UNIVERSITI MALAYSIA PAHANG

BORANG PENGESAHAN STATUS TESIS♦

JUDUL: **CASE STUDY OF SHORT-TERM ELECTRICITY LOAD FORECASTING WITH TEMPERATURE DEPENDENCY**

SESI PENGAJIAN: **2009/2010**

Saya **TAI HEIN FONG (861106-56-6446)**
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/~~Sarjana~~ /~~Doktor Falsafah~~)* ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang (UMP).
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (√)

☐	**SULIT**	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	**TERHAD**	(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)
√	**TIDAK TERHAD**	

Disahkan oleh:

_____ (TANDATANGAN PENULIS)

_____ (TANDATANGAN PENYELIA)

Alamat Tetap:

22-2-11 JALAN 13/32,
TAMAN JINJANG BARU,
52000 KUALA LUMPUR,
WILAYAH PERSEKUTUAN.

DR AHMED N ABD ALLA
(Nama Penyelia)

Tarikh: **29 OCTOBER 2009**

Tarikh: : **29 OCTOBER 2009**

CATATAN:
* Potong yang tidak berkenaan.
** Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu

♦ Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).

CASE STUDY OF SHORT-TERM ELECTRICITY LOAD FORECASTING WITH TEMPERATURE DEPENDENCY

TAI HEIN FONG

This thesis is submitted as partial fulfillment of the requirements for the award of the Bachelor of Electrical Engineering (Power System) (Hons.)

Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

OCTOBER, 2009

"I hereby acknowledge that the scope and quality of this thesis is qualified for the award of the Bachelor Degree of Electrical Engineering (Control and Instrumentation)"

Signature : _____

Name : DR AHMED N ABD ALLA

Date : 29 OCTOBER 2009

Signature : _____

Author : TAI HEIN FONG

Date : 29 OCTOBER 2009

ACKNOWLEDGEMENT

First and foremost, I would like to express the deepest gratitude to my supervisor, Dr Ahmed N Abd Alla, for his continued support, encouragement, and guidance in overseeing the progress of my project from its initial phase till its completion. Without his valuable advices and comments, this would not have been possible to achieve a good basis of project.

Secondly, I would like to extend words of appreciation to all the lecturers for their friendly help and guidance throughout the process of completing this project.

To all our friends and course mates, thank you for believing in me and helping me. The experiences and knowledge I gained would prove invaluable to better equip me for the challenges lie ahead.

Last but definitely not the least to my parent, I can never thank enough for their love, and for supporting me throughout my studies in University Malaysia Pahang (UMP).

ABSTRACT

Load forecasting is very essential to the operation of electricity companies. It enhances the energy-efficient and reliable operation of a power system. This is a case study of short-term load forecasting using Artificial Neural Networks (ANNs). This load forecasting program gives load forecasts half an hour in advance. Historical load data obtained from the electricity generation company will be use. The main stages are the pre-processing of the data sets, network training, and forecasting. The inputs used for the neural network are one set of historical load demand data and five sets of temperature data. The neural network used has 3 layers: an input, a hidden, and an output layer. The input layer has 5 neurons, the number of hidden layer neurons can be varied for the different performance of the network, while the output layer has a single neuron.

ABSTRAK

System yang dapat menganggarkan keperluan kuasa elektrik untuk kegunaan harian merupakan ciri yang penting untuk sesebuah syarikat yang menghasilkan kuasa elektrik. System tersebut dapat membekalkan kuasa elektrik secara berterusan tanpa menghadapi sebarang masalah. Oleh itu, kajian kes ini telah dikaji dengan menggunakan Artificial Neural Network (ANNs) bagi menghasilkan system tersebut. Kajian ini akan dapat membuat anggaran keperluan kuasa elektrik untuk setiap setengah jam. Maklumat yang digunakan bagi anggaran tersebut akan didapati daripada syarikat yang menghasilkan kuasa elektrik. Proses yang teribat dalam kajian ini terbahagi kepada tiga. Proses tersebut ialah proses mengumpul data, latihan Neural Network dan akhir sekali ialah proses anggaran kuasa elektrik. Maklumat yang digunakan ialah satu set kegunaan kuasa elektrik untuk setiap setengah jam yang lama dan lima set maklumat suhu. Neural network ini mengandungi tiga lapisan, input, hidden dan output. Lapisan input mengandungi lima neurons, untuk lapisan hidden, bilangan neurons adalah bergantung kepada pengguna dan akhir sekali ialah lapisan output yang mengandungi satu neuron.

TABLE OF CONTENTS

TITLE	PAGE
TITLE PAGE	i
DECLARATION	ii
ACKNOWLEDGEMENT	iv
ABSTACT	v
ABSTRAK	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	xi
LIST OF TABLES	xiii
LIST OF APPENDIX	xiv
CHAPTER 1: INTRODUCTION	**1**
1.1 : Introduction	1
1.2 : Project Objectives	3
1.3 : Thesis Outline	4
CHAPTER 2: FORECASTING METHODS	**6**

2.1: Important Factors for Forecasts 6
2.2: Short-term load forecasting methods 7
2.2.1: Similar-day approach 8
2.2.2: Regression methods 8
2.2.3: Time Series 8
2.2.4: Neural Network 8
2.2.5: Expert system 8
2.2.6: Fuzzy logic 8

CHAPTER 3: NEURAL NETWORK MODELS FOR TIME SERIES FORECAST 13

3.1: Introduction 13
3.2: Important Factors for Electricity Load 15

CHAPTER 4: INTRODUCTION TO WAVELET TRANSFORM 17

4.1: Introduction 17
4.2: Fourier Transform 18
4.3: Short-Time Fourier Transform 19
4.4: Wavelet transform (WT) 20
4.4.1: Continuous Wavelet Transform 21
4.4.2: Discrete Wavelet Transform 22
4.4.3: Non-Decimated Wavelet Transform 23
4.5: Comparison between Fourier and Wavelet transform 25

CHAPTER 5: NEURAL NETWORKS IN TIME-SERIES FORECAST **31**

5.1: Introduction to Neural Network 27
5.2: Artificial Neural Network 28
5.3: The human brain 30
5.4: Artificial Neurons 32
5.4.1: McCulloch-Pitts Neuron 33
5.4.1: McCulloch-Pitts Neuron 33
5.4.1: The Perceptron 37
5.5: The Perceptron Network 38
5.6: Supervised Learning Algorithms 41
5.6.1: Widrow-Hoff Algorithm 42
5.6.2: Back-propagation Algorithm 43
5.7: Classification of Neural Networks 47

CHAPTER 6: THE PROPOSED FORECAST MODEL **49**

6.1: Preliminary Forecast Model 49
6.1.1: Stage1: Pre Signal Processing 50
6.1.2: Stage2: Signal Prediction 51
6.1.3: Stage3: Post Signal Processing 52
6.2: Finalizing the Design 52
6.2.1: Number of Input Nodes and Output Neurons 53
6.2.2: Number of Hidden Neurons 53
6.2.3: Learning Algorithm 54
6.3: Final Proposed Forecast Model 54

CHAPTER 7: SIMULATION AND RESULTS **56**

7.1: Methodology 56
7.2: Simulations 62
7.2.1: Finding the Suitable Resolution Level 62
7.2.2: Setting the Number of Hidden Neurons 63
7.3: Results 64
7.4: Matlab GUI 71
7.5: Discussions 75

CHAPTER 8: CONCLUSION AND RECOMMENDATIONS **76**

8.1: Conclusion 76
8.2: Recommendations for Future Work 77

REFERENCES **78**

APPENDIX A **81**

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
1	Wavelet Enhanced Neural Network Model	14
2	Signal decomposition in time domain	15
3	Fourier Transform of a Time Series Signal	18
4	Short-Time Fourier Transform of a Time Series Signal	19
5	Wavelet Transform of a Time Series Signal	20
6	à Trous Wavelet Transform of a Time-Series Signal	24
7	Comparison of the Transformation Techniques	26
8	Neural Network	28
9	A neural network is an interconnected group of nodes, akin to the vast network of neurons in the human brain	29
10	Elements and connectivity of Neural Network	31
11	Structural Levels of Organization in the Brain	31
12	Linear Threshold Unit	35
13	OR Function	36
14	AND Function	36
15	Single-layer Perceptron	38
16	Modified Linear Threshold Unit	40
17	Block Diagram of Supervised Learning	42
18	The Widrow-Hoff Approach	43
19	Multi-layer Perceptron with Two Hidden Layers	44

20	Flow of Signals with the Back-propagation Algorithm	45
21	Classifications of Neural Network	47
22	Three Stages of the Preliminary Forecast Model	49
23	Wavelet Decomposition Process	51
24	Wavelet Recombination Process	52
25	Final Proposed NN Forecast Model	55
26	Comparison of Different Resolution Levels	62
27	Decomposition of Queensland Demand Series	64
28	Decomposition of Brisbane Temperature Series	64
29	Decomposition of Gold Coast Temperature Series	65
30	Decomposition of Townsville Temperature Series	65
31	Decomposition of Cairns Temperature Series	66
32	Decomposition of Rockhampton Temperature Series	66
33	Load Forecasted for 1 day (48 points)	67
34	Load Forecasted for 2 days (96 points)	67
35	Load Forecasted for 3 days (144 points)	68
36	Load Forecasted for 4 days (192 points)	68
37	Load Forecasted for 5 days (240 points)	69
38	Load Forecasted for 6 days (288 points)	69
39	Load Forecasted for 1 week (336 points)	70
40	The Main window	71
41	The Introduction	71
42	Use1	72
43	Use2	73
44	Use3	73
45	Use4	74
46	Credit	74

LIST OF TABLES

TABLE NO.	TITLE	PAGE
1	Truth-table for the OR Function	36
2	Truth-table for the AND Function	36
3	Summary of Various Conjugate Gradient Algorithms	47
4	The Mean Absolute Error obtained from the forecasting neural network model	75

LIST OF APPENDIXES

APPENDIX NO.	TITLE	PAGE
A	Program for Short-term Load Forecasting System	81

CHAPTER 1

INTRODUCTION

1.1 Introduction

A reliable and continues supply of electrical energy is necessary for the functioning of today's complex societies. Due to the increasing consumption and obstruction of various kinds, and the extension of existing electrical transmission networks, power systems operated closer and closer to their limits thus the chances of occurrences overloading, equipment failures and blackout increases. Moreover, another problem to be faced is that the electrical energy cannot be stored whereby the electricity is only generated when needed. As a result from the electricity supply and demand fluctuating, an accurate model for electric power load forecasting is essential to the operation and planning of electricity generation in order to provide an effective and reliable operation.

Load forecasting predict the demand for electricity over the planning period of time for the utility in planning generation schedules in a power system where it helps on making decision on when and how much electricity need to be generated. Load forecasts

are extremely important for energy suppliers, ISOs, financial institutions, and other participants in electric energy generation, transmission, distribution, and markets. Load forecasts can be divided into three categories: short-term forecasts which are usually from one hour to one week, medium forecasts which are usually from a week to a year, and long-term forecasts which are longer than a year. However, short-term load forecasting is more accurate than medium and long term load forecasting due to the several factors.

For short-term load forecasting several factors should be considered, such as time factors, weather data, and possible customers' classes. However according to the electric load prediction survey [1] published, it indicated that of the 22 research reports considered, 13 made use of temperature only, 3 made use of temperature and humidity, 3 utilized additional weather parameters, and 3 used only load parameters which means that the most important factor that decides on the accuracy of the result of short term load forecasting is the temperature. This highlighted that temperature will directly affected the accuracy of the short term load forecast model.

In this case, Artificial Neural Network (ANN) is used to obtain the short term load forecast model with the historical load demand data and temperature as the input parameters.

1.2 Project Objective

The main objective of this project is to propose a short-term load forecasting model with high forecasting accuracy based on the historical data of load demand by using the Artificial Neural Networks (ANN). This project is very usable in enhance the energy efficiency and reliability of a power system generation plant. The proposed short-term load forecasting model is ensuring adequate electricity generation from the power system generation plant to meet the consumers' demand and thus providing effective operation. The factors that need to be concern in the proposed short-term load forecasting model are the time and the temperature factors.

This project concentrates on the using of Artificial Neural Network (ANN) in creating the short-term load forecasting model. The scope of the project involved three stages which are summarized as followed:

i. Preliminary stage
   Step 1: Data Pre-processing
   Step 2: Data Prediction
   Step 3: Data Post-processing
ii. Network training
iii. Forecasting

1.3 Thesis Outline

The thesis is orderly organized into 6 chapters and they are outlined as below:

Chapter 1 explains the important in propose a short-term load forecasting model with high forecasting accuracy based on the historical data of load demand by using the Artificial Neural Networks (ANN). It also outlines objective and scope of this project.

Chapter 2 describes the forecasting methods used in the project. It gives a brief review of the important factors that need to be considering in load forecasting. Few short-term load forecasting methods such as similar-day approach, regression methods, time series, neural network, expert system and fuzzy logic are discussed.

Chapter 3 provides description and discussion on the neural network models for time series forecast. There are three important process involved in proposing the neural network models which are preliminary stage, network training and forecasting.

Chapter 4 indicates the introduction to Fourier and Wavelet transform. For Fourier transform, short-time Fourier transform is discussed. However for Wavelet transform, it consists of continuous, discrete and non-decimated.

Chapter 5 presents neural networks in time-series forecast. It discussed about the concept of artificial neural network and how it works. In additional, it also discussed about the perceptron network.

Chapter 6 gives the overview of the proposed forecast model that summarizes all the stages involved.

Chapter 7 shows the results and simulation obtained from the project.

Lastly, Chapter 8 is used for the conclusion and recommendations.

CHAPTER 2

FORECASTING METHODS

2.1 Important Factors for Forecasts

Generally, there are two different categories of forecasting models which are the traditional models and the modern technique.

Traditional forecast model employ time series and regression analysis through the use of statistical models such as peak load models and load shape models[8][9]. An example of such models is the Autoregressive Moving Average (ARMA model). ARMA model is a tool for understanding and perhaps predicting the future values of time series data especially in statistical and signal processing. These models are mostly linear methods and have limited ability to capture non-linearity in the load time series pattern. Therefore, it is much complex to operate the traditional forecast model. For modern techniques such as neural networks, fuzzy logic, and expert systems, it is known that these technique can be use to operate the load forecast model more effectively and accurately.

Since load forecasts can be divided into three categories: short-term forecasts which are usually from one hour to one week, medium forecasts which are usually from a week to a year, and long-term forecasts which are longer than a year, therefore this means that for each of the categories, there will be the most appropriate methods to operate the forecast models. First for the medium- and long-term forecasting, the so-called end-use and econometric approach are broadly used. Whereas, a variety of methods, which include the so-called similar day approach, various regression models, time series, neural networks, statistical learning algorithms, fuzzy logic, and expert systems, have been developed for short-term forecasting.

From the research, we know that there are a large variety of mathematical methods and ideas have been used for load forecasting. When the time goes on, the development and improvements of appropriate mathematical tools will lead to the development of more accurate load forecasting techniques thus more effective forecast models can be proposed.

However, the accuracy of load forecasting depends not only on the load forecasting techniques, but it also depends on few other factors such as weather condition and class of customers at that certain areas (e.g. residential, commercial, or industrial). Among all the factors, weather plays an important role in determining the load forecast model. Weather forecasting is an important issue where it affecting the pattern of the load demands indirectly thus the data of the temperature for a long period of time need to be obtained in proposing the load forecast models.

2.2 Short-term load forecasting methods

There is variety of statistical and artificial intelligence techniques have been developed for short-term load forecasting for example similar-day approach, regression

methods, time series, expert system and fuzzy logic. Each of the methods will be discuss in detail in the next session.

2.2.1 Similar-day approach

This approach is based on searching historical data for days within one, two, or three years with similar characteristics to the forecast day. Similar characteristics include weather, day of the week, and the date. The load of a similar day is considered as a forecast. Instead of a single similar day load, the forecast can be a linear combination or regression procedure that can include several similar days. The trend coefficients can be used for similar days in the previous years.

2.2.2 Regression methods

Regression is the one of most widely used statistical techniques. For electric load forecasting regression methods are usually used to model the relationship of load consumption and other factors such as weather, day type, and customer class. Engle *et al.* [7] presented several regression models for the next day peak forecasting. Their models incorporate deterministic influences such as holidays, stochastic influences such as average loads, and exogenous influences such as weather. References [12], [16], [10], [3] describe other applications of regression models to loads forecasting.

2.2.3 Time Series

Time series can be defined as a sequential set of data measured over time, such as the hourly, daily or weekly peak load. The basic idea of forecasting is to first build a

pattern matching available data as accurate as possible, then obtains the forecasted value with respect to time using established model.

Generally, series are often described as having following characteristic [5]:

$$X(t) = T(t) + S(t) + R(t) \qquad t = \ldots -1,0,1,2,\ldots$$

Here, T(t) is the trend term, S(t) the seasonal term, and R(t) is the irregular or random component (which can be generated using Matlab command). At this moment, we do not consider the cyclic terms since these fluctuations can have a duration from two to ten years or even longer which is not applicable to short-term load forecasting.

