Arima	N	log likelihood	sigma^2	AIC	BIC	FMSE	RMSE	F-RMSE	Validation
(1,0,1)(1,0,1)	56	-216.09	95.37	446	458.1521101	100.74338	9.76576	10.03710	12
(1,0,0)(1,0,1)	56	-220.63	118.6	453.26	465.4121101	119.20054	10.89036	10.91790	12
(1,1,1)(1,0,1)	56	-216.08	101	444.17	456.3221101	102.50753	10.04988	10.12460	12

Table 3: Results of *Australian Beer* data set

				Canadian Fatalities					
Neural Networks	N	Lags	MAPE	FMAPE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(2-2-1)	270	(1,12)	28.098038	46.94216	117.2931	116.97898	12	2	2
(2-1-1)	270	(1,12)	28.098038	46.942167	116.3622	116.97897	12	2	1
(2-1-1)	270	(1,12,13)	28.2087522	47.043125	117.9343	117.23687	12	3	2
(3-3-1)	270	NO	54.4283	14.4097	61.9428	55.2288	12	3	3
(6-3-1)	270	NO	45.177	14.2212	60.6358	54.37429	12	6	3
(12-3-1)	270	NO	61.5472	7.98081	34.8859	46.7052	12	12	3
(18-3-1)	270	NO	42.3344	8.7239	38.0575	47.5664	12	18	3
Holt-Winters	N		MAPE		RMSE	F-RMSE	Validation		
HW	270		9.53841		273.41323	27.324158	12		
Arima	N	log likelihood	sigma^2	AIC	BIC	FMSE	RMSE	F-RMSE	Validation
(1,0,2)(1,0,1)	270	-1379.13	1443	2774.25	2795.840532	2109.22848	37.98684	45.92634	12
(1,1,1)(1,0,1)	270	-1379.43	1441	2772.86	2794.450532	1979.41946	37.96051	44.49067	12
(2,1,2)(1,0,1)	270	-1369.13	1396	2754.26	2775.850532	1910.04274	37.36308	43.70404	12
(2,0,0)(1,0,1)	270	-1380.18	1470	2774.36	2795.950532	2345.87601	38.34058	48.43424	12

Table 4: Results of *Canadian Road Fatalities* data set

Table 1 compares the forecasting ability of NN and ARIMA models for non-seasonal data (i.e. *rs*). NN proves to be a reasonable model for this non-seasonal data set. Looking at the seasonal data sets, we can see that sometimes NN gives lower within-sample errors in compare to ARIMA, but when it comes to prediction, ARIMA has generally produced better forecasts.

Using NN for seasonal data, lags training does not always provide satisfactory forecasts. However, one-step ahead sequential data training has proven to generate better results almost in all seasonal cases. In the Australian Beer data set, although the results from training lags 1, 12, and 13 are close to our 3-3-1 sequential data training network, but the result of 3-3-1 model out runs (1,12,13) model. Among all the models, the ARIMA $(1,0,1)(1,0,1)$ has provided us with better results for both RMSE and Forecasted RMSE.

In the Canadian road fatalities data set, again, training the network with consecutive observations has given better results than forecasting by the lags. Here, we also used Holt-Winter's (HW) method (i.e. simpler model). Comparing all three models, i.e. HW, ARIMA,and NN, HW provides a better forecasting results than other two.

In AirPassengers data simulation, the results reveal that ARIMA has been the dominant forecasting model with its considerably lower RMSE and FRMSE. It is interesting to know that in the case of Canadian road fatalities data, ARIMA also conducted better forecasting than NN. This may suggest that when it comes to forecasting seasonal data sets, ARIMA appears to give better results than NN, although NN results are sometime very close to ARIMA.

The overall results of forecasting both seasonal and non-seasonal data sets show that neural networks also present very good forecasting models which can be used as an alternative to ARIMA models. Although both ARIMA and NN generate reasonable forecasts, but in some cases (i.e. Canadian road fatalities), simpler models such as

HW may create more satisfactory results than the other two. Refer to the Appendix 4 for selected R source codes. Note that over-fitting occurred with the 18-3-1 network (Australian Beer seasonal data set) with a very low RMSE for the training set (i.e. 0.0267) and very high (relative to RMSE for the within-sample) forecasted RMSE values (i.e. 22.063). In the coming chapter, we will talk more about the over fitting problem.

Chapter 7

Some applications of Neural Networks in forecasting

Neural Networks are applicable to many different fields such as medicine, and sport, but their applications are valued the most in the real world business problems. As mentioned neural network is best known for identifying trends and particular patterns in data; this may be the real reason why it is widely used in business world.

7.1 Application in Finance (Forecasting Indices and Stock Prices)

Investors and venture capitalist always follow up with what is happening in the market. For them any changes in the market means gain or loss of money. Although other than neural networks, some other models (i.e. random walk) are also being used in forecasting financial time series data, but neural networks is best known for its capability to extract weak low frequency periodic signals buried in strong high frequency signal (Renate Sitte, 2005.) This applies to daily trading high frequency data, Futures and Options Markets, and daily opening and closing of different indices

and equities in the market. The increasing availability of intraday high frequency data has made researchers to look at neural networks as an information source for volatility forecasting.

The volatility in the stock market creates fluctuations in the time series data. This can be observed as a mixture of slopes and knocks (rip or tank as the language being used amongst daily commodity traders (Toni Turner, 2000)). Hence, the trend, the day-to-day changes, and periodic variations is highly noticeable in the data. The trend is usually is the most identifiable factor in the long-term time series stock market data. However, day-to-day changes are very difficult to predict because of the randomness in its appearance. On the other hand, when it comes to periodic variations, we can see a seasonal pattern which stems to the business cycles in the economy; refer to the next topic to observe the affect of business cycle on sales data.

In this section we explore the forecasting value of data extracted from daily return series, its implied volatility, and high frequency returns within a day. Again, we compare the result of neural nets using R with ARIMA forecasting models. In the High Frequency data analysis section, we also look at Moving Average (MA) model which is widely used in daily commodity trading. MA analysis helps traders to predict the upcoming price trend. The main focus is to look at the daily return series of 10 different markets (indices), as well as looking at 2 intraday high frequency data sets, i.e Cisco Inc. and Microsoft Inc.

7.1.1 Description of Data:

1. NASDAQ COMPOSITE (11-Oct-84, 1-June-5): The NASDAQ Composite Index measures all NASDAQ listed common stocks on the Nasdaq Stock Market.

2. SP 500 (1-Jan-50,1-Jun-5): Standard and Poor's 500 is an index of 500 of the most widely held common stocks on the New York Stock Exchange (NYSE), as well as some non-NYSE stocks. It is one of the indicators of the overall health in the US stock market.

3. DOW JONES INDUSTRIAL (1-Oct-28, 1-Jun-5): Its Indices develops, maintains and licenses market indexes for investment products.

4. Other markets to look at: London -FTS100 (2-Apr-84, 1-Jun-5), France-CAC40 (1-Mar-90, 1-Jun-5), Germany-DAX100 (26-Nov-90, 1-Jun-5), Denmark-KFX (27-Jan-93, 1-Jun-5), Hong Kong (31-Dec-86,31-May-5), Russia-RTS (1-Sep-95, 2-Jun-5), Japan-Nikkei225 (4-Jan-84, 30-May-5.)

To evaluate the forecasting performance of Neural Networks against ARIMA models, we choose validation period as "out-of-sample" testing sample points, and keep the rest of observation as the training period to evaluate the goodness-of-fit in different models. For example we reserve the last 500 observations of NASDAQ COMPOSITE as validation period. The out-of-sample forecasting performance of the neural nets and ARIMA models are evaluated in terms of the following criteria: mean square error (MSE), or root mean square error (RMSE) for neural networks models and AIC, MSE and RMSE for ARIMA models. Tables 5-14 presents the results for both ARIMA and neural networks models.

It is observed from the results of all 10 tables that all the neural networks models, in general, perform much better outcome than ARIMA models when it comes to forecasting daily closed price of different indices in the market. There are cases that ARIMA shows a lower MSE than neural networks, as in the case of Hong Kong stock market, but the result of Forecasting RMSE suggests that neural networks is the best model when it comes to forecasting.

The performance of the best-performing neural networks model belongs to DOW JONES INDUSTRIES. A model with 30 input neurons and 30 hidden nodes with one hidden layer creates the best-performing model generating a within-sample RMSE of 2.7 and out-of-sample RMSE of 53.2 which is severely better result than ARIMA's (i.e. RMSE =38.24, and F-RMSE= 2619.3.) For SP500, Russia, NASDAQ, London, France, DAX100, and Denmark: NN is the best (i.e. better F-RMSE value) for out-of-sample forecast although the goodness of fit (RMSE) of both ARIMA and NN are very close. In case of Hong Kong and Japan, ARIMA models generate good mean square error for goodness of fit for validation periods, but the mean square error for the out-of-sample observations is very high.