We have such assumptions to make things a little easier for the moment:

1) The trend is a constant level;
2) The seasonal effect has period **s**, that is, it repeats after **s** time periods. Or the sum of the seasonal components over a complete cycle or period is zero.

$$\sum_{j=1}^{s} S(t+j) = 0$$

2.2.4 Neural Network

The use of artificial neural networks (ANN or simply NN) has been a widely studied electric load forecasting technique since 1990 [15]. Neural networks are essentially non-linear circuits that have the demonstrated capability to do non-linear curve fitting. Artificial Neural Networks are mathematical tools originally inspired by the way human brain process information. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation.

In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. In more practical terms neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. The outputs of an artificial neural network are some linear or nonlinear mathematical function of its inputs. The inputs may be the outputs of other network elements as well as actual network inputs. In practice network elements are arranged in a relatively small number of connected layers of elements between network inputs and outputs. Feedback paths are sometimes used. Before the use of artificial intelligence networks, the method of architecture needs to be determined.

The most popular artificial neural network architecture for electric load forecasting is back propagation. With the ability to approximate any continuous nonlinear function, the back propagation network has extraordinary forecasting abilities. Artificial neural networks can be use in conjunction with other forecasting techniques such as with regression trees, time series or fuzzy logic to perform a more accurate load forecasting.

2.2.5 Expert system

Expert system makes use of rules and procedures used by human experts in the field of interest into software that is then able to automatically make forecasts without human assistance. This means that basically Expert system forecasts the load according to rules extracted from experts' knowledge and operators' experience.

Expert system use began in the 1960’s for such applications as geological prospecting and computer design. Expert systems work best when a human expert is

available to work with software developers for a considerable amount of time in imparting the expert's knowledge to the expert system software. Also, an expert's knowledge must be appropriate for codification into software rules (i.e. the expert must be able to explain his/her decision process to programmers). An expert system may codify up to hundreds or thousands of production rules. Foe example, Ho *et al.* [11] proposed a knowledge-based expert system for the short-term load forecasting of the Taiwan power system. Operator's knowledge and the hourly observations of system load over the past five years were employed to establish eleven day types. To improve the load forecasting result and lower down the forecasting error, operators at a particular site need to be consulted. This method is promising, however, it is important to note that the expert opinion may not always be consistent, and the reliability of such opinion may be in question.

2.2.6 Fuzzy logic

Fuzzy logic is a generalization of the usual Boolean logic used for digital circuit design. An input under Boolean logic takes on a truth value of "0" or "1". Under fuzzy logic an input has associated with it a certain qualitative ranges. For instance a transformer load may be "low", "medium" and "high". Fuzzy logic allows one to (logically) deduce outputs from fuzzy inputs. In this sense fuzzy logic is one of a number of techniques for mapping inputs to outputs (i.e. curve fitting).

Among the advantages of fuzzy logic are the absence of a need for a mathematical model mapping inputs to outputs and the absence of a need for precise (or even noise free) inputs. With such generic conditioning rules, properly designed fuzzy logic systems can be very robust when used for forecasting. Of course in many situations an exact output (e.g. the precise 12PM load) is needed. After the logical processing of fuzzy inputs, a "defuzzification" process can be used to produce such precise outputs.

References [13], [14], [17] describe applications of fuzzy logic to electric load forecasting.

Among all the methods discuss as above, Artificial Neural Network (ANN) is considered as the most appropriate method to propose the short-term load forecasting model. This is because in practical terms, neural network are non-linear statistical data modeling tools where it can be used to model complex relationship between inputs and outputs or to find the patterns in data. Thus, the load forecast model is more accurate with lesser error.

CHAPTER 3

NEURAL NETWORK MODELS FOR TIME SERIES FORECAST

3.1 Introduction

I. Preliminary stage
   Stage 1: Data Pre-processing
   Stage 2: Data Prediction
   Stage 3: Data Post-processing
II. Network training
III. Forecasting

First and foremost, the latest historical data of load demand for year 2009 are collected from Australian Energy Market Operator (AEMO) for a period of time. The historical data of load demand are shown for every half an hour daily and up to one month. Since the short-term forecast requires knowledge of the load from one hour up to a few days therefore information derived from the short-term load forecasts are vital to the system as operations in terms of short-term unit maintenance work, weekly, daily, and hourly load scheduling of generating units, and economic and secure operation of

power systems .Those historical data of load demand obtained will be in time series function, which means the signal, is measured as a function of time. For a pure neural network (NN) model, it takes in a set of input data without undergoing the pre and post-process step done. This means that the time series function signal is directly fed into the neural network and outputting as a predicted data. However, the forecast model produced is not accurate due to the hidden pattern of noise in the raw data input. To solve this problem, wavelet transform technique is introduced.

Since the raw data obtained is non-linear and it may consist of noisy time series pattern, which will affect the accuracy of the forecast model's prediction therefore the data will be processed using the wavelet transform technique to extract out the noise pattern. The wavelet transform technique is consisting of three stages.

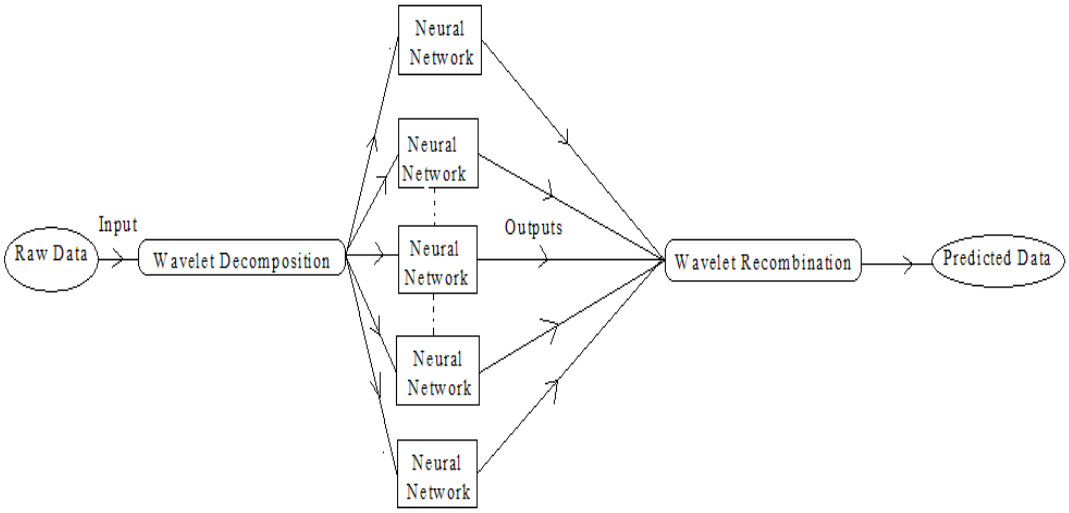

Figure 1: Wavelet Enhanced Neural Network Model

First the data will need to undergo decomposition where the data will be broken into sinusoids of different frequency instead of coefficients. Each of the decomposed scale is fed into one neural network as its input for training as well as for the prediction

in the next process. Finally, the decomposed scales at the output of neural network are recombined using wavelet recombination technique to obtain the predicted data.

The wavelet transform technique can be easily implemented using the MATLAB computing software which means the model is part of the simulation program.

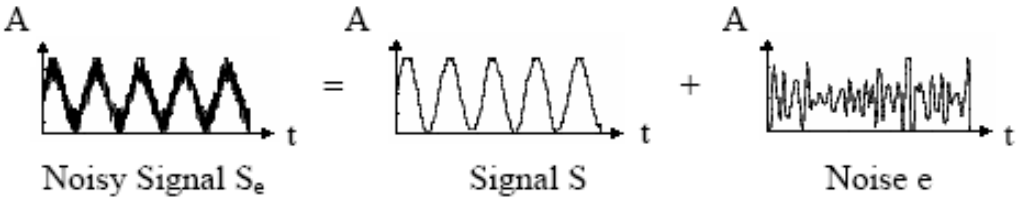

Figure 2: Signal decomposition in time domain

For the network training session, the supervised training is used to train the neural network. The network is presented with both the inputs as well as the expected output to train each of the neural networks. Then the expected output is used to compare with the actual output from the network.

Last but not least, for the forecasting part, the final proposed Neural Networks Forecast Model will be obtained.

3.2 Important Factors for Electricity Load

As mentioned earlier, there are many factors that have a close relationship with electricity load demand such as the time factors, weather data, and possible customers' classes. Among all the factors, the time factor is the main factor that play an important

role important in short term load forecasting. The time factors include the time of the year, the day of the week and the hour of the day. There are important differences in load between weekdays and weekends. The load on different weekdays also can behave differently from time to time. For example, morning time from Monday to Friday have structurally different loads than night time from Monday to Friday.

The temperature is also one of the factors that have a close relationship with electricity load demand. This can be easily seen that it is commonly for the electricity to increase during the hot weather in Malaysia. When the temperature is high, the usage of air conditioning will increase thus increasing of load demand. Since weather has been proven to have impact on the load demand, therefore it need to be considered in proposing the short term load forecast model where the data of the temperature forecasting need to be obtained. However, the temperature factor has les effect to the short term load forecasting model if compared to the time factor.

To achieve the objective of this project, we need to increase the accuracy of this short term load forecasting model, therefore wavelet transform technique has been used to extract the noise so that to increase the accuracy of the load forecast model. However, with the additional temperature data as the data input, the proposed forecast model will then become more accurate.

CHAPTER 4

INTRODUCTION TO WAVELET TRANSFORM

4.1 Introduction

Since the historical data of load demand obtained is in time series function with hidden noise pattern, therefore it need to undergo wavelet transform to extract out the hidden pattern of the time series function before it is being fed into the neural network.

Generally there are two popular mathematical transformations methods that can be applied to extract useful information from the time series function signal. Among the two methods, there are few mathematical transformations methods such as Fourier Transform, Short Time Fourier Transform, Wavelet Transform, Continuous Wavelet Transform, Discrete Wavelet Transform, and Non-Decimated Wavelet Transform that can extract out the hidden pattern of noise from the signal function. However, to decide which method will be the best use, it is depends on the output of the signal needed to obtain. So, it is beneficial for us to understand the usage of all the methods before making decision on with transformation method to be use.

4.2 Fourier Transform

Fourier transform is probably the most popular mathematical transformation that is applied to exact useful information that is hidden within a raw time series signal (time series signal in its original form) [1].

Figure 3: Fourier Transform of a Time Series Signal
(Figure obtained from MATLAB help file)

As shown in Figure 3, Fourier analysis is used to transform a time series signal to its frequency domain. This transformation is important as it provides a signal's frequency content for analysis. Nevertheless, FT has a serious drawback with the problem of localization [2] [1]. Time information is lost whenever the time series signal is transformed to the frequency domain. That means we can no longer tell when a particular event occurs. Thus FT is more suitable only for stationary signals where time is not of concern with the desired signal.

4.3 Short-Time Fourier Transform

Short-Time Fourier Transform (STFT) is a technique that corrects the deficiency of Fourier Transform (FT) by implementing an additional windowing technique that allows only a small section of the signal to be analyzed within a particular time frame. Figure 4 illustrates the STFT of a time series signal.

Figure 4: Short-Time Fourier Transform of a Time Series Signal
(Figure obtained from MATLAB help file)

From Figure 4, it shown that the transformed STFT signal is both a function of time and frequency. The signal within the given window size is equally segmented and from each of the segmented signal, the frequency components are extracted. The segmented signals are assumed to be stationary and are multiplied by a window function. Hence, the mathematical expression for a STFT signal can be described as shown in equation below.

$$STFT_{mn}(t,f) = \int_{-\infty}^{\infty} e^{-jw_0} \cdot f(t) \cdot g(t - nt_0)dt$$

Where $f(t)$ and $g(t - nt_0)$ denote the signal to be transformed and the window function respectively.

Although the STFT technique is able to provide some information of time in the transformed signal, it has a limitation of a fixed window size for all frequencies. The information is obtainable only with a limited precision as determined by the window size and this is undesirable as it causes problems related to the time-frequency resolutions. To resolve the problem, a variable window size is required and this leads to the implementation of wavelet transform which will be discussed shortly.

4.4 Wavelets transform (WT)

Wavelet transform (WT) is a scalable windowing technique. The adjustable window size allows the use of long time intervals when more precise low-frequency information is desired and short time intervals when desiring high-frequency information. Figure 4 illustrates the WT of a time series signal.

Figure 5: Wavelet Transform of a Time Series Signal
(Figure obtained from MATLAB help file)

Wavelet analysis adopts the concept of scale and link between scale and frequency. It breaks the signals into shifted scaled versions of the original wavelet as compared to Fourier analysis, where signals are broken into sinusoids of different

frequencies instead. In addition to that, the technique uses a time-scale region instead of a time-frequency region.

4.4.1 Continuous Wavelet Transform

Continuous wavelet transform (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted version of the wavelet function Ψ. The results of CWT are many wavelet coefficients C, which are a function of scale and position. The mathematical expression for a given signal $f(t)$ is described in equation below. The summation terms on the left and right of equation below represents the approximation and the detail respectively.

$$C(scale, position) = \int_{-\infty}^{\infty} f(t)\psi(scale, position, t)dt$$

CWT uses a fully scalable window that is shifted across the entire signal. CWT can operate at every scale, from that of the original signal up to some maximum scale that you determine by trading off your need for detailed analysis. The CWT is also continuous in terms of shifting: during computation, the analyzing wavelet is shifted smoothly over the full domain of the analyzed function. By iterating the shifting process many times with different scales (window size), a collection of time-frequency representations of the signal with different resolutions is obtained.

The most important step that involve in CWT is to calculate a number, C, that represents how closely correlated the wavelet is with this section of the signal. The higher C is, the more the similarity. More precisely, if the signal energy and the wavelet energy are equal to one, C maybe interpreted as a correlation coefficient. However, CWT is impracticability for implementation as calculating its wavelet coefficients at

every possible scale is an extremely tedious task, which may result in massive amount of data to be generated as well even for a computer to perform the task. To solve this problem, Discrete Wavelet Transform is introduced.

4.4.2 Discrete Wavelet Transform

The computation of wavelet coefficients for discrete wavelet transform (DWT) makes use of the scales and position based on power of two. The DWT algorithm is capable of producing coefficients of fine scales for capturing high frequency information, and coefficients of coarse scales for capturing low frequency information. Therefore the Discrete Wavelet Transform (DWT) is easy to implement and reduces the computation time and resources required. In CWT, the signals are analyzed using a set of basis functions which relate to each other by simple scaling and translation. In the case of DWT, a time-scale representation of the digital signal is obtained using digital filtering techniques. The signal to be analyzed is passed through filters with different cutoff frequencies at different scales. If we consider a mother wavelet function Ψ and for a given signal$f(t)$, a mathematical representation for a DWT can be expressed as

$$f(t) = \sum_{k} c_{j0,k}\phi_{j0,k}(t) + \sum_{j>j0}\sum_{k} w_{j,k}\,2^{\frac{j}{2}}\psi(2^{j}t-k)$$

Where j is the dilation or level index, k is the translation or scaling index, $ø_{j0,k}$ is a scaling function of coarse scale coefficients $c_{j0,k}$, $w_{j0,k}$ is the scaling function of detail (fine scale) coefficients and all functions of $\Psi(2^{j}\,t-k)$ are orthonormal.

Normally, the DWT is widely use for signal compression due to its ability to set a large portion of the coefficients to zero without any substantial loss of information. In addition, if more properties other than the stationary properties of a signal are desired,

DWT would definitely be a better option as compared with the traditional technique of Fourier Transform (FT).

Despite many advantages with DWT, its application in time series analysis is limited by the lack of translation invariance due to a fixed dyadic grid. One way to resolve the problem is to adopt the approach of Non-Decimated Wavelet Transform.

4.4.3 Non-Decimated Wavelet Transform

The non-decimated wavelet transform (NWT) can be used to overcome the problem encountered with DWT by using a redundant basis transformation in which it produces equal-length wavelet coefficients for each resolution level. There are two types of application for NWT which are the à trous transform and the mirror-bounded stationary wavelet transform. However for this thesis, the à trous transform is used.

The algorithm of the à trous transform is similar to DWT and the only difference is that à trous transform do not have decimation step. The basic implementation of this algorithm is rather simple. If we consider a given time series signal $c_0(k)$, the à trous transform is performed by passing the signal through a series of low pass filters h_l. The result obtained at the output of each filters is the approximation (low frequency information) coefficient series. The number of times to filter the signal depends on the highest resolution level determined for the filtering process. That is, if the highest resolution level set is n, the signal will be filtered n times with a chain of approximation coefficient series $c_n(k)$; obtained at each of the different resolution levels. The mathematical expression that describes this process is given as

$$c_j(k) = \sum_{l=0}^{L-1} h_l c_{j-1}(k + 2^{j-1} l)$$

Other than producing approximation coefficient series at level *j*, the process of à trous wavelet transform also generates the wavelet (high frequency information) coefficient series. Referring to equation below, the wavelet coefficient series is obtained by taking the difference between $c_{j-1}(k)$ and $c_j(k)$.

$$w_j(k) = c_{j-1}(k) - c_j(k)$$

The process of generating wavelet coefficient series is further illustrated with the block diagram as shown in Figure 6.