Nasdaq								
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation
(2,1,3)	5207	-1439.820	0.102	2891.950	2931.297	0.319	4.849	500
(2,0,2)	5207	-1465.020	0.103	2942.040	2981.387	0.320	5.349	500
(1,1,1)	5207	-1461.260	0.103	2928.520	2967.867	0.320	5.131	500
(3,1,3)	5207	-1438.950	0.102	2891.900	2931.247	0.319	4.818	500
(1,0,0)	5207	-1470.380	0.103	2946.750	2986.097	0.321	4.671	500

Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	5207	509.262		0.108	0.089	0.329	0.299	500	3	3
(3-4-1)	5207	507.349		0.108	0.045	0.328	0.211	500	3	4
(6-6-1)	5207	440.582		0.094	0.044	0.306	0.211	500	6	6
(15-15-1)	5207	191.025		0.041	0.044	0.202	0.210	500	15	15
(30-30-1)	5207	44.330		0.009	0.044	0.097	0.210	500	30	30
(40-20-1)	5207	33.431		0.007	0.047	0.085	0.216	500	40	20
(70-30-1)	5207	6.087		0.001	0.053	0.036	0.231	500	70	30

Table 5 : NASDAQ COMPOSITE (11-Oct-84, 1-June-5)

DOW								
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation
(0,1,2)	19249	-97449.30	1463.00	194904.60	194951.79	38.25	2619.30	10000
(0,1,1)	19249	-97456.24	1464.00	194916.50	194963.69	38.26	2694.08	10000
(2,1,1)	19249	-97447.75	1463.00	194903.50	194950.69	38.25	2547.77	10000
(4,1,1)	19249	-97427.43	1460.00	194866.90	194914.09	38.21	2615.63	10000

Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	19249	77998.68		8.41	2809.82	2.90	53.01	10000	3	3
(12-3-1)	19249	77373.19		8.35	2809.00	2.89	53.00	10000	12	3
(12-12-1)	19249	7416615.00		8.01	2809.00	2.83	53.00	10000	12	12
(15-10-1)	19249	77302.42		8.35	2809.00	2.89	53.00	10000	15	10
(25-10-1)	19249	73668.00		7.95	2811.12	2.82	53.02	10000	25	10
(15-12-1)	19249	76711.92		8.29	2809.00	2.88	53.00	10000	15	12
(30-30-1)	19249	67219.04		7.29	2811.12	2.70	53.02	10000	30	30
(12-20-1)	19249	77347.84		8.35	2809.00	2.89	53.00	10000	12	20
(12-9-1)	19249	70960.97		7.67	2811.12	2.77	53.02	10000	12	9
(40-20-1)	19249	69821.63		7.56	2813.24	2.75	53.04	10000	40	20
(30-15-1)	19249	72916.93		7.91	2811.12	2.81	53.02	10000	30	15
(35-35-1)	19249	73461.83		7.95	2809.00	2.82	53.00	10000	35	35

Table 6: DOW JONES INDUSTRIAL (1-Oct-28, 1-Jun-5)

S&P 500								
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation
(0,1,1)	13942	-43702.83	30.95	87409.66	87454.92	5.56	123.33	1000
(2,1,3)	13942	-43668.59	30.79	87351.18	87396.44	5.55	106.53	1000
(3,1,2)	13942	-43671.16	30.81	87354.32	87399.58	5.55	106.65	1000
(2,1,2)	13942	-43680.32	30.85	87370.64	87415.90	5.55	106.08	1000

Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	13942	297845.78		23.02	132.54	4.80	11.51	1000	3	3
(3-6-1)	13942	295211.33		22.82	132.96	4.78	11.53	1000	3	6
(6-6-1)	13942	278568.95		21.54	132.75	4.64	11.52	1000	6	6
(12-6-1)	13942	281875.49		21.80	132.37	4.67	11.51	1000	12	6
(12-12-1)	13942	295007.30		22.82	132.73	4.78	11.52	1000	12	12
(15-5-1)	13942	292729.97		22.65	132.64	4.76	11.52	1000	15	5
(25-20-1)	13942	199270.17		15.43	132.45	3.93	11.51	1000	25	20
(30-30-1)	13942	288402.61		22.34	132.78	4.73	11.52	1000	30	30

Table 7: SP 500 (1-Jan-50,1-Jun-5)

London (FTS100)

Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation
(0,1,3)	5346	-27520.94	1740.00	55051.88	55091.38	41.71	587.80	500
(2,1,3)	5346	-27520.00	1740.00	55054.16	55093.66	41.71	598.99	500
(3,1,2)	5346	-27521.47	1741.00	55056.95	55096.45	41.73	603.14	500
(0,1,2)	5346	-27541.70	1754.00	55091.39	55130.89	41.88	647.00	500

Neural Networks	N	Residual SS	MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	5346	8929580.00	1843.84	853.81	42.94	29.22	500	3	3
(3-6-1)	5346	8929580.00	1843.84	853.81	42.94	29.22	500	3	6
(12-12-1)	5346	8745706.00	1808.00	854.39	42.53	29.23	500	12	12
(20-20-1)	5346	8705346.00	1820.73	856.15	42.67	29.26	500	20	20
(15-15-1)	5346	8634628.00	1787.60	990.36	42.28	31.47	500	15	15
(4-4-1)	5346	8844375.00	1826.71	853.81	42.74	29.22	500	4	4
(30-30-1)	5346	8753828.00	1817.32	853.81	42.63	29.22	500	30	30
(40-20-1)	5346	8709950.00	1812.20	854.39	42.57	29.23	500	40	20
(35-35-1)	5346	8704480.00	1809.65	853.81	42.54	29.22	500	35	35
(40-30-1)	5346	8628295.00	1795.22	851.47	42.37	29.18	500	40	30
(50-35-1)	5346	8654360.00	1804.55	854.39	42.48	29.23	500	50	35
(200-3-1)	5346	8093966.00	1742.23	853.81	41.74	29.22	500	200	3
(200-10-1)	5346	8059365.00	1734.72	852.06	41.65	29.19	500	200	10
(150-3-1)	5346	8234960.00	1753.93	853.81	41.88	29.22	500	150	3

Table 8: London -FTS100 (2-Apr-84,1-Jun-5)

Germany (DAX 100)

Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation
(2,1,3)	3654	-20139.36	3610.00	40292.72	40329.94	60.08	711.80	300
(0,1,3)	3654	-20146.59	3624.00	40303.18	40340.40	60.20	707.65	300
(3,1,1)	3654	-20145.02	3621.00	40302.04	40339.26	60.17	715.60	300
(4,1,4)	3654	-20134.73	3600.00	40289.96	40327.18	60.00	708.69	300

Neural Networks	N	Residual SS	MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	3654	12879551.00	3844.00	1230.61	62.00	35.08	300	3	3
(4-6-1)	3654	12859004.00	3839.04	1230.61	61.96	35.08	300	4	6
(15-15-1)	3654	12718816.00	3809.36	1231.24	61.72	35.09	300	15	15
(100-3-1)	3654	12095669.00	3720.63	1231.31	61.00	35.09	300	100	3
(150-10-1)	3654	11804908.00	3684.49	1234.12	60.70	35.13	300	150	10
(300-3-1)	3654	10902326.00	3570.06	1233.41	59.75	35.12	300	300	3
(30-30-1)	3654	10724533.00	3226.24	1321.32	56.80	36.35	300	30	30
(1000-2-1)	3654	7022191.00	2983.34	1265.22	54.62	35.57	300	1000	2

Table 9: Germany-DAX100 (26-Nov-90, 1-Jun-5)

Russia (RTS)

Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation
(1,1,0)	2436	2693.210	0.006	-5380.420	-5345.631	0.080	0.879	200
(1,0,0)	2436	2677.120	0.006	-5346.240	-5311.451	0.080	0.779	200
(1,1,1)	2436	2693.210	0.006	-5378.420	-5343.631	0.080	0.897	200

Neural Networks	N	Residual SS	MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	2436	13.146	0.006	0.008	0.077	0.092	200	3	3
(6-6-1)	2436	12.552	0.006	0.008	0.075	0.090	200	6	6
(3-4-1)	2436	13.161	0.006	0.008	0.077	0.091	200	3	4
(10-5-1)	2436	12.231	0.005	0.008	0.074	0.091	200	10	5
(20-10-1)	2436	8.775	0.004	0.009	0.063	0.094	200	20	10
(30-10-1)	2436	7.566	0.003	0.009	0.059	0.092	200	30	10
(30-20-1)	2436	5.528	0.003	0.015	0.050	0.121	200	30	20