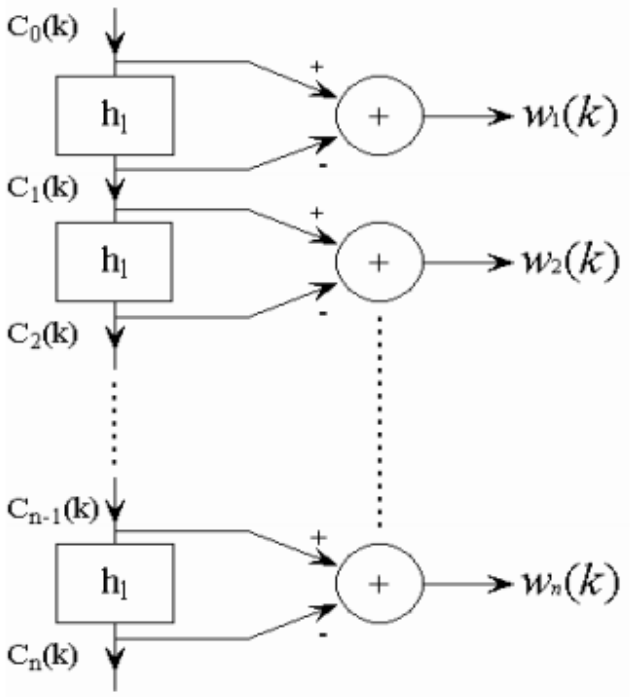

Figure 6: à Trous Wavelet Transform of a Time-Series Signal

Last but not least, the signal can be reconstructed using the mathematical expression as described in equation below

$$c_0(k) = c_n + \sum_{j=1}^{n} w_j(k)$$

4.5 Comparison between Fourier and Wavelet transform

Fourier analysis has a serious drawback. When a signal is transformed into the frequency domain, time information is lost. If it is mainly concerned with stationary signals, signals that don't change much over time, this drawback is not very important. Wavelet analysis is a windowing technique, similar to the STFT, with variable-sized windows. It allows the use of long time intervals, when more low frequency information is sought, and shorter regions, when more high frequency information is what you are after. Wavelet analysis is capable of revealing aspects of data that other signal analysis techniques miss, including aspects such as trends, breakdown points, discontinuities, and self- similarity. It is also often used to compress or de noise a signal without any appreciable degradation.

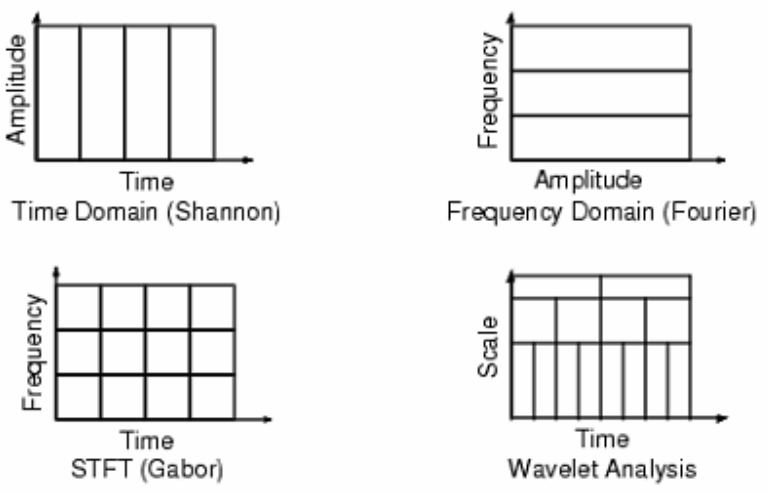

Figure 7: Comparison of the Transformation Techniques

(Figure obtained from MATLAB® help file)

CHAPTER 5

NEURAL NETWORKS IN TIME-SERIES FORECAST

5.1 Introduction to Neural Network

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements.

Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. Such a situation is shown below. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input or target pairs are needed to train a network. Neural networks have been trained to perform complex functions in various fields, including pattern recognition, identification, classification, speech, vision, and control systems.

Today neural networks can be trained to solve problems that are difficult for conventional computers or human beings. Throughout the toolbox emphasis is placed on neural network paradigms that build up to or are themselves used in engineering, financial, and other practical applications.

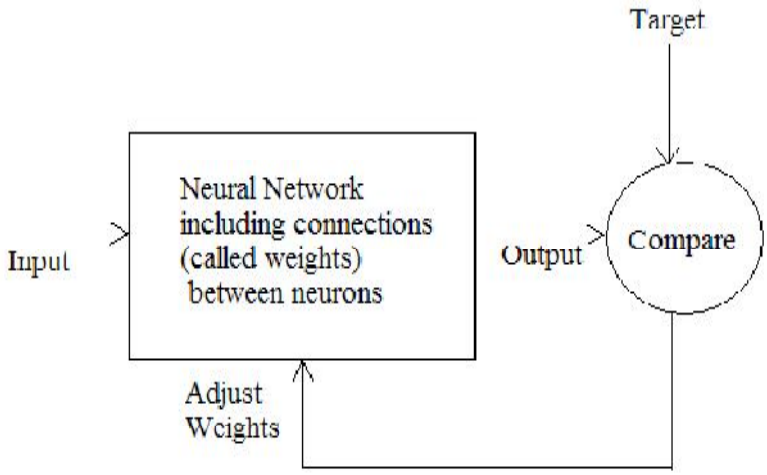

Figure 8: Neural Network

5.2 Artificial Neural Network

An artificial neural network (ANN), often just called a "neural network" (NN), is a mathematical model or computational model based on biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.

In more practical terms neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. In general, it can be say that the function of neural network is to produce a certain output pattern when it is fed by a certain input pattern.

Neural network adopt an approach of parallel processing. It consists of a network processor which known as neurons that operates in parallel. Thus, neural network is able to operate in fastest time. The operation of the neural network is similar to a human brain. To understand how the neural network works, we need to understand how the human brain works.

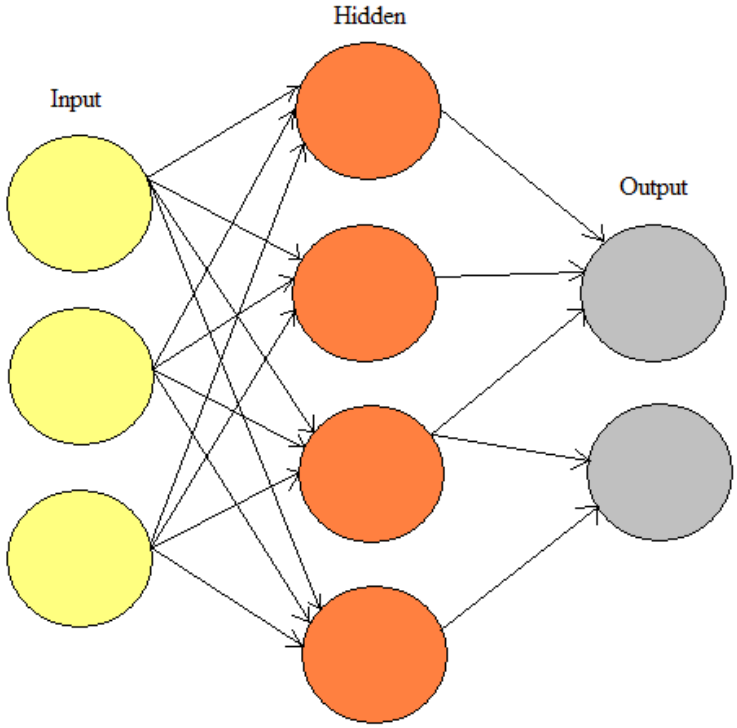

Figure 9: A neural network is an interconnected group of nodes, akin to the vast network of neurons in the human brain

5.3 The human brain

The human brain, an unparalleled pattern recognition system has approximately 10^{10} neurons; each on average may have as many as 10 connections to other neurons [18]. Neurons in the brain are described as processors where they communicate through a network of axons and synapses.

Firstly, the neurons take in information from the other neurons through the dendrites. Then when the information is sufficient, it will produce an output which is sent out from the cell body to the respective neurons through the axons where the links that provide information coupling from an axon to a dendrite are known as synapses. Synapses do not offer a direct linkage, but rather, they behave like a temporary chemical bond between the two elements instead. Forming of the chemical bond takes place when chemicals known as the neurotransmitters are released by the synapses. The coupling strength of this chemical bond is determined by the amount of neurotransmitters released. As such, the brain learns as the coupling strength varies, hence knowledge is acquired through the learning process.

Thus, the human brain can be considered to be densely connected electrical switching network conditioned largely by biochemical processes. Human brain's powerful capabilities in remembering, recalling, correlating, interpreting and reasoning have always have been a candidate for modeling and simulation. The vast neural network has an elaborate structure with very complex interconnection. This is illustrated in Figure 10 below [4]:

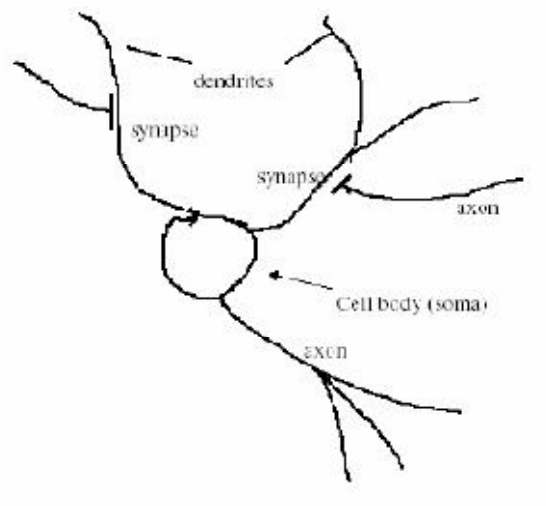

Figure 10: Elements and connectivity of Neural Network

After basic structure of a neuron has been discussed, we should have a quick view on the various structural levels of organization in the brain. The structural organization shows the unique characteristic of the brain and it is found that these unique characteristics are similar with today's modern digital computer. However, the artificial neurons that are used to build the neural networks are truly primitive as compared to those found in human brain.

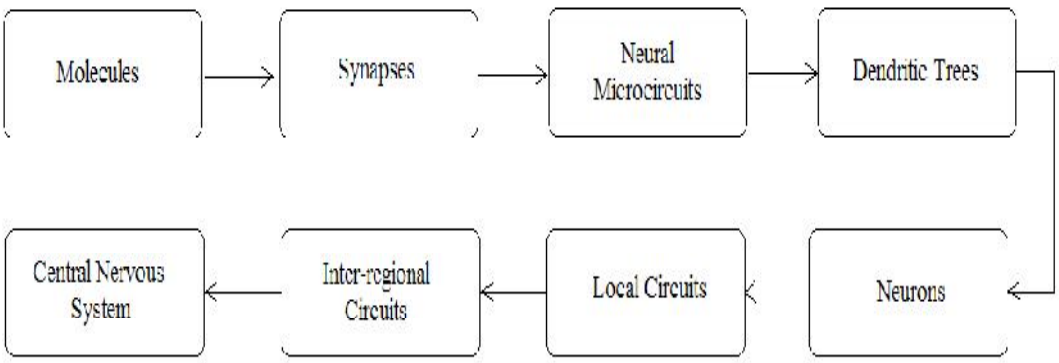

Figure 11: Structural Levels of Organization in the Brain

5.4 Artificial Neurons

Artificial neurons are ‘man made’ neurons that mimic the learning ability of a natural neuron. In practical, these artificial neurons are processing elements that are interconnected within the NNs. An artificial neuron also refers as a mathematical function conceived as a crude model or abstraction of biological neurons. Artificial neurons are the constitutive units in an artificial neural network. Depending on the specific model used, artificial neurons can have different names, such as semi-linear unit, Nv neuron, binary neuron, linear threshold function or McCulloch-Pitts neuron.

The artificial neuron receives one or more inputs (representing the one or more dendrites) and sums them to produce an output (synapse). It takes in information through its input lines and the weighted sum of these inputs is obtained by combining the information from all these inputs. An output from the artificial neuron is then generated based on a transfer function which depicts the relationship between the weighted sums of inputs and the output. The transfer functions usually have a sigmoid shape, but they may also take the form of other non-linear functions, piecewise linear functions, or step functions. They are also often monotonically increasing, continuous, differentiable and bounded. In general, the function of artificial neurons is similar to that of a neuron in the human brain. An example of the transfer function for an artificial neuron can be mathematically expressed as

$$y = \frac{1}{1 + e^{-k*x}}$$

Where y is the output of a neuron, x is weighted sum of inputs and k is the parameter of the neuron. This transfer function is commonly known as a sigmoid function [21].

5.4.1 McCulloch-Pitts Neuron

The first neural network was designed by Warren McCulloch and Walter Pitts in 1943. It is called McCulloch-Pitts (M-P) where it represents the very first attempt of an artificial neuron model. The weights on a McCulloch-Pitts neuron are set so that the neuron performs a particular simple logic function, with different neurons performing different functions. There are few requirements for McCulloch-Pitts neurons which are summarized as follows

- The activation of a McCulloch-Pitts neuron is binary. That is, at any time step, the neuron either fires (has an activation of 1) or does not fire (has an activation of 0).
- McCulloch-Pitts neurons are connected by directed, weighted paths.
- A connection path is excitatory if the weight on the path is positive; otherwise it is inhibitory. All excitatory connections into a particular neuron have the same weights.
- Each neuron has a fixed threshold such that if the net input to the neuron is greater than the threshold, the neuron fires.
- The threshold is set so that inhibition is absolute. That is, any nonzero inhibitory input will prevent the neuron from firing.
- It takes one time step for a signal to pass over one connection link

The M-P neuron is an artificial neuron with a fixed or otherwise known as hard threshold. The M-P neuron is also commonly referred to as a Linear Threshold Unit (LTU) and its operation can be described as shown in Figure 12.

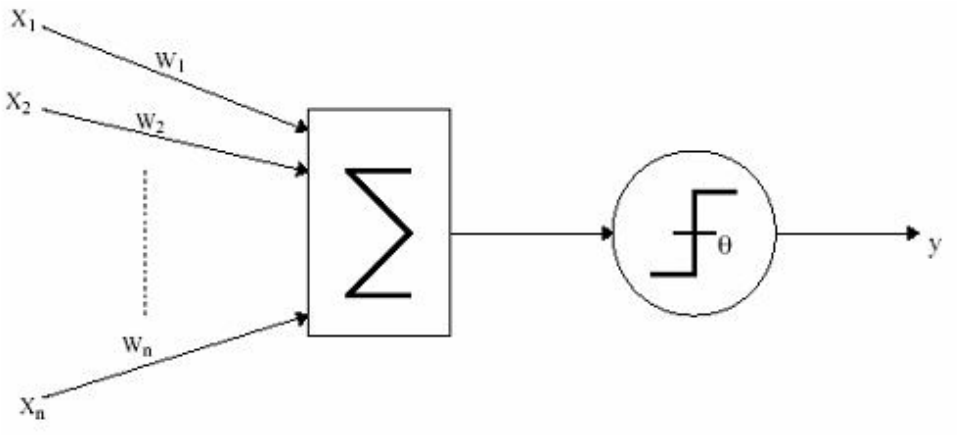

Figure12: Linear Threshold Unit

As depicted in Figure 12, from the left, x_n is the input, w_n is the input weight, $\sum$ is the weighted sum of inputs follow by the threshold, θ and finally y is represented as the output of the LTU. The model can also be mathematically expressed as

$$y = f\left[\sum_{i=1}^{n} w_i x_i - \theta\right]$$

Where f[z] is the fixed transfer function and $[\sum_{i=1}^{n} w_i - x_i - \theta]$ is the weighted sum of inputs. The transfer function depends on certain conditions to generate an output. Take the binary operation for example; the transfer function will generate an output based on these conditions:

$$f[z] = 1 \qquad ; z >= 0$$
$$f[z] = 0 \qquad ; z < 0$$

If for a bipolar operation, the transfer function will depend on the following conditions instead:

$$f[z] = +1 \qquad ; z <= 0$$
$$f[z] = -1 \qquad ; z > 0$$

For the case of a bipolar operation here, the transfer function is also called a step function. However, for the mathematical expression as above, it is more commonly known as the signum or sign function.

The M-P neurons are proven to be capable of solving simple Boolean functions like the logical AND and OR functions. However, its capability will not be proven here, but rather, is demonstrated instead. For an OR function, consider a LTU with an output *y* and two inputs, x_1 and x_2 that takes on the values of either 1 or 0. Then, an output value of 1 will be generated based on the following:

$$[\ (x_1 * w_1) + (x_2 * w_2)\ ] \geq \theta$$

In other words, equation above simply shows that the output will be a value of 0 unless the sum of weighted inputs is greater than the defined threshold value.