Table 10: Russia-RTS (1-Sep-95, 2-Jun-5)

					Hong Kong (Hang Seng)						
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation			
(4,1,1)	4554.00	-29583.35	25850.00	59180.69	59219.23	160.78	2460.44	450.00			
(3,1,4)	4554.00	-29584.87	25868.00	59187.74	59226.28	160.84	2565.82	450.00			
(1,1,2)	4554.00	-29593.25	25963.00	59196.50	59235.04	161.13	2469.37	450.00			
(2,0,1)	4554.00	-30343.96	35962.00	60699.92	60738.46	189.64	2000.15	450.00			
Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes	
(3-3-1)	4554.00	116604895.00		28440.12	4596.84	168.64	67.80	450.00	3.00	3.00	
(3-6-1)	4554.00	116604895.00		28440.12	4596.84	168.64	67.80	450.00	3.00	6.00	
(12-12-1)	4554.00	114656007.00		28026.11	4610.41	167.41	67.90	450.00	12.00	12.00	
(30-15-1)	4554.00	114446969.00		28096.46	4596.84	167.62	67.80	450.00	30.00	15.00	
(100-5-1)	4554.00	109964888.00		27469.75	4594.13	165.74	67.78	450.00	100.00	5.00	
(30-30-1)	4554.00	130885235.00		32134.51	4890.20	179.26	69.93	450.00	30.00	30.00	
(200-5-1)	4554.00	105277839.00		26973.46	4596.84	164.24	67.80	450.00	200.00	5.00	
(500-5-1)	4554.00	89899042.57		24951.05	4608.92	157.96	67.89	450.00	500.00	5.00	
(100-20-1)	4554.00	116580335.00		29123.13	5112.25	170.66	71.50	450.00	100.00	20.00	

Table 11: Hong Kong-Hang Seng (31-Dec-86, 31-May-5)

					Japan (Nikkei 225)						
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation			
(3,1,0)	5267	-26633.45	64729.00	73276.90	73316.32	254.42	3844.96	500			
(1,1,0)	5267	-36647.97	65087.00	73301.94	73341.36	255.12	4034.82	500			
(3,1,1)	5267	-36633.45	64729.00	73278.90	73318.32	254.42	3844.94	500			
(2,1,1)	5267	-36633.64	64734.00	73277.29	73316.71	254.43	3812.26	500			
Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes	
(3-3-1)	5267	337694182.00		70899.18	6303.41	266.27	79.39	500	3	3	
(3-4-1)	5267	337694182.00		70899.18	6303.41	266.27	79.39	500	3	4	
(15-15-1)	5267	336391695.00		70803.89	6226.95	266.09	78.91	500	15	15	
(30-15-1)	5267	333768161.00		70474.32	6177.17	265.47	78.60	500	30	15	
(30-30-1)	5267	333777748.00		70476.44	6175.29	265.47	78.58	500	30	30	
(100-5-1)	5267	325615639.00		69784.20	6175.29	264.17	78.58	500	100	5	
(50-15-1)	5267	333002173.00		70618.84	6155.19	265.73	78.46	500	50	15	
(40-40-1)	5267	333633868.00		70594.90	6458.37	265.70	80.36	500	40	40	

Table 12: Japan-Nikkei225 (4-Jan-84, 30-May-5)

					France (CAC40)						
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation			
(0,1,3)	3839	-20279.13	2281.00	40568.26	40605.78	47.76	537.08	300			
(3,1,2)	3839	-20277.95	2280.00	40569.90	40607.42	47.75	515.99	300			
(0,1,2)	3839	-20286.20	2290.00	40580.40	40617.92	47.85	574.73	300			
(3,1,3)	3839	-20276.82	2278.00	40567.63	40605.15	47.73	511.70	300			
Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes	
(3-3-1)	3839	8526121.00		2411.79	856.73	49.11	29.27	300	3	3	
(4-3-1)	3839	8501005.00		2405.41	856.73	49.05	29.27	300	4	3	
(12-3-1)	3839	8467699.00		2401.49	856.97	49.01	29.27	300	12	3	
(15-15-1)	3839	8252901.00		2342.56	858.49	48.40	29.30	300	15	15	
(25-25-1)	3839	7854680.00		2235.40	877.94	47.28	29.63	300	25	25	
(150-3-1)	3839	7659850.00		2260.05	857.32	47.54	29.28	300	150	3	
(150-10-1)	3839	7659968.00		2260.05	855.56	47.54	29.25	300	150	10	
(200-10-1)	3839	7438122.00		2227.84	861.42	47.20	29.35	300	200	10	
(100-50-1)	3839	7988753.00		2323.24	857.32	48.20	29.28	300	100	50	

Table 13: France-CAC40 (1-Mar-90, 1-Jun-5)

Denmark (KFX)											
Arima	N	log likelihood	sigma^2	AIC	BIC	RMSE	F-RMSE	Validation			
(0,1,1)	3085	-7122.33	5.95	14250.66	14286.87	2.44	31.80	300			
(1,1,1)	3085	-7122.10	5.94	14252.21	14288.42	2.44	31.55	300			
(0,0,2)	3085	-7140.85	5.94	14291.71	14327.92	2.44	31.79	300			
Neural Networks	N	Residual SS		MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes	
(3-3-1)	3085	16876.74		6.07	4.74	2.46	2.18	300	3	3	
(30-30-1)	3085	13683.09		4.97	4.72	2.23	2.17	300	30	30	
(15-15-1)	3085	13315.61		4.88	4.71	2.19	2.17	300	15	15	
(300-3-1)	3085	13952.96		5.62	4.74	2.37	2.18	300	300	3	
(100-5-1)	3085	16005.36		5.95	4.74	2.44	2.18	300	100	5	
(100-15-1)	3085	5307.82		1.96	7.73	1.40	2.78	300	100	15	
(6-6-1)	3085	16638.80		5.99	4.74	2.45	2.18	300	6	6	
(1000-2-1)	3085	7392.73		4.12	5.38	2.03	2.32	300	1000	2	

Table 14: Denmark-KFX (27-Jan-93, 1-Jun-5)

The growing interest in modeling and forecasting stock market daily trading data over the past years has made researchers pay more attention to Artificial Neural Networks. As changes in stock prices are known to exhibit non-linear dependencies, Neural Networks has successfully been used by some researchers for forecasting. Results from the above simulation (i.e.10 different market) indicate a superior performance of Artificial Neural Networks as compared to ARIMA models. The only hard step in modeling using neural networks is deciding subjectively about the values of different parameters in neural networks such as number of input nodes, and hidden nodes (Note: the rule of thumb has been explained in the previous chapters). In many case, an appropriate set of parameters has been decided by using trial and error technique which allows the researcher to find out the range of correct parameters. For example, by looking at Denmark(KFX) market, when we used input neurons and hidden nodes less or more than 15, we did not achieve the same good results of a 15-15-1 network. Hence, we consider a network consists of 15 input and hidden neurons as our desired network.

7.2 Forecasting High Frequency Intraday Stock Price Data

High-frequency data are observations of financial intraday data taken daily in a short time sequence, i.e. 5, 10, 20 minutes or in seconds. Usually when we download the intraday high frequency data from Electronic Communication Networks (ECNs) which are designed for on-line trading, the data sets are time stamped (tick-by-tick data), CyberTrader. Every price fluctuation is called a tick. Depending on how active is the market for that particular stock, we can have 1 tick per minute or 100 ticks in a minute. When it comes to equity markets such as NASDAQ, and NYSE (New York Stock Exchange), usually all recorded trades and quotes are available on Trades and Quotes of NYSE (Turner, 2000).

Modeling and statistical analysis of high frequency data becomes a challenge because of the following reasons:

- Enormous number of daily observations (i.e. Average daily number of quotes in the USD/EUR spot market is about 20,000 per day), Zivot (2003)
- Error in the daily recorded data which demands adjustments and correction before conducting any type of analysis on the data
- Tick-by-tick data create an irregular spaced time series with random daily numbers of observations
- Moreover, this type of data usually shows periodic patterns in the market activities which is because by the occurrences of many correlated movements of other

indicators during the daily trading period, i.e. from 9:30 to 16:00 everyday in NASDAQ.