From Figure 13, it can be observed that the possible combinations of inputs are only 00, 01, 10 and 11. The results of the OR function is shown in the truth-table (refer to Table 1).

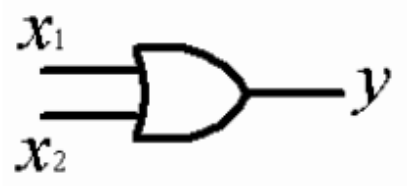

Figure 13: OR Function

x_1	x_2	θ	y
0	0	0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	0.5	1

Table 1: Truth-table for the OR Function

For the same set of input combinations, the AND operation can be performed by setting the threshold, θ to be 1.5. The truth-table for the resulting AND operation is shown in Table 2.

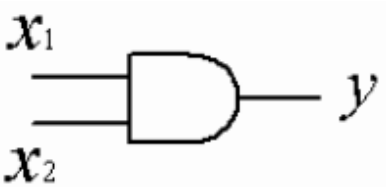

Figure 14: AND Function

x_1	x_2	θ	y
0	0	1.5	0
0	1	1.5	0
1	0	1.5	0
1	1	1.5	1

Table 2: Truth-table for the AND Function

However, the McCulloch-Pitts Neuron is unlike the biological networks, the parameters of their networks had to be designed, as no training method was available.

5.4.2 The Perceptron

In the late 1950s, Frank Rosenblatt and several other researchers developed a class of neural networks called perceptrons. The neurons in these networks were similar to those of McCulloch and Pitts. Rosenblatt's key contribution was the introduction of a learning rule for training perceptron networks to solve pattern recognition problems. He proved that his learning rule will always converge to the correct network weights, if weights exist then solve the problem. Learning was simple and automatic. Examples of proper behavior were presented to the network, which learned from its mistakes. The perceptron could even learn when initialized with random values for its weights and biases. Unfortunately, the perceptron network is inherently limited. These limitations were widely publicized in the book Perceptrons by Marvin Minsky and Seymour Papert. They demonstrated that the perceptron networks were incapable of implementing certain elementary functions. It was not until the 1980s that these limitations were overcome with improved (multilayer) perceptron networks and associated learning rules.

Today the perceptron is viewed as an important network. It remains a fast and reliable network for the class of problems that it can solve. In addition, an understanding of the operations of the perceptron provides a good basis for understanding more complex networks. Thus, the perceptron network and its associated learning rule are well worth discussion here. In the next section, we will define what we mean by a learning rule, explain the perceptron network and learning rule, and discuss the limitations of the perceptron network.

5.5 The Perceptron Network

The simplest form of a NN that is used to classify a particular pattern type is known as the perceptron. The patterns classified in this form of NN are linearly separable; that means the patterns are separable by planes or hyper-planes, like those that are separable by straight lines. The perceptron was a modification of the LTU which now has the ability to update its weights automatically. This Rosenblatt's perceptron is more of importance to psychological science. Nevertheless, the nature of its adaptive behavior has made the perceptron a learning machine that is of particular interest to engineers. Figure15 shows an example of a basic, single-layer perceptron with one output.

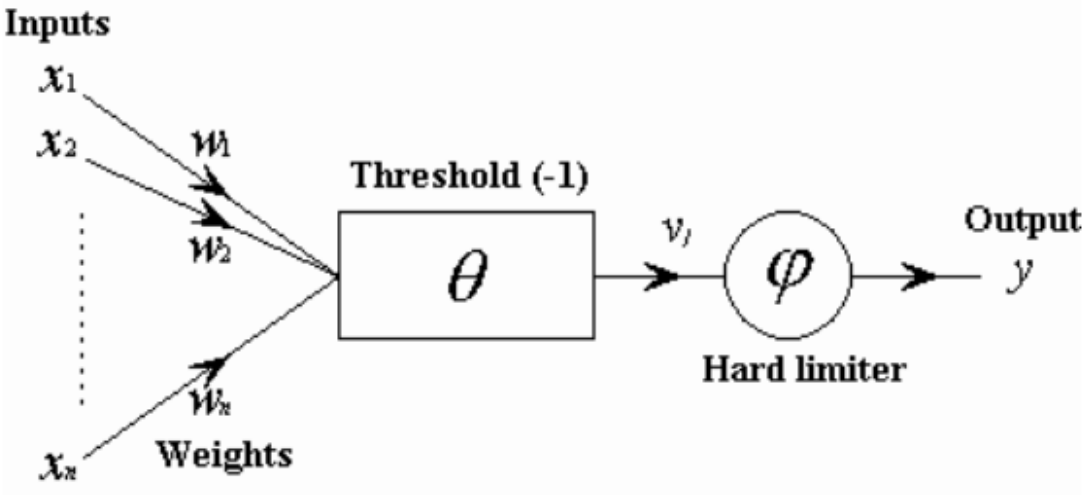

Figure 15: Single-layer Perceptron

From the model of Figure 15, the linear summation of the inputs at the output of the threshold notation (that is also the input to the hard limiter) can be defined in the case of an elementary perceptron as

$$v = \sum_{i=1}^{n} w_i x_i - \theta = 0$$

The synaptic weights in this perceptron can either be fixed or adapted on an iteration by iteration basis. For the adaptation, the model adopts a learning algorithm which is known as the perceptron convergence algorithm (PCA). PCA is a weight-updating rule and can be mathematically expressed as

$$w_{n+1} = w_n + \eta[d_n - y_n] * x_n$$

Where η is the learning-rate parameter which is a positive constant less than unity, d_n and y_n are respectively the desired and the actual response and x_n is the input vector.

The development of PCA is illustrated in the modified version of the LTU. Referring to Figure 16, the modified LTU can be observed to be using a neuron equation that is similar to the M-P Neuron. The difference is that its threshold, θ is now treated as a synaptic weight associated permanently with a fixed input value of -1that behaves like a threshold. The mathematical expression for the modified LTU model is then defined as

$$y = f[\sum_{i=0}^{n} w_i x_i]$$

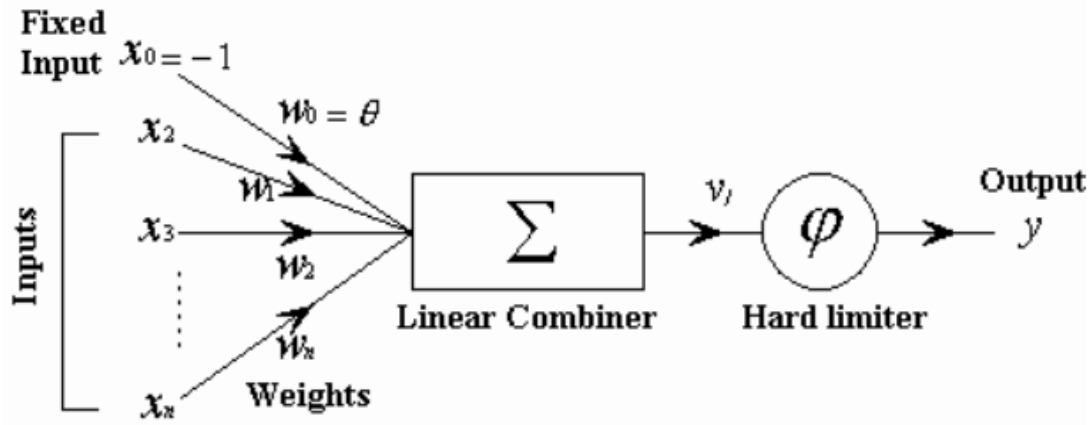

Figure 16: Modified Linear Threshold Unit

The PCA has three classifications which are supervised, reinforcement or unsupervised learning.

In supervised learning, the learning rule is provided with a set of examples (the training set) of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Where p_q is an input to the network and t_q is the corresponding correct (target) output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets. The perceptron learning rule falls in this supervised learning category.

Reinforcement learning is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade (or score) is a measure of the network performance over some

sequence of inputs. This type of learning is currently much less common than supervised learning. It appears to be most suited to control system applications.

In unsupervised learning, the weights and biases are modified in response to network inputs only. There are no target outputs available. They learn to categorize the input patterns into a finite number of classes. This is especially useful in such applications as vector quantization.

5.6 Supervised Learning Algorithms

In this project, supervised learning algorithms will be used to train the neural networks. The term supervised learning which is also known as active learning, can be described with the concept of the availability of an external teacher. The teacher is assumed to have the knowledge of an environment that is represented by a set of input-output examples. Suppose now that the teacher and the Neural Network are exposed to training by feeding it with a particular set of training examples drawn from the environment. The teacher can provide a target response (output) to the Neural Network by depending on its own knowledge. When comparing the target response with the actual response of the Neural Network, we found out that there is an error measure (difference) between the two responses.

As a response to the error, the Neural Network will adjust its network parameters based on the error measure. This adjustment is carried out iteratively in a step-by-step fashion until an optimum response is achieved to minimize the error. When the desired response is achieved, the learning system will “freeze” where it shows the learning process is completed. For better understanding, the concept is described with a block diagram (refer to figure 17). There are few supervised learning algorithm can be applied for short-term load forecasting such as the Widrow-Hoff algorithm and the back-

propagation algorithm. However for this project, back-propagation algorithm will be use to propose the short-term load forecast model.

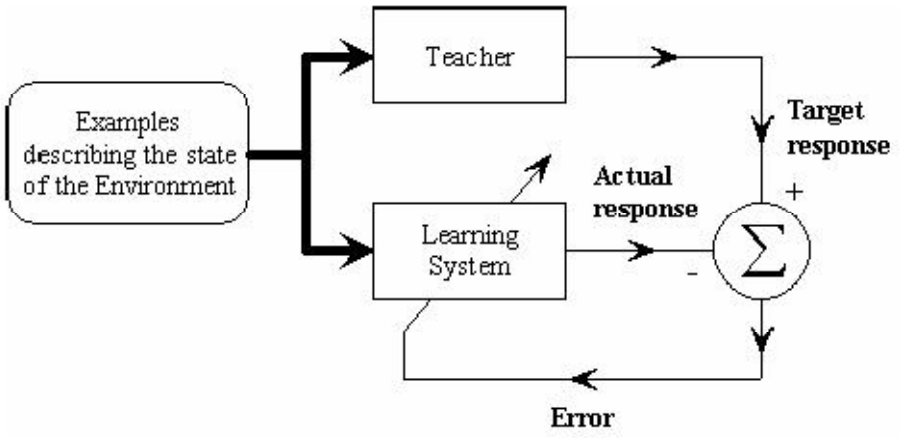

Figure 17: Block Diagram of Supervised Learning

5.6.1 Widrow-Hoff Algorithm

PCA works well with a single LTU for solving pattern classification problems that are linearly separable. However, when it comes to solving classification problems that not linearly separable, the algorithm is not able to terminate. In other words, as it approaches the desire response, it goes into a saturated form of oscillation around the response rather than terminating itself. In computing, algorithms with such phenomenon are not desired. To have the problem resolved, the Widrow-Hoff (W-H) algorithm was introduced.

The W-H algorithm, also known as the least mean square algorithm or the delta rule, was formulated by Widrow and Hoff in 1960. Similar to the PCA, the W-H algorithm was also designed to operate with a single neuron. Although it was unable to solve classification problems that are not linearly separable, it was able to eliminate the

problem faced by the PCA whilst achieving a reasonable accuracy with the desired response. The W-H approach is illustrated in Figure 18.

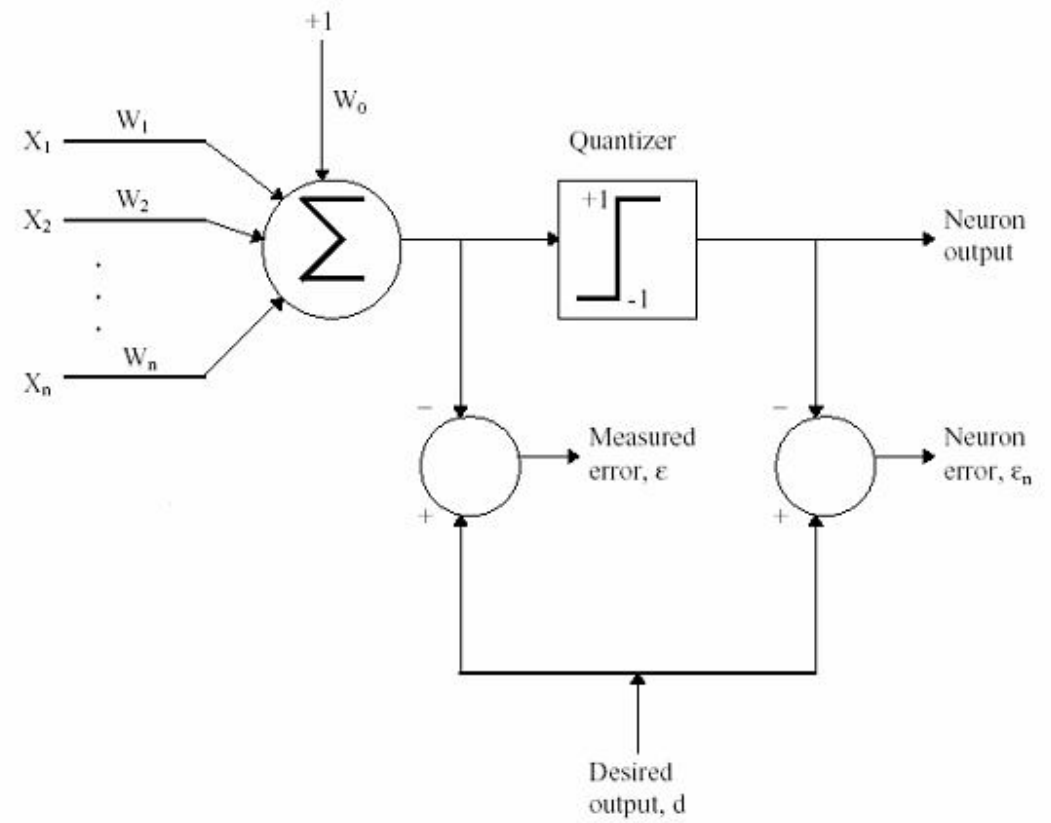

Figure 18: The Widrow-Hoff Approach

5.6.2 Back-propagation Algorithm

The most popular artificial neural network architecture for electric load forecasting is back propagation. Back propagation neural networks use continuously valued functions and supervised learning. That is, under supervised learning, the actual numerical weights assigned to element inputs are determined by matching historical data (such as time and temperature) to desired outputs (such as historical electric loads) in a

pre-operational “training session. For this project, the input variables include historical hourly load data and temperature. The model can forecast load profiles from one to seven days. To handle these complex computations, the multi-layer perceptron (MLP) networks need to be introduced. Figure 19 illustrates an example of a MLP with two hidden layers.

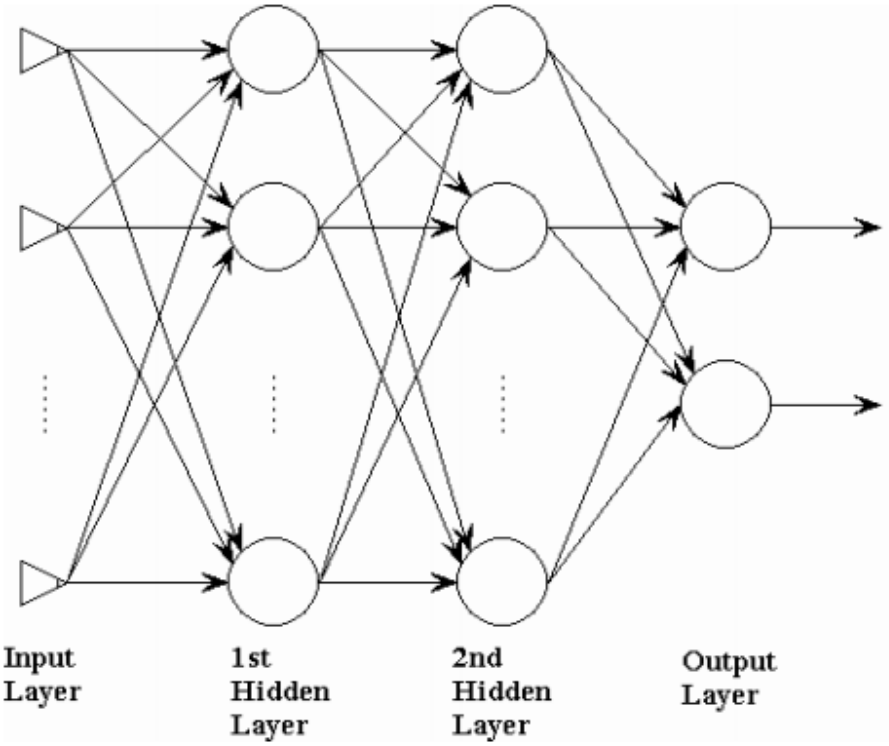

Figure 19: Multi-layer Perceptron with Two Hidden Layers

MLPs are feed-forward Neural Networks commonly trained in a supervised fashion with the error back-propagation (BP) algorithm [19] [20]. The back propagation consists of a two direction which are the forward pass and backward pass. The operation take place by first passing an input signal forward through the network in the direction towards the output until a set of actual response is achieved by the network. Then the error signal is generated from the difference between the actual and target response. Finally the generated error signal is passed backwards through the hidden layers, in the

direction towards the inputs. During the forward pass, the synaptic weights of the network are fixed. It is during the backward pass that the synaptic weights are adjusted to adapt the network in producing desired outputs. Figure 20 illustrates the flow of the forward and the backward signals.

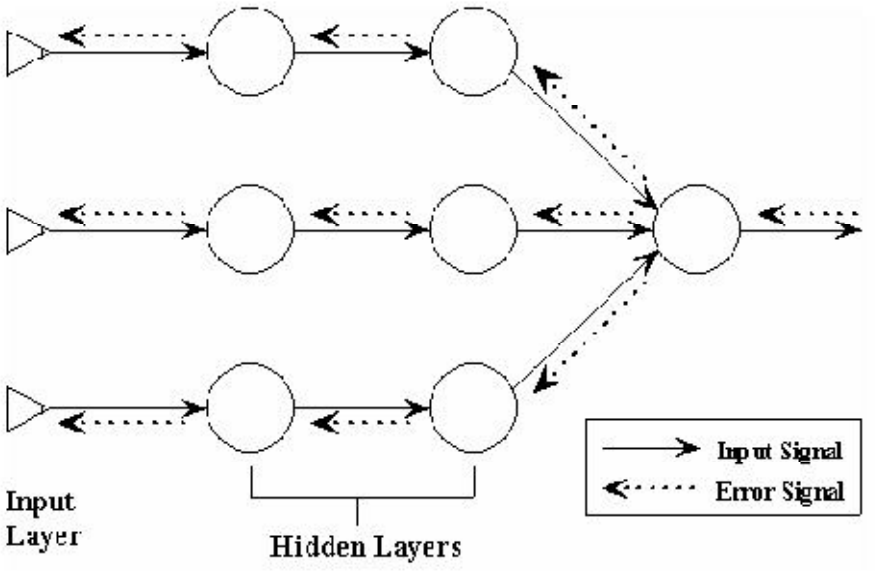

Figure 20: Flow of Signals with the Back-propagation Algorithm

The adaptation of a BP MLP network with an arbitrary number of layers can be mathematically expressed as

$$\Delta w = \eta \sum_{p} \delta_{layer} \times V_{input}$$

where η is the learning rate, V_{input} is the input vector appearing at the input terminal of the weight, and δ_{layer} is a term associated with the error function which is layer dependent.

The basic BP algorithm is a gradient descent algorithm which adjusts the network weights along the steepest descent direction of the error function (that is, the direction in which the error function decreases most rapidly; negative of the gradient). Since the BP algorithm is widely employed with supervised MLP networks, it is not surprising that many variants of it are known to exist. These variants are commonly known as the Conjugate Gradient Algorithms (CGA). Table 3 is a summary of various popular CGAs obtained from the MATLAB® help file.

Types of CGA	Comments on the algorithms
Fletcher-Reeves Update (FRU)	• Line search is required. • Small storage requirement. • Good choice for networks with large number of weights
Polak-Ribiére Update (PRU)	• Line search is required. • Slightly larger storage requirement than Fletcher-Reeves Update. • Faster convergence for certain problems.
Powell-Beale Restarts (PBR)	• Line search is required. • Performance is better than Polak-Ribiére Update for certain problems. • Slightly larger storage requirement and generally faster convergence than Polak-Ribiére Update.
Scaled Conjugate Gradient (SCG)	• Line search is not required. • Requires more iterations to

	converge but shorter time needed for computation in each iteration. • Good general purpose training algorithm.

Table 3: Summary of Various Conjugate Gradient Algorithms

5.7 Classification of Neural Networks

NNs can be classified according to the type of input data or the type of algorithm that is implemented. The various NN classifications are described as in Figure 21.

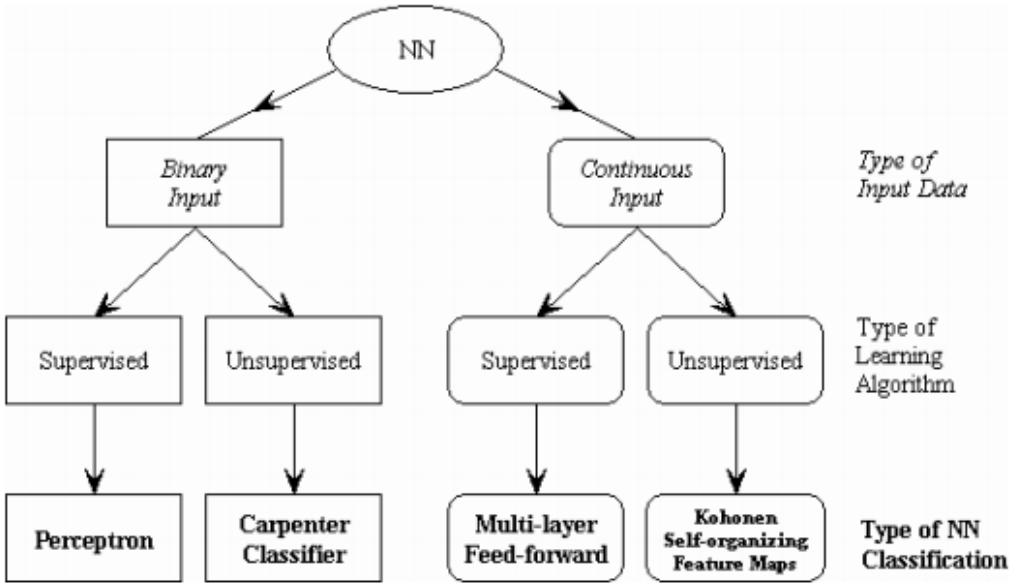

Figure 21: Classifications of Neural Network

NNs that take in binary inputs are usually associated with Boolean logic functions. That is, solving logical functions with inputs taking values of 1 or 0. Since the perceptron is capable of solving Boolean functions, it is typically employed for such NNs.

NNs that take in continuous type of inputs (e.g. time series data) can be presented with any range of values for both its input and output vectors. This type of NN is commonly used for tasks such as pattern classification and function approximation. If a supervised learning algorithm is adopted, a typical implementation would be the multi-layer feed- forward NN [24]. The multi-layer feed-forward NN is also a popular architecture that has been proven to give satisfactory results for time-series predictions [10][14].

CHAPTER 6

THE PROPOSED FORECAST MODEL

6.1 Preliminary Forecast Model

The preliminary forecast model has three main stages. This is illustrated in Figure 22.

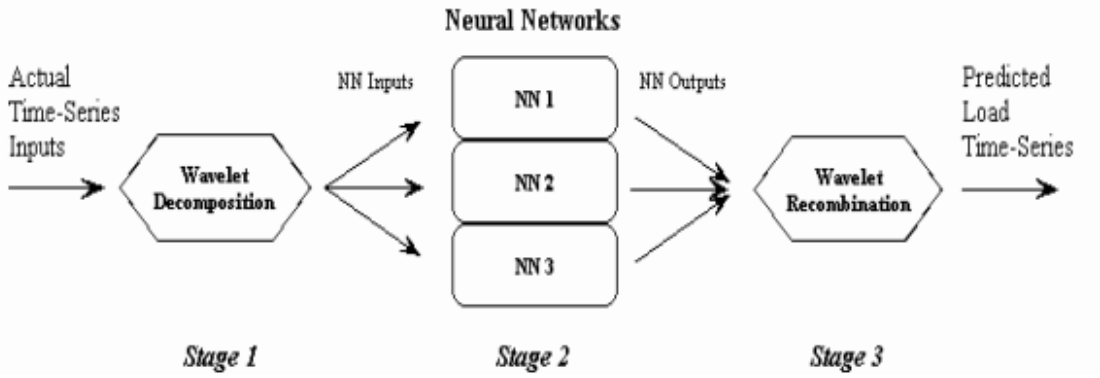

Figure 22: Three Stages of the Preliminary Forecast Model

As depicted in Figure 22, wavelet techniques are implemented in the first and the last stages. The actual time-series (electricity load and temperature data) are first decomposed into a number of wavelet coefficient signals and one approximation signal. The decomposed signals are then fed into the NNs at the second stage to predict the future time-series patterns for each of the signals. Finally, the predicted signals are recombined in the last stage to form the final predicted load time series.

6.1.1 Stage 1: Pre Signal Processing

For pre signal processing, historical data is fed to the model as time-series signals. Non-decimated Wavelet Transform (NWT) is used as the pre signal processor in the model and depending on the selected resolution level, the respective time-series signals are decomposed into a number of wavelet coefficients. In other words, if the resolution level is defined as n, after decomposing the signal, there will be one approximation coefficient series as well as n number of detail coefficient series. These decomposed coefficients are then normalized and fed as inputs to the signal predictor (NNs) for either training or forecasting use. The decomposition process is illustrated in Figure 23.

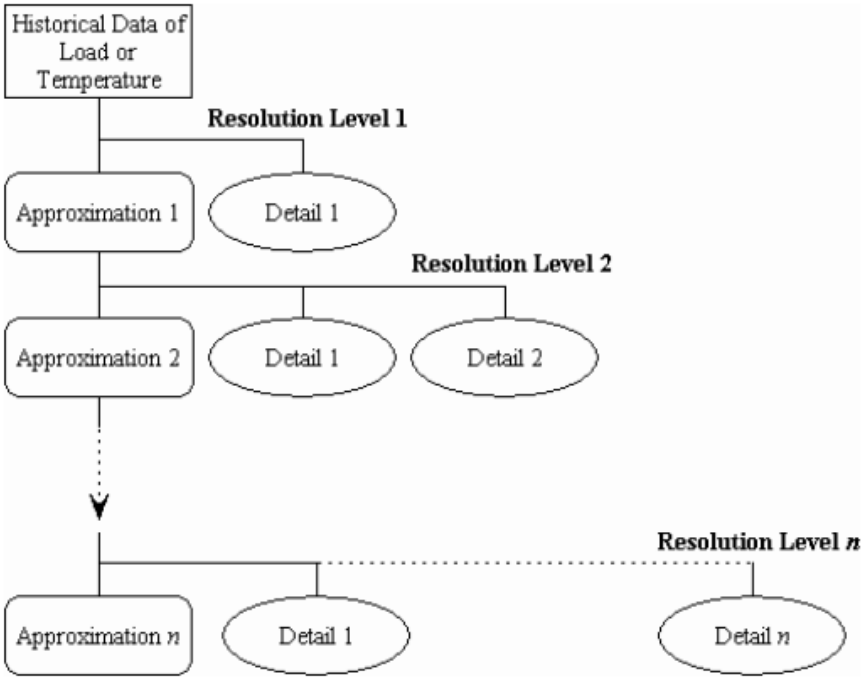

Figure 23: Wavelet Decomposition Process

The most suitable resolution level is identified based on the smoothness of the approximation signal at that level (i.e. having all the high frequency components removed). The desired approximation signal should depict a general pattern of its original. A higher resolution level would produce a smoother approximation signal. The details on determining this resolution level will be discussed in the simulation section.

6.1.2 Stage 2: Signal Prediction

NNs are used for signal prediction in the forecast model. The number of NNs needed for the model is determined by the number of wavelet coefficient signals at the output of the pre-processor. For each wavelet coefficient signal (including the approximation), one NN is required to perform the corresponding prediction.

6.1.3 Stage 3: Post Signal Processing

For post signal processing, the same wavelet technique (NWT) and resolution level as mentioned in section 6.1.1 are used. In this stage, the outputs from the signal predictor (NNs) are combined to form the final predicted output. This is achieved by summing all the predicted wavelet coefficients. Figure 24 illustrates the recombination process.

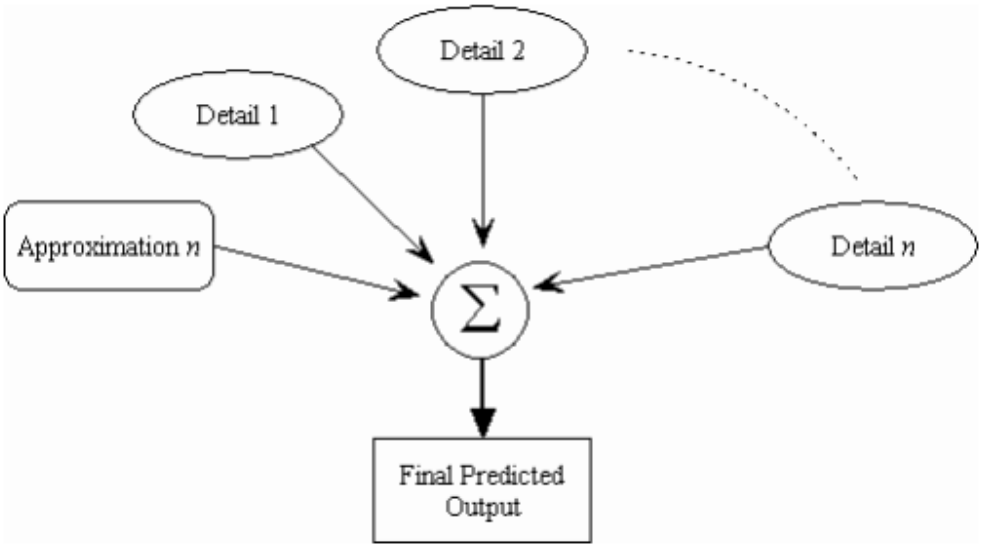

Figure 24: Wavelet Recombination Process

6.2 Finalizing the Design

Since BP algorithm with supervised MLP feed-forward networks is a popular choice for time-series predictions, the proposed forecast model will be implemented as accordingly. Other design considerations are discussed in the subsequent subsections.

6.2.1 Number of Input Nodes and Output Neurons

The elements in the input layer are called nodes instead of neurons because they are only used to distribute input variables to the hidden layer. As mentioned earlier, the training time-series data consist of temperature and electricity load of Queensland, hence the initial plan is to have two input streams, one for the average load data and the other is for the temperature data. Unfortunately, there is no average temperature data available for use; thus the solution is to take seven sets of temperature data from five major cities (higher power consumption regions) of Queensland and hence, there will be five input streams for temperature instead of one. Therefore, the total number of input nodes for the proposed model is fixed at six.

The forecast model is set to have one output neuron that takes one value of the time-series signal as the target.

6.2.2 Number of Hidden Neurons

The learning capability of a NN depends on the number of neurons in its hidden layer. Determining the number of hidden neurons is important; if the number of hidden neurons is insufficient, the consequence will result in an inaccurate approximation. On the other, excessive hidden neurons are not desirable. This will affect the learning progress of the network. In other words, instead of learning the problem, the network will tend to memorize it. Over-sizing the hidden layer can also lead to longer training cycles. With adequate time for testing, the most suitable number of neurons for the NN is explored.

6.2.3 Learning Algorithm

As mentioned earlier in section 6.2, the BP algorithm will be used as the learning algorithm for the forecast model. Since the standard BP algorithm is a gradient descent algorithm, variants of the algorithm (refer to Table 3) are studied. For the purpose of this thesis, the SCG algorithm is found to be suitable. This is due to its advantage of fast computational time for a large NN size. Using this algorithm will significantly shorten the time needed to train the NNs. Further information regarding SCG algorithm and its benchmark with other algorithms is obtainable from the MATLAB® help file.

6.3 Final Proposed Forecast Model

After finalizing the design considerations, the final proposed NN forecast model is as depicted in Figure 25 on the next page. The model is a two-layer feed-forward NN with two layers of adjustable weights.