- Other factors which make the job of working with high-frequency data a hard task are the constant interruption in the market by Market Makers (MM), and quite hours in the market during Lunch time. According to Toni Turner, roughly from 10:30 to 11:30 we can feel the existence of MMs because of the higher volume (market Cap) and volatility in the equity price. Lunch time hours, from 11:30 to 12:30, also considered to be an unusual period for price movement in the market. (Note: "The National Association of Securities Dealers (NASD) defines a MM as a member firm that buys and sells NASDAQ securities at prices that market maker displays in NASDAQ, for its own account and risk", SwiftTrade Inc.)

In addition, discrete price movements, and constant changes in bid and ask during the day may distort inferences based on standard statistical models (Zivot,2003.) All these factors are the reasons behind why dealing with high frequency financial data is an extremely difficult task when it comes to forecasting.

7.2.1 Pre-processing High-Frequency Data

In this section, we have used the one-day $5-min$ trades and quotes for Cisco stocks (Symbol in NASDAQ: CSCO) listed under NASDAQ on June 15, 2005, as well as a long data series for the trades and quotes for Microsoft stocks (Symbol in NASDAQ:MSFT) for the period of 05/01/1997-05/ 5/1997, i.e. per second. CSCO data

set contains "price", number of shares traded, and trade time which starts from 9:30 am till 4:00 pm (every 5 Minutes). Also, MSFT data set contains "price" in US dollars, size of the trade (number of shares), trade date , and trade time which is in seconds since the midnight of the day(e.g. 34220); dataset taken from (Zivot,2003.)

7.2.2 Forecasting Cisco Stock Price

According to Toni Turner (2000), when a daily trader sits down in the morning, there are many important things that he/she needs to keep in mind. For example, the daily economic news may cause temporary reversal of the dominant trend, orchestrated by Market Makers, which allows them to 'shake out' Day Traders and enable them to either renew net buying at a lower price, or renew net selling from a higher price. This change of the trend may continue for a few days, depending on the importance of the economic news, i.e. change of interest rate is an important one which will affect the market for a while, Bloomberg Inc. This is another reason why modeling and forecasting High Frequency data is a difficult task.

SP500 (Standard and Poor's Index) is one of the powerful leading indicators for stock traders. They are the most liquid trading instrument in the world after the US dollar. They represent the reaction of the broadest array of market participants at any given time. By observing the price action f the SP 500 futures, traders will notice that more often than not, individual stocks has tendency to follow the movement of the futures. Keen traders will learn to recognize this correlation, and use it to successfully predict price movement, SwiftTrade Inc. Figure 40 shows the price trend for SP500 index

(Symbol: $\wedge GSPC$), CSCO, and Juniper Networks Inc (Symbol in NASDAQ: JNPR), one of Cisco's main competitors, on Wednesday (June 15, 2005.)

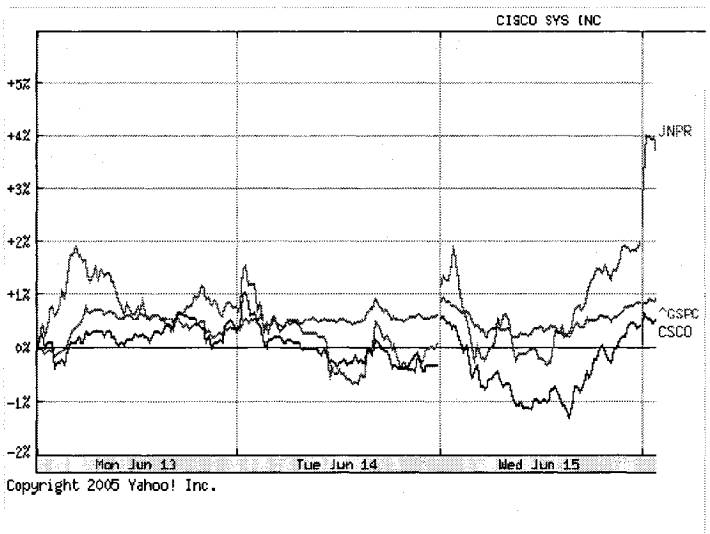


Figure 41: Graph of SP500, with CSCO and JNPR, Yahoo Inc.

All on-line decision support tools (Graphics Software) provided to active daily traders such as eSignal, and CyberTrader, are connected to ECNs (Electronic Communication Networks) to provide real-time information for the desired stock or index to on-line traders. One of the most important parameters that increase the ability of the trader to forecast the right price trend is the Graphical Presentation of Moving Average price. As it is shown in the following graph, we have the 5-min candle sticks data for CSCO stocks from CyberTrader GUI (June 15, 2005, $9:30-16:00$). At the same time, this application has provided us with 15-min and 200-min moving. Moving Average (MA) are calculations that smooth price movement to disclose the underlying trend, Figure 42.

Figure 42: Candlestick 5−*min* Graph of CSCO on CyberTrader.com

7.2.3 Comparing Neural Networks with Moving Average Model

Looking at the CSCO and MSFT results of neural networks models in compare to Moving Average (MA) model, we can observe the following results:

There are 78 observations in the CSCO on-day trade data set. After running different models of NN, a network with 12 input neurons and 12 hidden neurons with 1 hidden-layer creates the best result in compare to the MA result which is widely being used in on-line trend forecasting in daily trading. CSCO results are as follows:

Neural Networks	N	Residual SS	FRSS	MAPE	MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	78	0.0387	0.0046	0.0912	0.0006	0.0008	0.0235	0.0277	6	3	3
(6-6-1)	78	0.0327	0.0047	0.0843	0.0005	0.0008	0.0221	0.0278	6	6	6
(12-6-1)	78	0.0292	0.0046	0.0817	0.0005	0.0008	0.0219	0.0278	6	12	6
(12-12-1)	78	0.0288	0.0812	0.0000	0.0005	0.0008	0.0217	0.0275	6	12	12
Moving Average	N	Residual SS	FRSS	MAPE	MSE	FMSE	RMSE	F-RMSE	Validation		
MA	78	0.3430	0.0052	0.2900	0.0047	0.0009	0.0685	0.0295	6		

Table 15: CSCO simulation results for both NN and ARIMA models

As it is shown on the MSFT table of results, there are 32028 observations for the trading days between 05/01/1997 and 05/ 5/1997. The favorite network for forecasting MSFT stocks is 3-3-1 network (3 input neurons and 3 hidden nodes). The result of this network out-performs MA in finding the underlying trend of the MSFT stock. MSFT results are as follows:

Neural Networks	N	Residual SS	FRSS	MAPE	MSE	FMSE	RMSE	F-RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(3-3-1)	32028	7163.0350	156.7131	0.0619	0.2209	0.0435	0.4700	0.2086	3600	3	3
(6-6-1)	32028	6829.3452	157.7444	0.0598	0.2106	0.0438	0.4590	0.2093	3600	6	6
(12-3-1)	32028	6819.4042	157.1426	0.0593	0.2104	0.0437	0.4587	0.2089	3600	12	3
Moving Average	N	Residual SS	FRSS	MAPE	MSE	FMSE	RMSE	F-RMSE	Validation		
MA	32028	117272.4958	12367.6920	1.1992	3.6164	3.4355	1.9017	1.8535	3600		

Table 16: MSFT simulation results for both NN and ARIMA models

We considered Intraday High-Frequency Data (HF) which is a reporting of each individual price fluctuation for a specific stock. For this reason, HF Intraday prices are very useful when it comes to those wanting to analyze the trading strategies and

back-test their trading system. In our analysis, we considered 2 different stock prices (a small data set 5-min data, and a extremely big data set) to compare Neural Networks with conventional MA trend forecasting method in the currently used on-line trading tools. The results show that Neural Network is a very viable alternative to MA considering the speed of calculation and constant learning by the network. According to Toni Turner, since the profit margin for daily traders is usually between 1 to 3 cents, a more reliable forecasting tool can reduce the losses for the day and brings more money to traders.

7.3 Application in Business (Sales and Demand Forecasting)

ARIMA modeling (Moving Average , Autoregression , or combinations of both), traditionally, is used for forecasting sales. Since this model predicts future sales only based on past sales data, it doesn't perform as good when it faces situations like the influence of price (i.e. an external variable) on sale. Although a linear regression model can take external variables (i.e. price) into account, but the approximation function of linear regression model is restricted to be linear. When external variables have effect on the final result of our forecasting, Neural Networks becomes a viable alternative which takes into account these variables allowing arbitrary non-linear approximation functions learned directly from the data.

One of the biggest advantages of NN, as already mentioned, is that there is no need to make assumption about the specific functional form of the system that is going

to be modeled. Hence, basically, NN can estimate any function with arbitrary accuracy. The specific model is derived from the observations with only one limitation which is the size of the observed data. A bigger data set gives NN enough to create a well trained network. Since the most independencies between input variables and the amount of sale are non-linear, Neural Networks modeling becomes very appealing to the problem of sales forecasting. NN forecasting model of sales can directly be derived from historical data of the sales volume, prices, and other variables. So, we can say that NN learns the particular characteristics of each of the products rather than using the same functional form for all of them.

Here, we are going to look at the sales volume data from company X in Toronto, Canada (Note: The data has been provided by Professor John Nash at the University of Ottawa). The aim of this simulation is first to investigate the missing sales data in the series, and then to find a proper model which can give us a reasonable forecast of future sale.

7.3.1 Pre-processing Data for Forecasting

Neural Networks have difficulty to forecast growing seasonal time series data such as our Sales data. Hence, we need to de-trend and de-seasonalize the data before training the proper networks.

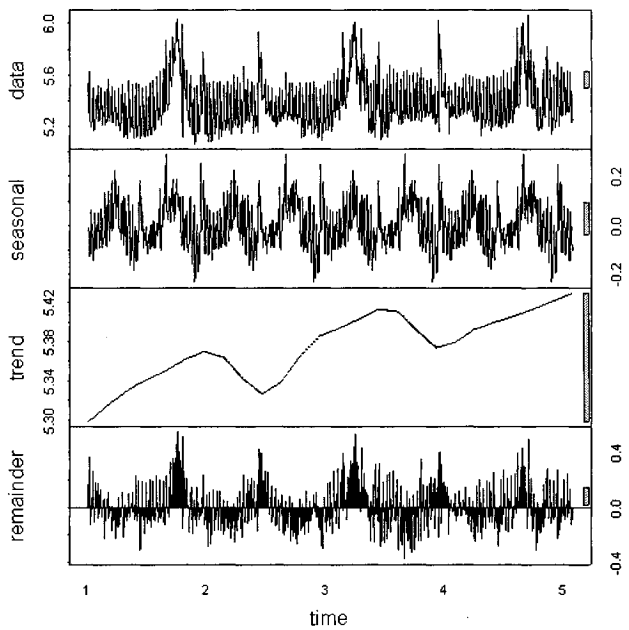


Figure 43: sales data analysis

Using *R*, we pre-process the data getting ride of the trend and seasonality and then using the remaining part, which is the original sales data, we build our first network (Nash, 2001.)

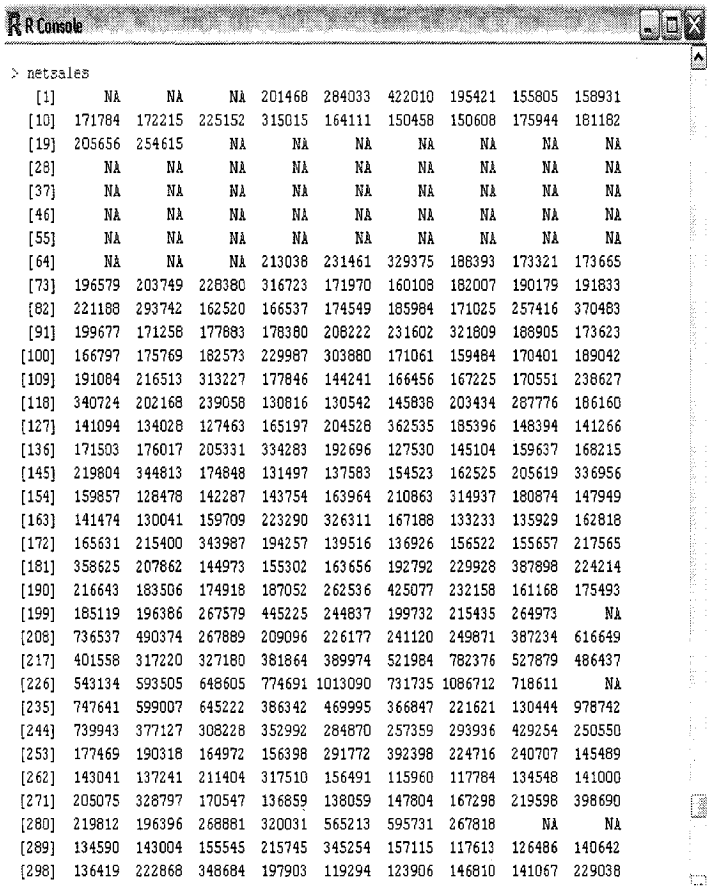


Figure 44: Sales Data set

```
 Call:
 stl(x = log10(tssales), s.window = 21)

Components
Time Series:
Start = c(1, 1)
End = c(3, 240)
Frequency = 240
              seasonal    trend     remainder
1.000000  0.0267369480 5.354349  3.316205e-02
1.004167  0.0310957903 5.354342  1.369746e-01
1.008333  0.1891920959 5.354335 -2.240350e-01
1.012500  0.1771149420 5.354328 -2.272370e-01
1.016667  0.0202198420 5.354321  7.882771e-02
1.020833  0.0672603678 5.354314  2.037480e-01
```

Figure 45: Detrend Sales data

7.3.2 Training the Network

In this particular data set, Figure 43, we have missing values (NA) of daily sales volume from this company; (number of available observations is 1067 which covers the period of 5/5/2000 to 4/6/2003). One of the biggest advantages of NN is to find the right model despites the missing data. In our case, the missing data is relatively very small portion of our data that is why it may not affect the final results as in the case of smaller data sets. The sales data contains, in total, 89 missing values in which almost half (49) of the missing values reside within the first 66 observations (out of 1105.) This gives a big chunk of good data to forecast and then back propagate and find the beginning missing values.

Dealing with Missing Values (NA)

R provides methods to handle NA values in time series analysis. For example, in the *nnet* function of the neural networks library of *R*, there is a parameter called *na.action*. It specifies the action to be taken if NAs are found. The default action is for the procedure to fail, but an alternative is to use *na.omit*, which leads to rejection of cases with missing values on any required variable.

When we ran the NN simulator, it first finds the best model out of the existing values then goes back and estimates the missing values (NA values) based on the best model chosen. At the end it forecasts the required steps ahead.

Table 17 contains the different NN models of the sales data. Again, our purpose is to

find the parameters for the desired forecasting model which gives us the lowest RMSE (root mean square error) over the time span of the series. We used the designed algorithm of Feed-Forward neural networks models built in R. We applied the training data set sequentially to our network which also uses back-propagation to minimize forecasting errors, refer to the algorithm in Appendix 7.

Based on the results indicated in Table 17, the best model is 3-3-1 network which has 3 input neurons and 3 hidden nodes with one hidden layer for the period of 5/May/2000 to 4/June/2003. Our trail and error contains input neurons from 3 days up to 240 days (roughly the operating days in a course of one year) and hidden nodes from 3 up to 30 days (a month.) The model suggests that if we feed every 3 days sequentially, we get the lowest Forecasted RMSE, which we are aiming for.

Sales									
Neural Networks	N	Lags	MAPE	FMAPE	RMSE	F.RMSE	Validation	Input Neurons (Units)	Hidden Nodes
(2-2-1)	1105	(1,7)	37.60	35.20	150151.20	204400.84	120	2	2
(3-3-1)	1105	(1,7,31)	38.15	35.45	151729.70	203953.75	120	3	2
(4-3-1)	1105	(4,5,6,7,8)	37.66	35.22	150671.60	204369.79	120	5	3
(3-3-1)	1105	NO	44.04	26.88	104774.81	155975.20	120	3	3
(6-6-1)	1105	NO	43.85	24.73	102010.83	159266.58	120	6	6
(12-12-1)	1105	NO	46.76	19.07	94019.14	170651.66	120	12	12
(24-12-1)	1105	NO	44.50	17.20	91141.53	170954.07	120	24	12
(24-24-1)	1105	NO	43.53	18.28	88076.20	162169.62	120	24	24
(30-30-1)	1105	NO	42.94	17.24	91140.59	170924.80	120	30	30
(120-7-1)	1105	NO	50.71	19.46	89999.06	171713.32	120	120	7
(120-12-1)	1105	NO	50.71	19.46	89999.08	171712.36	120	120	12

Arima	N	log likelihood	AIC	BIC	RMSE	F.RMSE	Validation
(1,1,1)	1105	-12747.11	25502.22	25532.27	112160.60	212327.45	120
(1,0,1)	1105	-12772.1	25554.21	25584.26	113578.17	147528.92	120
(1,0,0)	1105	-12775.56	25559.12	25589.17	113973.68	147192.10	120

Table 17: simulation results for both NN and ARIMA

7.4 Over fitting

One of the typical problems with NN is when we continue training the network where we have a very low within-sample MSE, but very high out-of-sample MSE (validation.) This is called over-fitting which usually happens with noisy data, but its more common when we choose high number of input neurons and a small set of data (refer to Table 5, 70-30-1 network for NASDAQ.) Usually when big networks are being chosen, the same way that they can approximate essentially any function they can also perfectly over-fit the noise in the data set. In the sales data presented before, because of the noisy, and relatively, big set we had be aware of over-fitting problem.