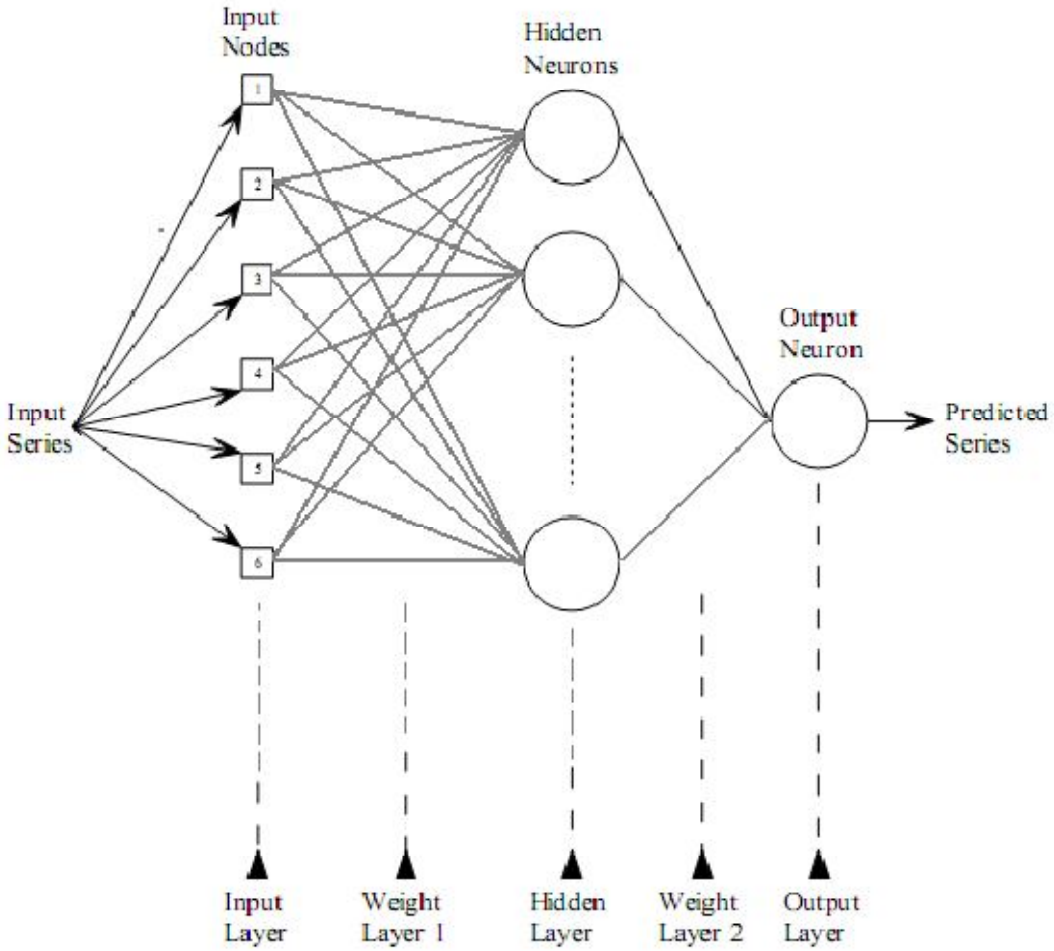

Figure 25: Final Proposed NN Forecast Model

CHAPTER 7

SIMULATION AND RESULTS

7.1 Methodology

First and foremost, three set of electricity load and temperature data for months of January, February and March 2009, on a half-hourly basis are downloaded from the AEMO website. The format of data downloaded is in Microsoft Office Excel Comma Separated Values File. For the electricity load data, it consists of the settlement date, time and the total demand whereas for the temperature data, it consists of settlement date, time and the temperature value.

For the proposed model, it is tested with six sets of historical data containing the electricity load and temperature data for the month of January 2009, on a half-hourly basis; one set of electricity load data of Queensland and five sets of temperature data from five different locations (higher power consumption regions) of Queensland. The set of electricity load data is downloaded from the AEMO website. The reason for using five sets of temperature data is a compromise due to the unavailability of a general set of average temperature data for Queensland. The five different locations where the

temperature data has been collected from are namely Brisbane, Gold Coast, Townsville, Cairns and Rockhampton.

The simulation results are obtained through the use of four different programs main.m; nnCreate.m; nnForecast.m and nnRetrain.m (refer to the Appendix A for listing of codes). These programs were written with the MATLAB® software, using the standard commands found in its toolbox. A few other additional commands were downloaded from the NETLAB Neural Network Software Website.

Although temperature data is included with the electricity load data in this forecast model, however the model is mainly meant to forecast only the electricity load. Therefore, the final predicted series is formed by setting the training target of the NNs to be a later load series. Temperature is only used with load to form a particular time-series pattern as the inputs for training the NNs or for load predictions.

The creating of the purposed forecasting model is beginning with the selection of the desired resolution level, n. Users need to choose the desired amount of data to use and data from the month that to be used. Then the program will read the historical demand data into tDemandArray and display number of points in the selected time series demand data. Before the wavelet decomposition technique (NWT) is applied, the five sets of temperature data are first normalized and then multiplied by a power consumption factor. The reason for having to use a power consumption factor is due to regional effects. During the wavelet decomposition, the Non-decimated Wavelet Transform (NWT) is used as the pre signal processor in the model and depending on the selected resolution level, the respective time-series signals are decomposed into a number of wavelet coefficients according to the resolution level which defined as n. After decomposing the signal, there will be one approximation coefficient series as well

as n number of detail coefficient series. The function swt (Discrete stationary wavelet transform 1-D) is used for the wavelet decomposition.

SWT Discrete stationary wavelet transform 1-D performs a multilevel 1-D stationary wavelet decomposition using either a specific orthogonal wavelet ('wname' see WFILTERS for more information) or specific orthogonal wavelet decomposition filters. SWC = SWT(X,N,'wname') computes the stationary wavelet decomposition of the signal X at level N, using 'wname'. N must be a strictly positive integer (see WMAXLEV for more information). 2^N must divide length(X).

Moreover, the original demand data signal and the approximation demand data signal are plotted. The same wavelet decomposition process is undergoing by the temperature data to obtain the original demand data signal and the approximation demand data signal graph.

In the nnCreate.m file, it involves the data preprocessing where the mature Neural Network is created. The decomposed coefficients obtained previously are then normalized and fed as inputs to the signal predictor (NNs) for either training or forecasting use. To capture the pattern of the signal, the model is set to accept dataPoints multiply six sets of training examples where included one set of demand and five sets of temperature (one set for each five different locations).The normalised demand data from point 2 is to be taken as the output value for the first iteration of training examples.

Next, the program will create and train the Neural Network for the respective demand and temperature coefficient signals. The number of input nodes and hidden neurons for the respective demand and temperature coefficient signal is stetted. In additional, the setting number of training examples and target outputs of the training examples are done. For the preparing of the training examples, firstly, training of the

inputs to be six is set with user defined number of iterations (dataPoints). Then to create the new neural network for respective coefficient signal, the function NET = MLP(NIN, NHIDDEN, NOUT, FUNC) is used.

NET = MLP(NIN, NHIDDEN, NOUT, FUNC) takes the number of inputs, hidden units and output units for a 2-layer feed-forward network, together with a string FUNC which specifies the output unit activation function, and returns a data structure NET. The weights are drawn from a zero mean, unit variance isotropic Gaussian, with variance scaled by the fan-in of the hidden or output units as appropriate. This makes use of the Matlab function RANDN and so the seed for the random weight initialization can be set using RANDN('STATE', S) where S is the seed value. The hidden units use the TANH activation function.

The fields in NET are

type = 'mlp'

nin = number of inputs

nhidden = number of hidden units

nout = number of outputs

nwts = total number of weights and biases

actfn = string describing the output unit activation function:

'linear'

'logistic

'softmax'

w1 = first-layer weight matrix

b1 = first-layer bias vector

w2 = second-layer weight matrix

b2 = second-layer bias vector

Here W1 has dimensions NIN times NHIDDEN, B1 has dimensions 1 times NHIDDEN, W2 has dimensions NHIDDEN times NOUT, and B2 has dimensions 1 times NOUT.

For the training of the Neural Network, the function NETOPT(NET, OPTIONS, X, T, ALG, LEN). NETOPT Optimize the weights in a network model. NETOPT is a helper function which facilitates the training of networks using the general purpose optimizers as well as sampling from the posterior distribution of parameters using general purpose Markov chain Monte Carlo sampling algorithms. It can be used with any function that searches in parameter space using error and gradient functions.

[NET, OPTIONS] = NETOPT(NET, OPTIONS, X, T, ALG) takes a network data structure NET, together with a vector OPTIONS of parameters governing the behavior of the optimization algorithm, a matrix X of input vectors and a matrix T of target vectors, and returns the trained network as well as an updated OPTIONS vector. The string ALG determines which optimization algorithm (CONJGRAD, QUASINEW, SCG etc.) or Monte Carlo algorithm (such as HMC) will be used.[NET, OPTIONS, VARARGOUT] = NETOPT(NET, OPTIONS, X, T, ALG) also returns any additional return values from the optimization algorithm.

After the training of the Neural Network, the file is saved. Then the program will load the trained Neural Network for load forecasting and recombines the predicted outputs from the Neural Network to form the final predicted load series. The function Y = MLPFWD(NET, X), forward propagation through 2-layer network is used for the load forecasting.

The Y = MLPFWD(NET, X) takes a network data structure NET together with a matrix X of input vectors, and forward propagates the inputs through the network to generate a matrix Y of output vectors. Each row of X corresponds to one input vector and each row of Y corresponds to one output vector. [Y, Z] = MLPFWD(NET, X) also

generates a matrix Z of the hidden unit activations where each row corresponds to one pattern.

Finally, the actual time series against predicted result is plotted and the mean absolute error is calculated. The model is evaluated based on it prediction errors. A successful model would yield an accurate time-series forecast. The performance of the model is hence measured using the mean absolute percentage error (MAPE) which is defined as

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{|x_i - y_i|}{x_i}\right) * 100\%$$

where N is the number of points measured, x_i is the actual values and y_i is the predicted values.

7.2 Simulations

7.2.1 Finding the Suitable Resolution Level

As mentioned in Chapter 6.1.1, the best resolution level is tested. The results are as follow:

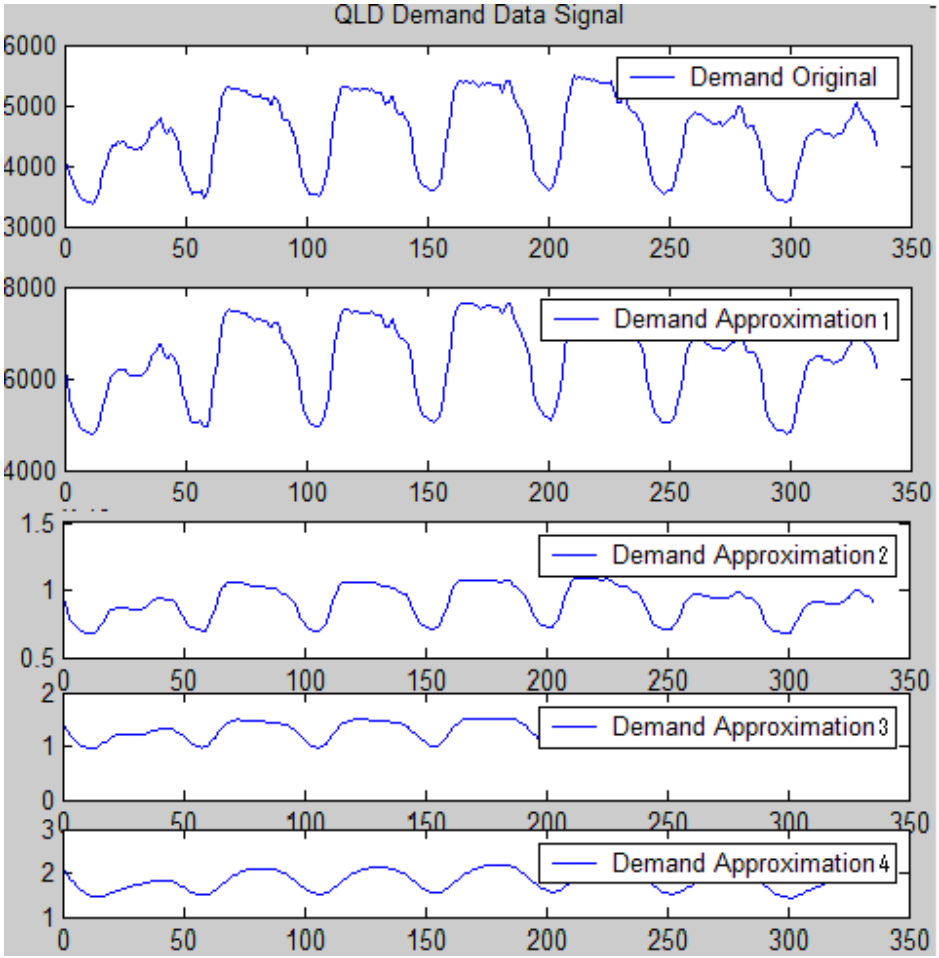

Figure 26: Comparison of Different Resolution Levels

From Figure 26, it can be observed that the approximation signal at resolution level two is sufficiently smooth to represent a general pattern of the original signal. Therefore, resolution level two is chosen for both pre and post processing stages.

7.2.2 Setting the Number of Hidden Neurons

The proposed model is tested with a number of different values of hidden neurons, however no significant changes are observed with the predicted results. This is a normal phenomenon as until now, no one has proven a typical configuration for the hidden neurons (refer to the MATLAB help file). Therefore, a default configuration for setting the number of hidden neurons can be adopted by setting the value lesser or equal to the number of inputs and greater than the number of outputs. For the purpose of this thesis, the results obtained from the simulations are based on setting the number of hidden neurons to three.

7.3 Results

Six sets of historical time-series data are decomposed at resolution level 2 as follows:

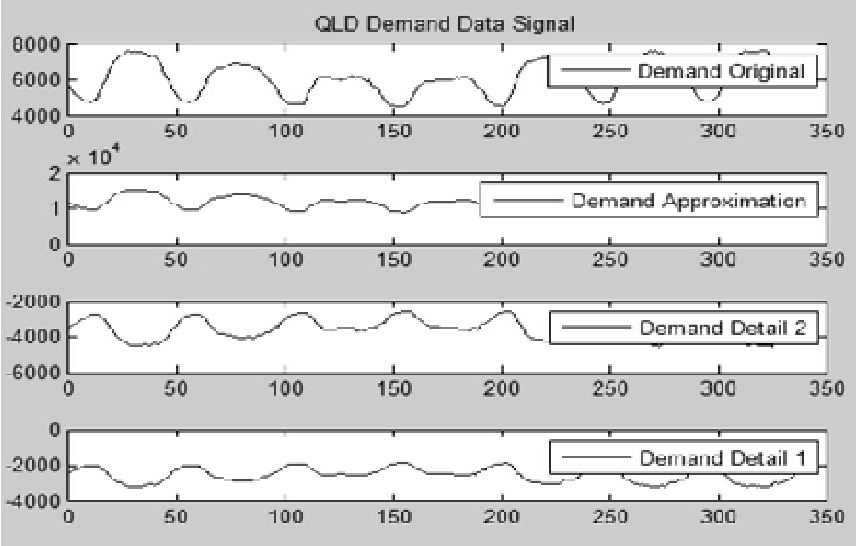

Figure 27: Decomposition of Queensland Demand Series

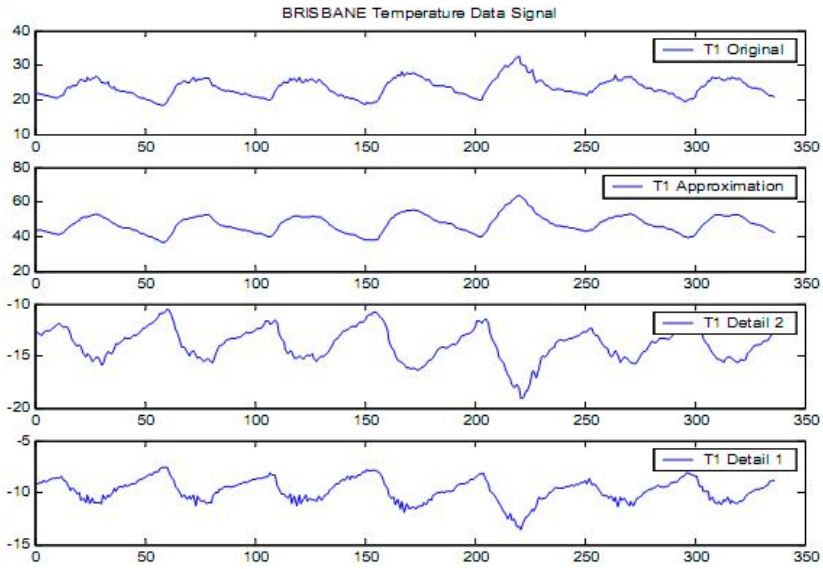

Figure 28: Decomposition of Brisbane Temperature Series

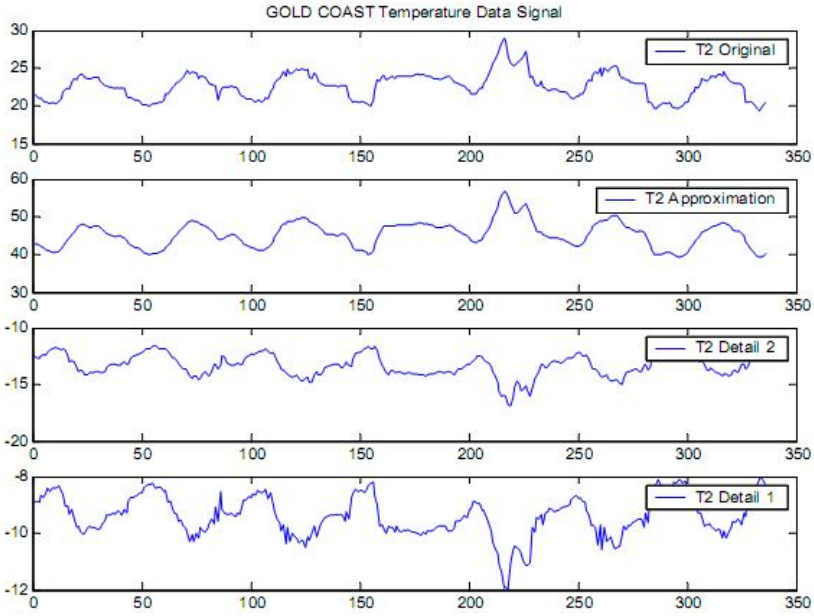

Figure 29: Decomposition of Gold Coast Temperature Series

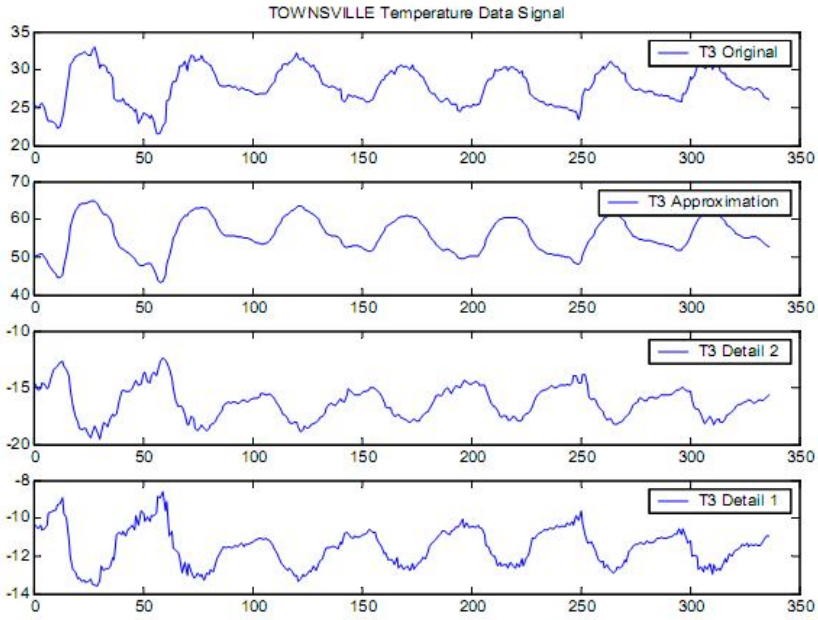

Figure 30: Decomposition of Townsville Temperature Series

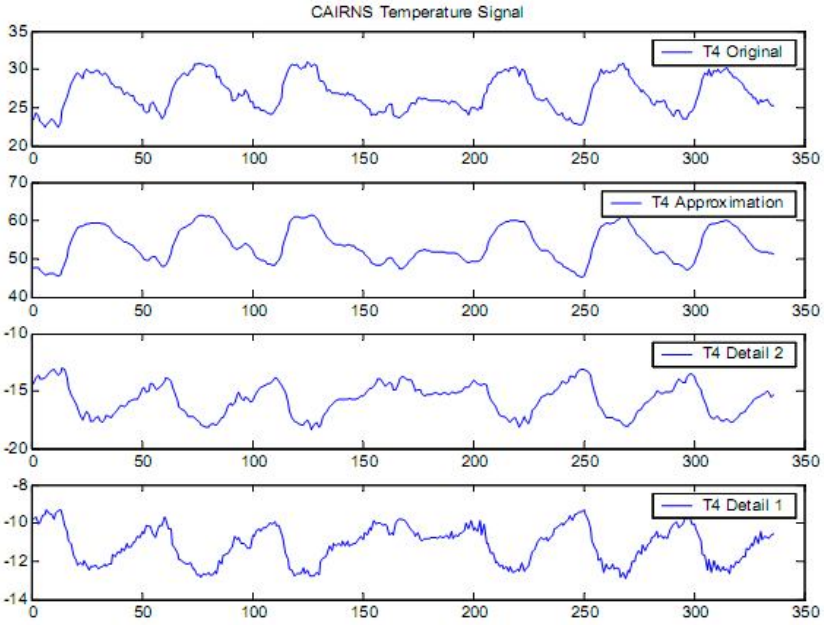

Figure 31: Decomposition of Cairns Temperature Series

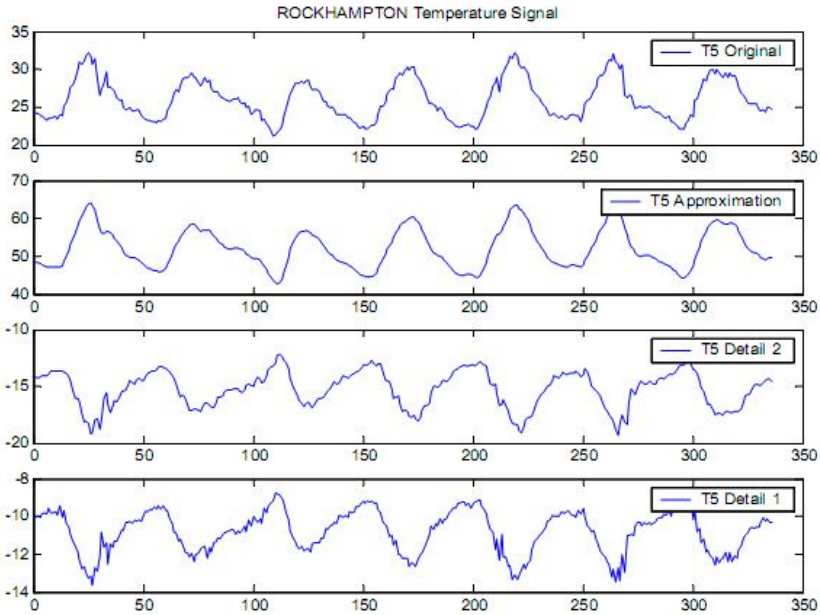

Figure 32: Decomposition of Rockhampton Temperature Series

The performance of the forecast model was evaluated and the results are as follows:

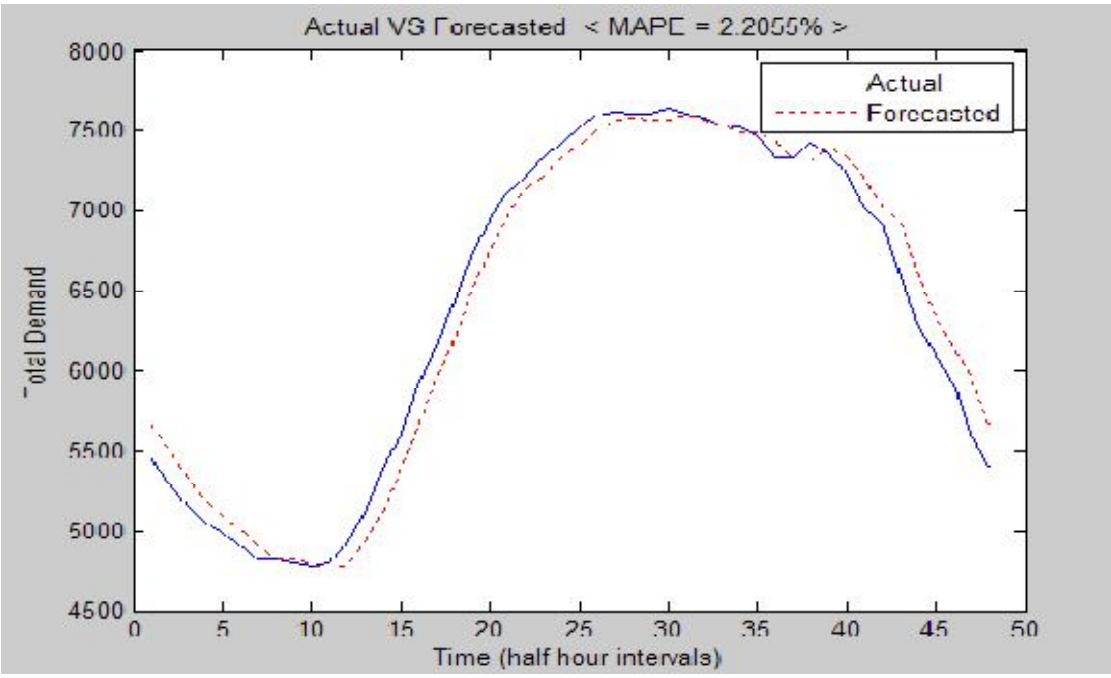

Figure 33: Load Forecasted for 1 day (48 points)

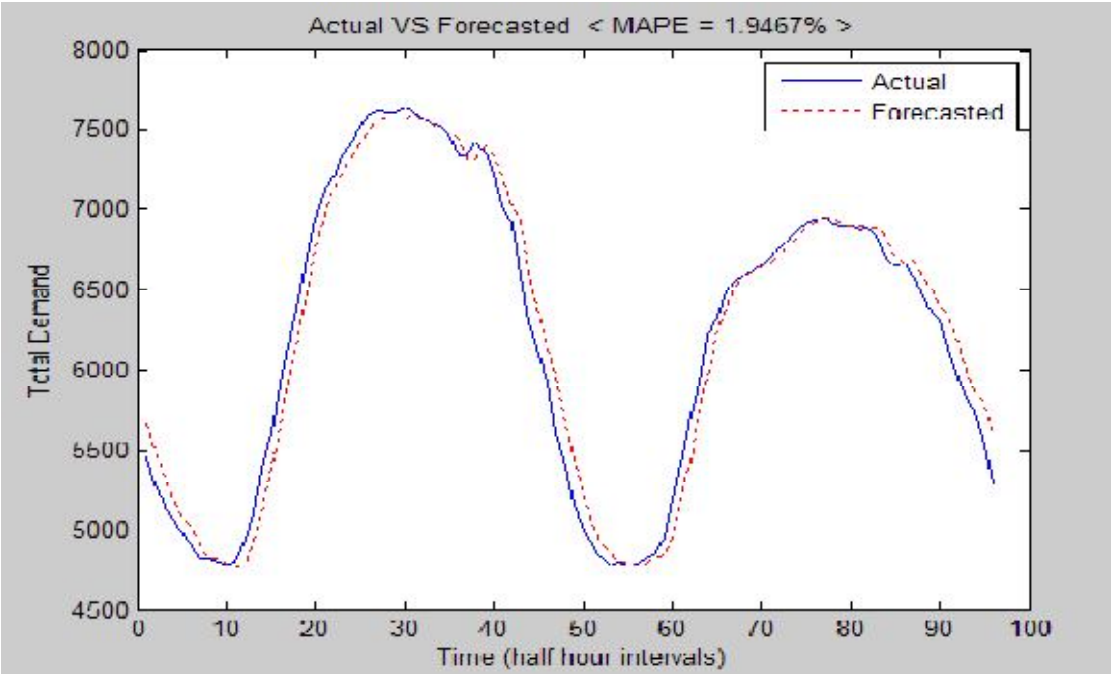

Figure 34: Load Forecasted for 2 days (96 points)

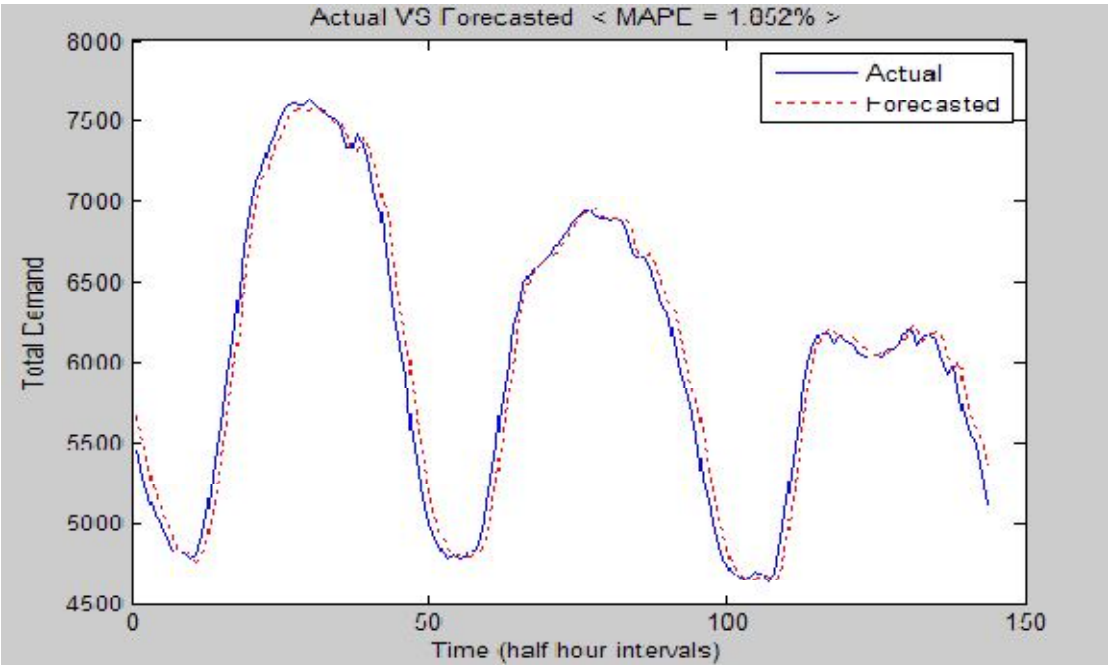

Figure 35: Load Forecasted for 3 days (144 points)

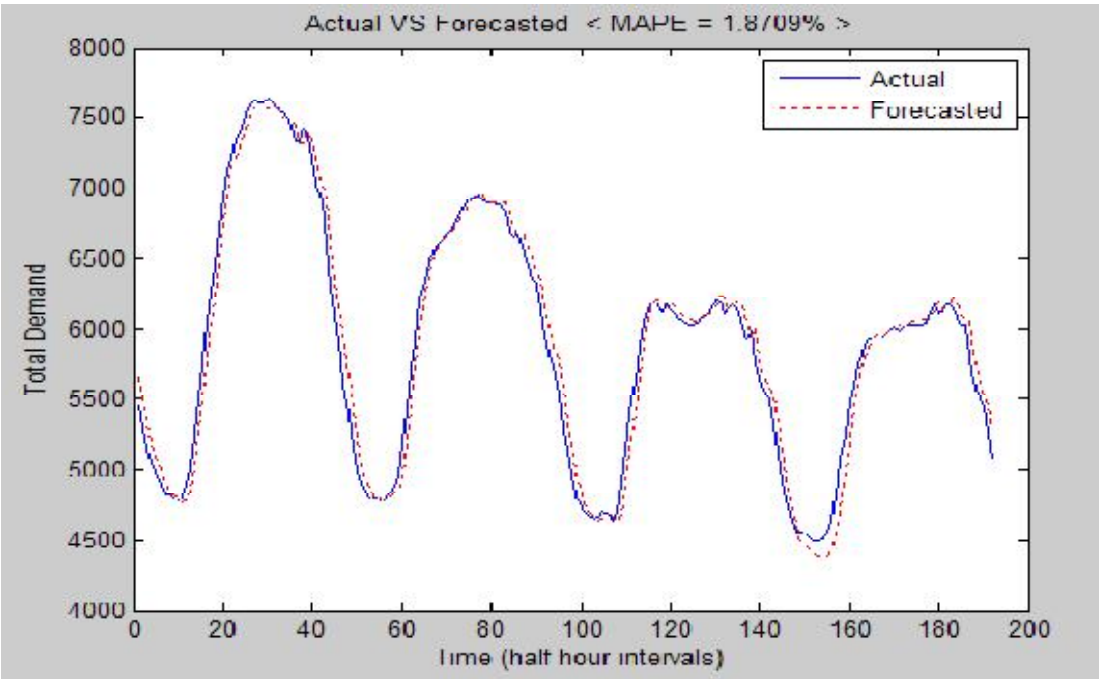

Figure 36: Load Forecasted for 4 days (192 points)

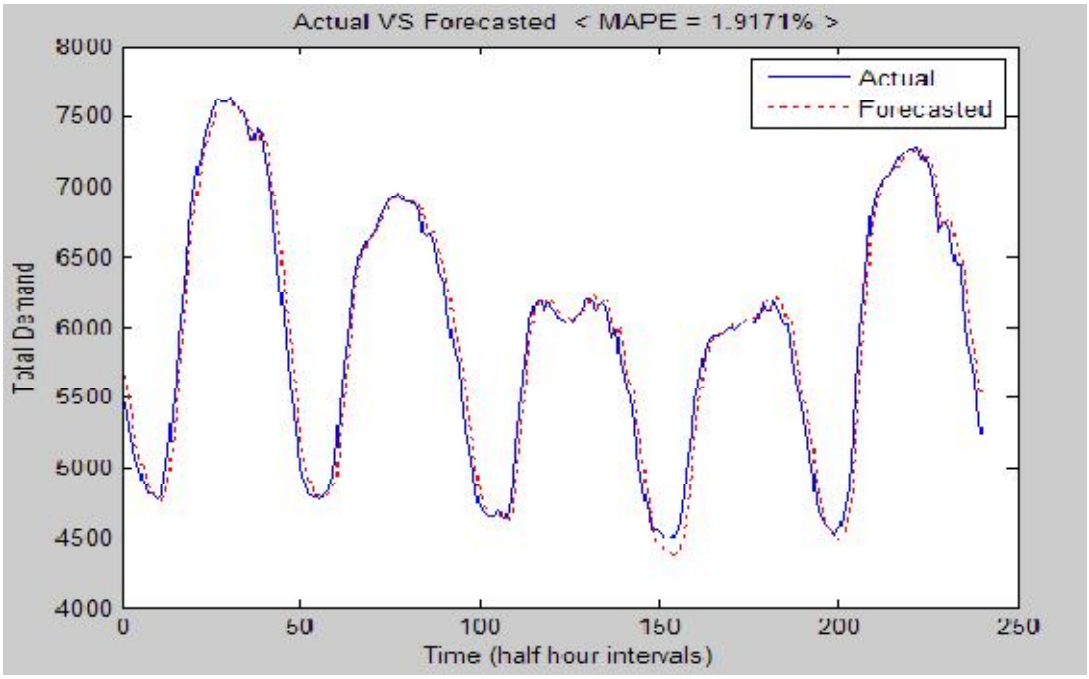

Figure 37: Load Forecasted for 5 days (240 points)