In the forecasting algorithm that has been designed for the sales data, we have used weight decay function during training. Using weight decay during training, the network prefers the low weight values, and other weights become zero. When weight decay function, i.e. $decay = 1e-2$, is being used the network is less affected by over-fitting because the approximation functions become smoother than those of models without weight decay function (Lars Niklasson, 2004.)

$nn < -nnet(inputmatrix, outputmatrix, size = hidden, MaxNWts = 100000, maxit = 10000, linout = T, skip = T, decay = 1e-2, reltol = 1e-7, abstol = 1e-7, range = 1.0)$

Chapter 8

Conclusion and Open Questions

This thesis presents some applications of neural networking algorithms to forecasting. The initial portion of this thesis entailed a study of the background as well as the different methodologies of artificial neural networks. The final portion presented a comparative analysis of different models of neural networks and ARIMA models for time series forecasting.

In the Empirical Evaluation section, the nature of the data being used (i.e. seasonal or non-seasonal) was described and the type of neural networking architecture (i.e. input-hidden-output) including the parameters being used have been identified.

Most of the models are accompanied by the computational results, *R*-source codes, the output of neural networking (trained heuristically) and ARIMA forecasting methods as well as multiple plots and tables for the empirical observations.

Empirical results are as good as the range of trials. What we have found so far indicates that neural networks do better forecasting than ARIMA consistently when

dealing with non-seasonal data (though sometimes only slightly better). Three out of four of trials for seasonal series show better forecasting values using ARIMA rather than neural networks. The final results indicate that depending on the nature of the data, whether seasonal or non-seasonal, neural networks may or may not give better results than ARIMA. However, based on the comparisons conducted in this thesis, the neural network models for non-seasonal data appear to be more successful than those for seasonal data.

While neural networks achieved better results than ARIMA 13 times out of 13 for non-seasonal data, they were only better 1 time in 4 for seasonal data series. Moreover, in most cases, when we ignored the seasonal structure of the seasonal data, the NN process actually did slightly better, which is a counter-intuitive result.

Nevertheless, the results show that artificial neural networks are a viable alternative to conventional forecasting techniques, such as ARIMA models. Based on the comparisons conducted in this thesis, we can say that the neural networks are competitive with ARIMA models for various time series data. Although simple rules have been suggested to estimate the possible number of input and hidden neurons, the user must still judiciously (using trial and error) select the appropriate number of input and hidden nodes, and this selection does not appear to be easily automated. Indeed, it is the opinion of the author that while ARIMA methods do a credible job in forecasting time series automatically, neural networks approaches require the intervention of a knowledgeable user.

With modification of hidden layers, activation function, input variables, skip-layer connections, and other elements of the nnet function of R, we can produce different results. This richness of choice, without particular guidelines for users as to which choices will improve performance, makes it awkward to generally recommend neural networks over other forecasting methods.

Furthermore, as indicated early in the introduction of thesis, there is an extensive literature concerning forecasting with neural networks. However, some of the reported outcomes from different reports are contradictory or at least inconclusive in providing guidance to prospective users. The present work attempts to provide such guidance. In particular, we have included the necessary tools to conduct time-series forecasting using neural networks with R which is widely available and freely downloadable.

One of the research issues that may be considered as future work is that of finding the solution to the problem of forecasting a series having a gap between estimation and validation periods:

- $---Estimation---|---Gap---|startforecast---endforecast$

This applies to those situations where we are interested in training the network based on some specific lags or with a particular "end of data", where we must then skip some of the time points before we generate forecasts.

This problem arises in practice where businesses cannot assemble and process data immediately, or where forecasts cannot be broadcast and used ready for the next time period. This exacerbates the difficulty of finding the best combination of controls for

the different methods given the estimate, gap, and forecasting periods. For NN, the controls are the inputs, hidden neurons and layers and the various neuron structures. For ARIMA, we have the selection of seasonal and non-seasonal autoregressive, differencing and moving average terms.

Clearly, more research and experience into the relative models and methods for forecasting can be envisaged. Such future work may include designing new architectures for neural networking models in order to generate more reasonable results in forecasting times series data, especially as demand grows for multivariate models.

Books And Articles

- Abdi, H.,Valentin, D.,Edelman, B. (1999), " Neural Networks (Sage University Papers Series on Quantitative Application in the Social Sciences", series no. 07 – 124. Thousand Oaks, CA: Sage

- Aiken, Milam (1999), "Using a Neural Network to Forecast Inflation", Industrial Management and Data Systems, Volume 99 (pp. 296 - 301)

- Amir F. Atiya, Suzan M. El-Shoura, Samir I. Shaheen, Mohamed S. El-Sherif (1999), " A Comparison Between Neural-Network Forecasting Techniques-Case Study: River Flow Forecasting", IEEE Transactions on Neural Networks, Vol.10, NO. 2, MARCH 1999

- Baum E. B. and Haussler D. (1998), " What size net gives valid generalization?", Neural Computation, 1, pp. 151- 160.

- Chambers J. M., "Computing with Data: Concepts and Challenges", The American Statistician, 53:1 (1999), pp. 73-84. (Web version is extended from the journal version.)

- Chatfield C. (1993), "Neural networks: Forecasting Breakthrough of Passing Fad?" International Journal of Forecasting 9, pages 1-3

- Cheung Y., Huang R., "A Divide-And-Conquer Fast Implementation of Radial Basis Function Networks With Application To Time Series Forecasting", Neural Processing Letters, Vol. 21, No. 3. (June 2005), pp. 189-206.

- Duda, R. O. and Hart, P. E. (1973), "Pattern Classification and Scene Analysis", Wiley, New York

- Elman J. , "Finding Structure in Time", University of California, *Cognitive Science, 1990*, 14(2):179–211

- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problem. Annals of Eugenices, 7, 179–188.

- Gentle, J. E. "Singular Value Factorization." §3.2.7 in Numerical Linear Algebra for Applications in Statistics. Berlin: Springer-Verlag, pp. 102-103, 1998.

- Golub, G. H. and Van Loan, C. F. "The Singular Value Decomposition" and "Unitary Matrixes." §2.5.3 and 2.5.6 in Matrix Computations, 3rd ed. Baltimore, MD: Johns Hopkins University Press, pp. 70-71 and 73, 1996

- Gonzalez S. (2000), "Neural Networks for Macroeconomic Forecasting: A Complementary Approach to Linear Regression Models",Working Papers-Department of Finance Canada, Papers 2000-07, Department of Finance Canada

- Gradojevic N. and Yang J., Bank of Canada, Working Papers 2000-23 , "The Application of Artificial neural networks to Exchange Rate Forecasting: The Role of Market Microstructure Variables"

- Hertz,Krogh,Palmer. (1991),"Introduction to the Theory of Neural Computation", Addison-Wesley Publishing Company, Redwood City, CA, 1991

- Kohonen, T. (1977). Associative Memory: A System Theoretical Approach. New York: Springer

- Kolarik T. , Rudorfer G. (1994), "Time Series Forecasting Using Neural Networks", Holden-Day Inc.
- Kohzadi N. et al. "Neural networks for forecasting: An Introduction", Canadian Journal of Agricultural Economics, no.43, pp.463-474 (1995)
- Kuan, C.M. and H. White (1994), "Adaptive Learning with Nonlinear Dynamics Driven by Dependent Processes," Econometrica, Econometric Society, vol. 62(5), pages 1087-1114
- Lee, S.(1999) "Regularization in skewed binary classification." *Computational Statistics, 14 (2), 277–292*
- Lin F. ,Huo Yu X., Gregor S. and Irons R., "Time Series Forecasting with Neural Networks", Complexity International Journal , Volume 02, April 1995, $http://journal-ci.csse.monash.edu.au/ci/vol02/cmxhk/cmxhk.html$
- Luetkepohl, H. (1991), "Introduction to Multiple Time Series Analysis". Springer Verlag, NY, 500-503
- Maindonald J. and Braun J., "Data Analysis and Graphics Using R", Cambridge University Press, 2003
- Nash, J. C. "The Singular-Value Decomposition and Its Use to Solve Least-Squares Problems." Ch. 3 in Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation, 2nd ed. Bristol, England: Adam Hilger, pp. 30-48, 1990
- Nash J.C. and Nash M. (2003), " Practical forecast for managers", Oxford University Press Inc., New York

- Nash J.C. and M Walker-Smith (1995), "Nonlinear Parameter Estimation: An Integrated System in BASIC", Marcel Dekker, Inc. New York

- Niklasson L. and Van Gelder T. (1994), "Can Connectionist Models Exhibit Non-Classical Structure Sensitivity?" *In Proceedings of the Cognitive Science Society (pp. 664-669). Hillside, NJ: Erlbaum.*

- Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; and Vetterling, W. T. "Singular Value Decomposition." §2.6 in Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed. Cambridge, England: Cambridge University Press, pp. 51-63, 1992

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

- Ripley, B. D. (1996), "Pattern Recognition and Neural Networks", Cambridge University Press, 1996, ISBN 0-521-46086-7

- Sahin F. , "A Radial Basis Function Approach to a Color Image Classification Problem in a Real Time Industrial Application", Virginia Polytechnic Institute and State University, 1997

- Singh S., "A Long Memory Pattern Modeling and Recognition System for Financial Time-Series Forecasting", University of Exeter, *Pattern Analysis and Application, vol 2, issue 3, pp. 264-273, 1999*

- Sitte R. and Li Z., "Scaling for MEMS Virtual Prototyping: Size and Motion Dynamics Visualizations", WSCG (Short Papers) 2005: 37-40.

- Tkacz G. and Sarah Hu, Bank of Canada, Papers 1993-3: " Forecasting GDP Growth Using Artificial Neural Networks"
- Turner T., "A Beginner's Guide To Day Trading Online", Adams Media Corporation, 2000.
- Venables, W. N. and Ripley, B. D., "Modern Applied Statistics with S-plus",Springer. ISBN 0-387-98825-4, 1994
- Venables, W. N. and Ripley, B. D., "Modern Applied Statistics with S. Fourth edition", Springer. ISBN 0-387-98825-4, 2002
- Yan B. and Zivot E., "Analysis of High-Frequency Financial Data withS-Plus", Department of Economics, Insightful Corporation 2003

Web Sites

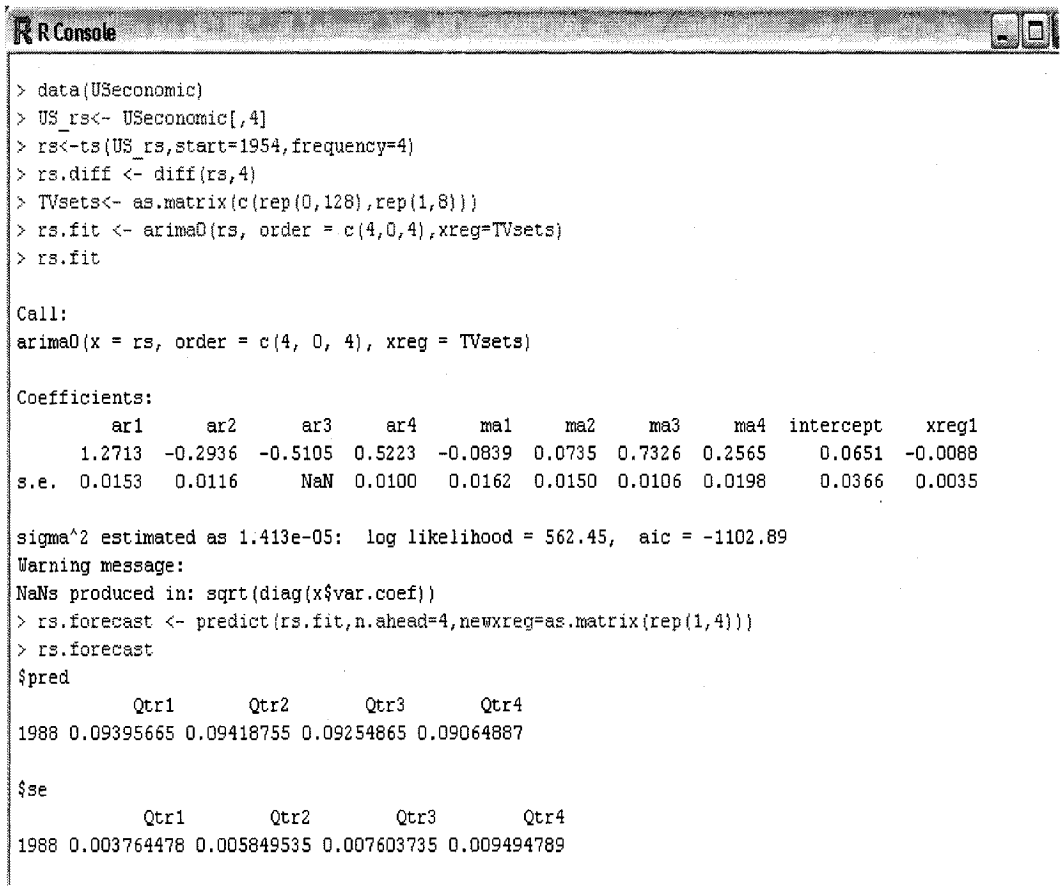
- Blais A. and Mertz D., An introduction to neural networks, $http://www-106.ibm.com/developerworks/library/l-neural$, date accessed: January 10, 2005
- Boersma P. and Weenink D., Principal Component Analysis, $http://www.fon.hum.uva.n$ date accessed: January 03,2005
- CyberTrader, Charles Shwab Company,$http://www.CyberTrader.com/$, date accessed: January 15, 2005
- Insightful Corporation, at $http://www.insightful.com/$date accessed: June 21, 2005

- LAPACK, Singular Value Decomposition,
  *http : //www.netlib.org/lapack/lug/node*53.*html*, date accessed: March 20,2005
- Market Information, *http : //www.forbes.com*,date accessed: June 12, 2005
- Paplinski A.P., Neural Networks, *www.csse.monash.edu.au*, date accessed: August 06, 2003
- QuantStudio Data Analysis and Trading Framework at
  http : //www.smartquant.com/references.php,
  date accessed: January 15, 2005
- SwiftTrade Inc.,*http : //www.swifttrade.com/*, date accessed: January 1, 2005

Appendix 1

Here you can find some of the examples of *R* Software explained in the thesis. The examples are based on *air passenger* and *USeconomic* (*rs*) data.

The following is the ARIMA(4,0,4) model used to forecast the four quarters of the discount rate on 91-day treasury bills, *rs*. Its lowest AIC value between all the ARIMA models described in the thesis has made ARIMA(4,0,4) the best model to forecast the non-seasonal data set.

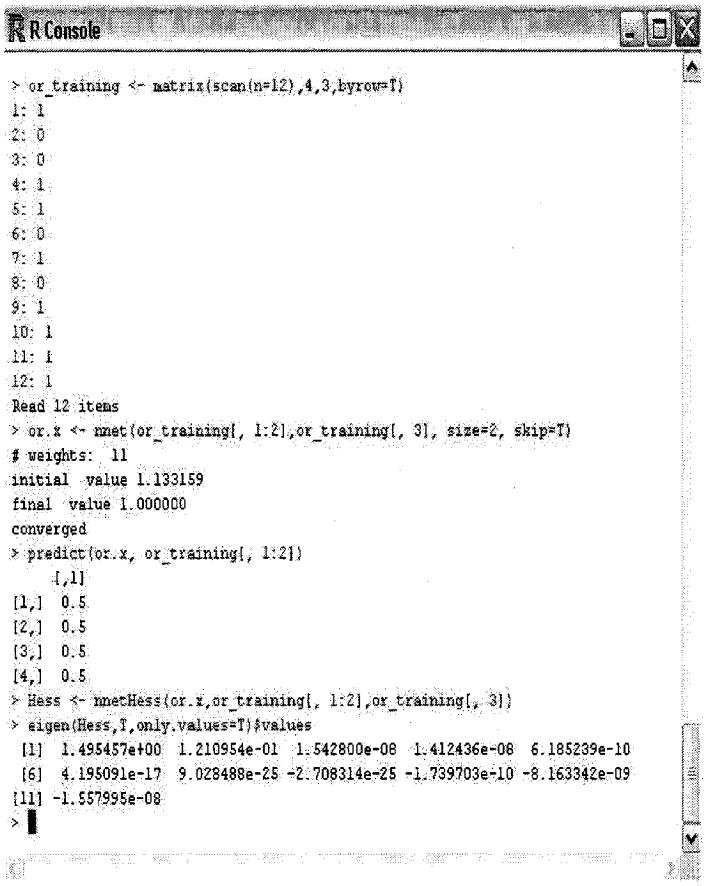


As shown in the next figure, we can use the Box-Ljung test to test the significance of autocorrelation at each lag. The significance value of Box-Ljung value should be less than or equal to .05 for all or almost all lags.

```
> AP<-AP.stl$time.series[,"trend"] + AP.stl$time.series[,"remainder"]
> x<-as.matrix(AP)
> Youtput<-x[13:128]
> set.seed(7777)
> XinputVars<- cbind(
+ x[12:127],
+ x[11:126],
+ x[10:125],
+ x[9:124],
+ x[8:123],
+ x[7:122],
+ x[6:121],
+ x[5:120],
+ x[4:119],
+ x[3:118],
+ x[2:117],
+ x[1:116]
+ )
>
> # build a 13-7-1 network with skip layer connections and linear output.
>
> ap.nn<-nnet(XinputVars,Youtput,size=7,linout=T,skip=T,maxit=1000,decay=1e-2,reltol=1e-7,abstol=1e-7,range=1.0)
# weights:  111
initial  value 7148581.637342
final  value 8967.783597
converged
>
> pr<-predict(ap.nn,XinputVars)
> y<-x[1:116]
> res<-y-pr
> Box.test(res,lag=12,type=c("Ljung-Box"))

        Box-Ljung test

data:  res
X-squared = 139.3696, df = 12, p-value < 2.2e-16
```

Appendix 2

Training the logical *OR* function can be implemented in *R*:



Training an OR Network using *R* ($R_{GUI} - output$)

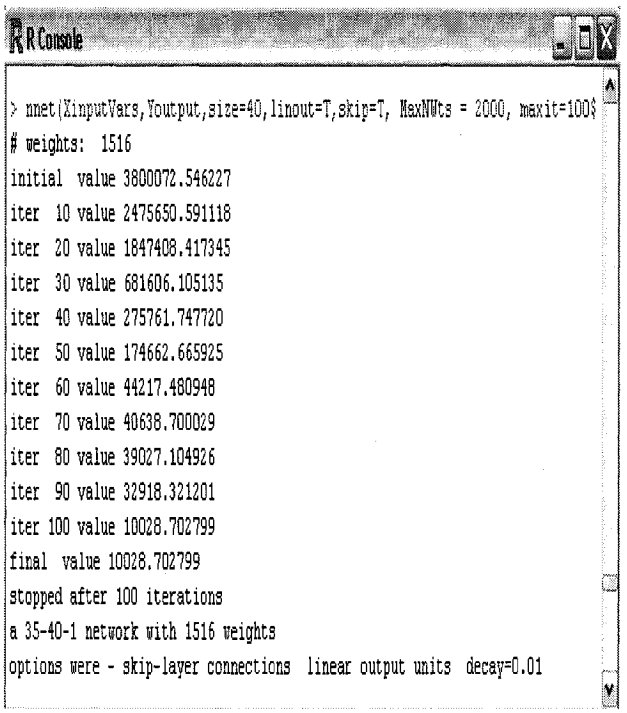
According to Abdi et al.(1999), since the *XOR* function is not linearly separable, hence it is not possible to train a perceptron to "learn" the *XOR* function (exclusive *OR*, that is x or y but not both).

Appendix 3

We use *nnetHess* function of *R* to solve the problems related to non convergence. *nnetHess* evaluates the Hessian or matrix of second derivatives of the specified neural network. This is normally called via argument *Hess=TRUE* to *nnet*.

- nnetHess(network, training data, classes for training data, weights used in the nnet fit)
- network is the object of class *nnet* as returned by *nnet* in *R*

One of the reason why a network does not converge may be related to the low iteration number.



Non-converged network ($R_{GUI} - output$)

As you can see, one of the obvious reasons for a network not to converge is to allocate a small number to its maximum iteration parameter (i.e. in *nnet*, *maxit*). At the above example *maxit*is initialized to 100 which is very small for this network to converge. NN needs to conduct more iterations to achieve a full convergence.

Appendix 4

Sample source code for Holt-winter's forecasting in R:

HW source code for Australian monthly beer production data ($R_{GUI} - output$)

Appendix 5

Sample source code for multi-step ahead Neural Networks forecasting in R:

R Console

```
> predict_nstep <- function(newdata, nstep, perceptron, hidden )
+ {
+   set.seed(7777)
+   Ndata<-as.matrix(newdata)
+   inp<-as.matrix(Ndata)
+   rvalue<-nrow(inp)
+   XPV<- as.matrix(inp[perceptron :(rvalue - 1)])
+   pPlus<-perceptron + 1
+   YO<-inp[pPlus:rvalue]
+   for(iter in 1:(perceptron - 1))
+   {
+
+   P<-(perceptron - iter)
+   R<-(rvalue - iter)
+   Xp<- cbind(XPV, inp[P:(R - 1)])
+   XPV<-as.matrix(Xp)
+   }
+
+   for(it in 1:nstep)
+      {
+
+   nn<-nnet(XPV,YO,size=hidden, MaxNWts =10000,maxit=10000,linout=T,skip=T, decay=1e-2
+
+   Oh<-predict(nn,Ndata[rvalue - perceptron:rvalue])
+   Ndata<-append(Ndata,Oh)
+       }
+   data<-as.matrix(Ndata)
+   print(paste("Number of Periods Forecasted is:", nstep))
+   rd<-nrow(data)
+   print(data[(rd - nstep + 1):rd])
+   par(mfrow = c(1,1))
+   plot(data,type="l",col=2)
+
+ }
>
```

Appendix 6

R sample source code for training the network by specific lags (Chatfield ,1993):

R Console

```
> trainlags <- function(data,lags,size,retry=1,maxit=10000,
+   trace=F,nntrace=F,...)
+  {
+    p <- length(lags)
+    maxlag <- max(lags)
+    n <- length(data)
+    x <- matrix(NA,n - maxlag, p)
+    for(i in 1:p) x[,i] <- data[(maxlag+1-lags[i]):(n-lags[i])]
+    y <- data[(maxlag+1):n]
+    HUGE <- 1e37
+    minval <- HUGE
+    rang <- 1/max(abs(data))
+    for(i in 1:retry){
+      g <- nnet(x,y,size=size, MaxNWts =100000, rang=rang,linout=T,maxit=maxit,trace=nntrace,...)
+      if (trace)
+        {
+  if (min(eigen(nnet.Hess(g,x,y))$val) < 0)
+    cat("Try ",i,": SS=",round(g$val,3),"(Possible non-minimum)\n")
+    else
+      cat("Try ",i,": SS=",round(g$val,3),"\n")
+        }
+      if ( g$val < minval)
+       {
+  gbest <- g
+  minval <- g$val
+        }
+    }
+    if ( minval == HUGE)
+      error("Minimum not found")
+    gbest$x <- x
+    gbest$y <- y
+    gbest$lags <- lags
+    if(trace)
+      cat ("Minimum SS = ",round(gbest$val,3)," in ",retry, "attempts\n")
+    structure(gbest, class = c("trainlags","nnet"))
```

Appendix 7

R sample source code for training the network by sequential data input:

R Console

```
> train <- function(newdata, validation,perceptron, hidden )
+ {set.seed(7777)
+ x<-as.matrix(newdata)
+ r<-nrow(x)
+ rval<-(r - validation)
+ XperceptronVars<- as.matrix(x[perceptron:(rval - 1)])
+ perceptronPlus<-perceptron + 1
+ Youtput<-x[perceptronPlus:rval]
+ valid<-as.matrix(x[rval:(r - 1)])
+ for(iter in 1:(perceptron - 1))
+ {
+ P<-perceptron - iter
+ R<-rval - iter
+ Xperceptron<- cbind(XperceptronVars, x[P:(R - 1)])
+ XperceptronVars<-as.matrix(Xperceptron)
+ Valid1<-cbind(valid, x[rval:(r - 1)])
+ valid<-as.matrix(Valid1)
+ }
+ sample<-nrow(XperceptronVars)
+ nn<-nnet(XperceptronVars,Youtput,size=hidden,MaxNWts =100000,maxit=10000,linout=T,skip=T$
+ proutput<- predict(nn,XperceptronVars)
+ residualSS<-sum((Youtput - proutput)^2)
+ print(paste("Residual sum of squares:", residualSS))
+ Oh<-predict(nn,valid)
+ fcast<-x[(rval + 1):r]
+ fsample<-nrow(fcast)
+ ab<-sum(abs(proutput - fcast)/ fcast) * 100
+ MAPE<- ab / fsample
+ Youtput[which(Youtput==0)]<- NA
+ PE<-((Youtput-proutput)/ Youtput) * 100
+ NMAPE<- mean(abs(PE),na.rm=TRUE)
+ print(paste("NMAPE",NMAPE))
+ F_residualSS<-sum((fcast - Oh)^2)
+ work<-data.frame(Youtput,proutput,PE)
+ return(work)
```