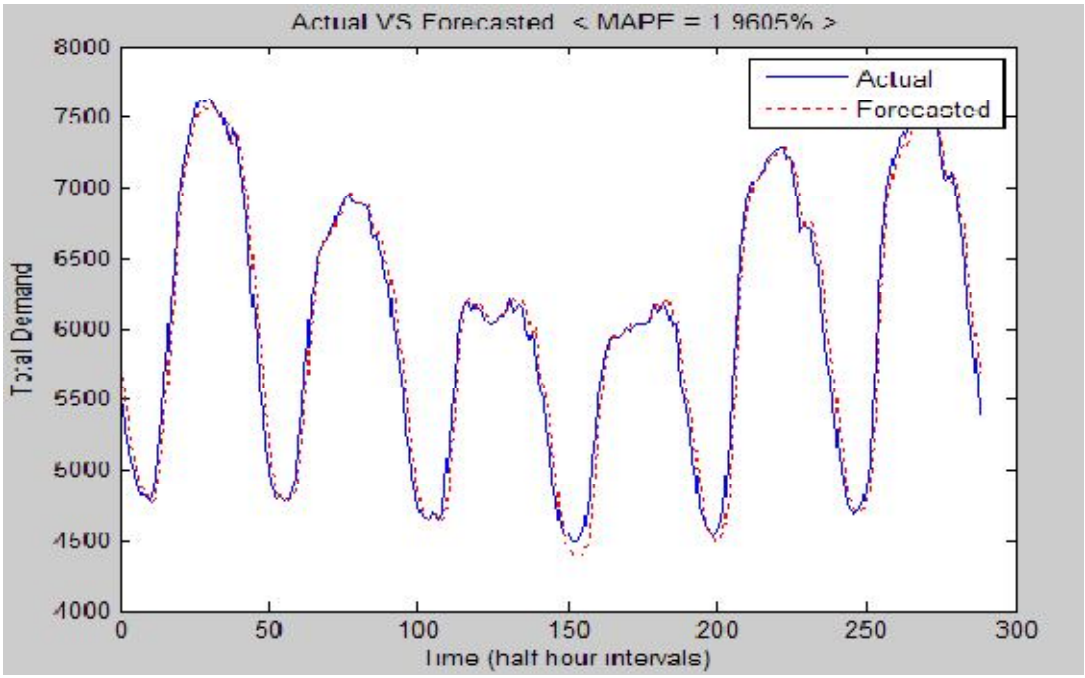

Figure 38: Load Forecasted for 6 days (288 points)

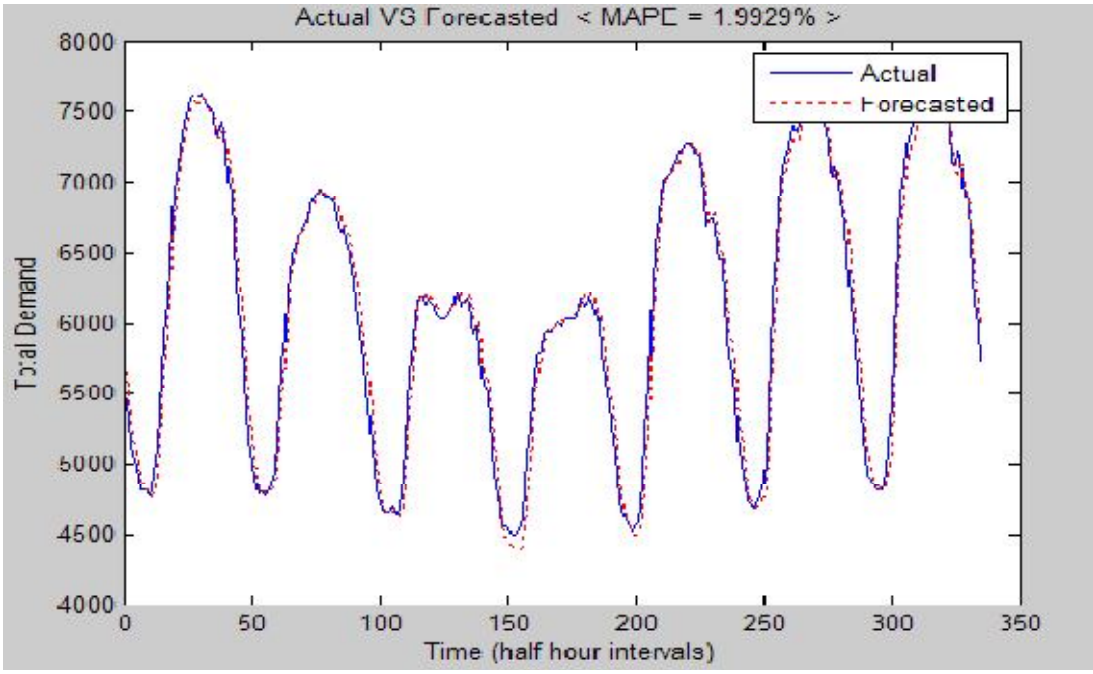

Figure 39: Load Forecasted for 1 week (336 points)

7.4 Matlab GUI

To begin the Short-term Load Forecasting System, first we need to open the Main.m file. At the Debug menu, clicks run Main. Then The Main window as Figure 40 will appear. The Main window contain five buttons which are the Introduction, How to use?, Start, Credit and Exit.

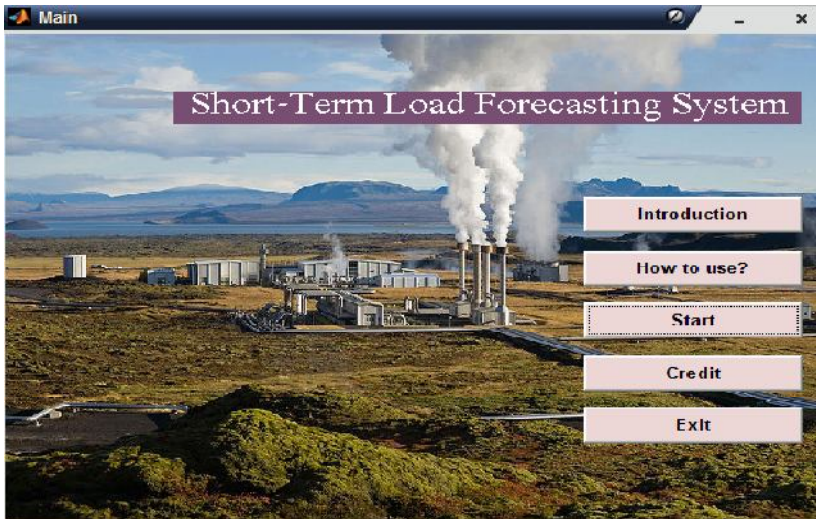

Figure 40: The Main window

If the Introduction button is clicked, Figure 41 will appear.

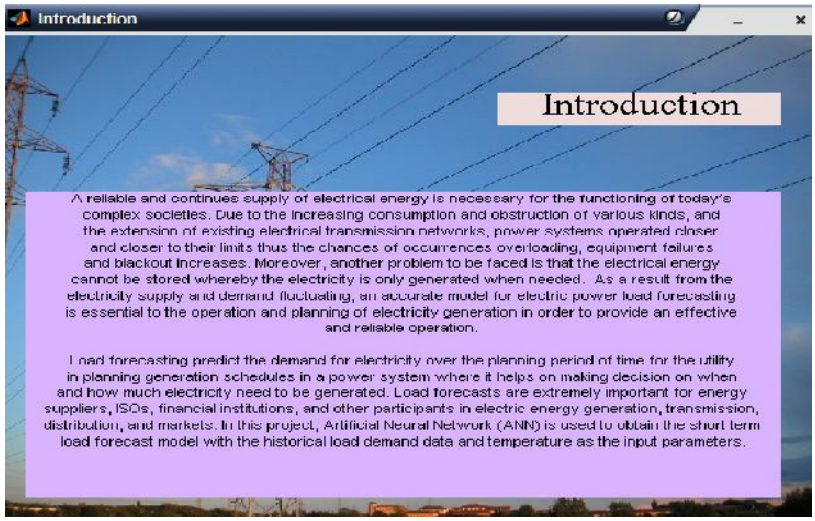

Figure 41: The Introduction

However, if the How to use? button is clicked, Figure 42, 43, 44, 45 will appear one by one every time the next button is clicked. The figures show step by step instruction to the first time user in using the Short-term Load Forecasting system. It gives a brief idea on what data is needed by the system on every stage to perform the load forecasting.

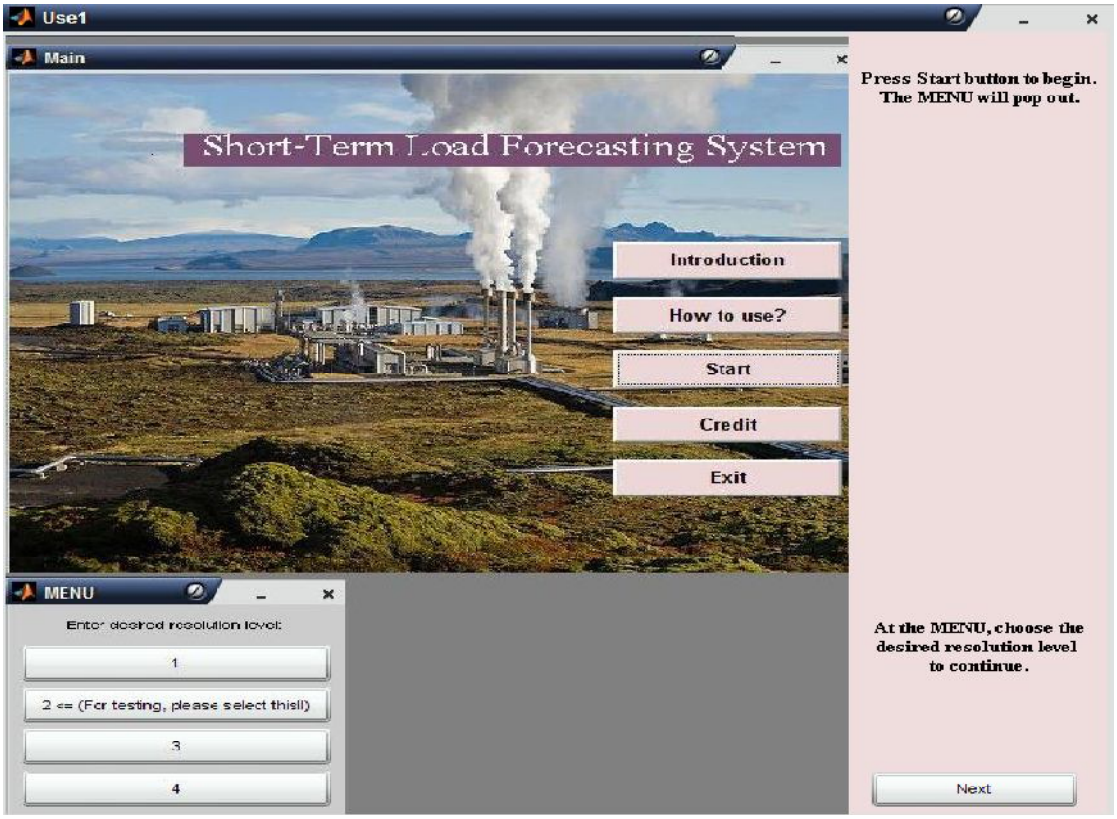

Figure 42: Use1

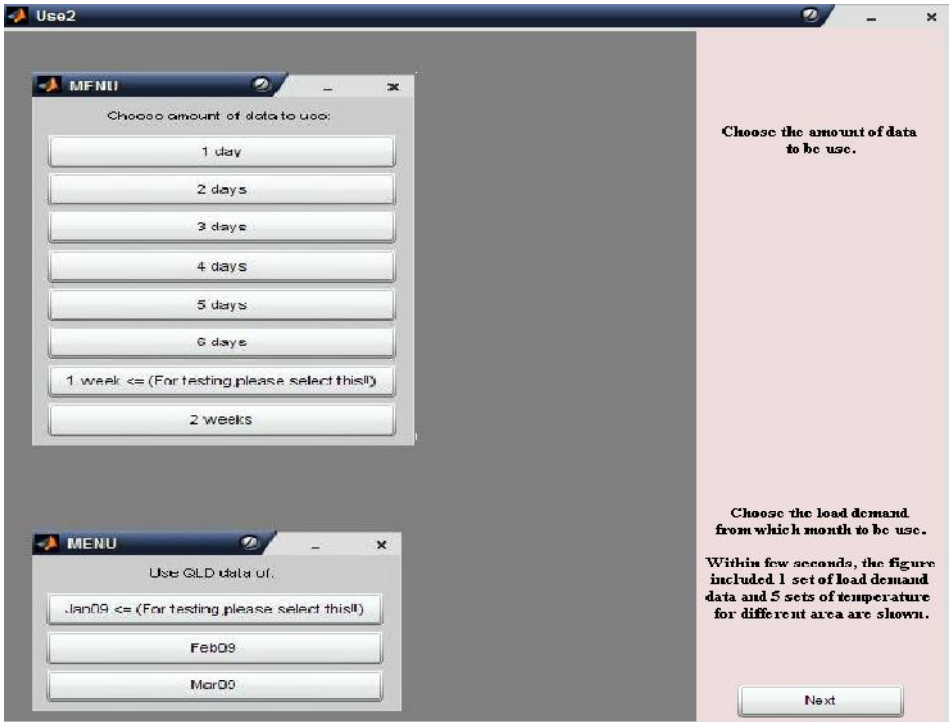

Figure 43: Use2

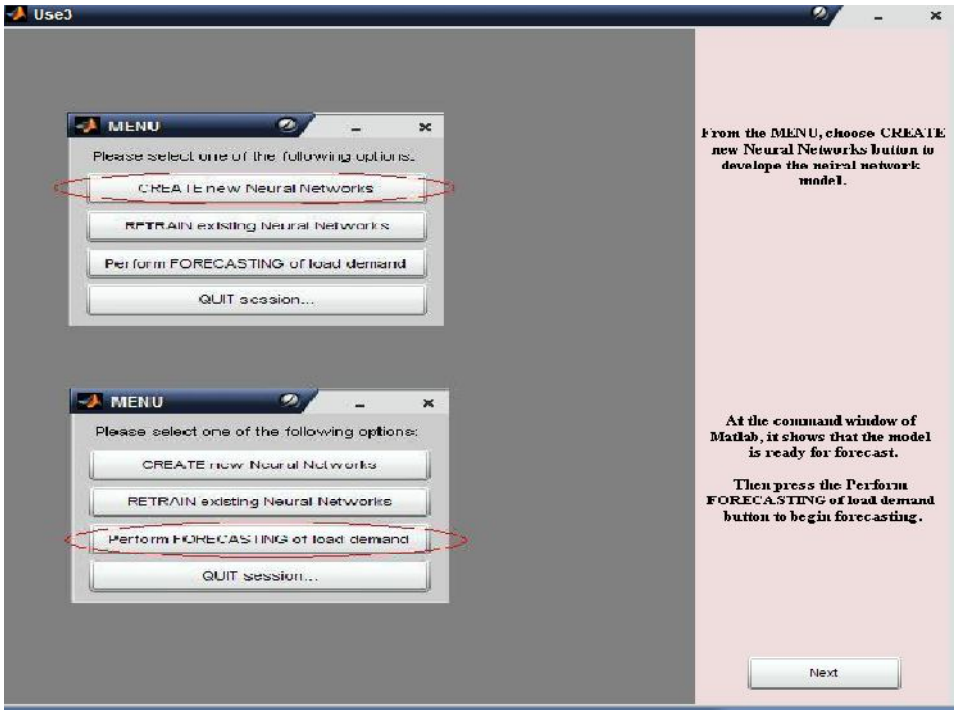

Figure 44: Use3

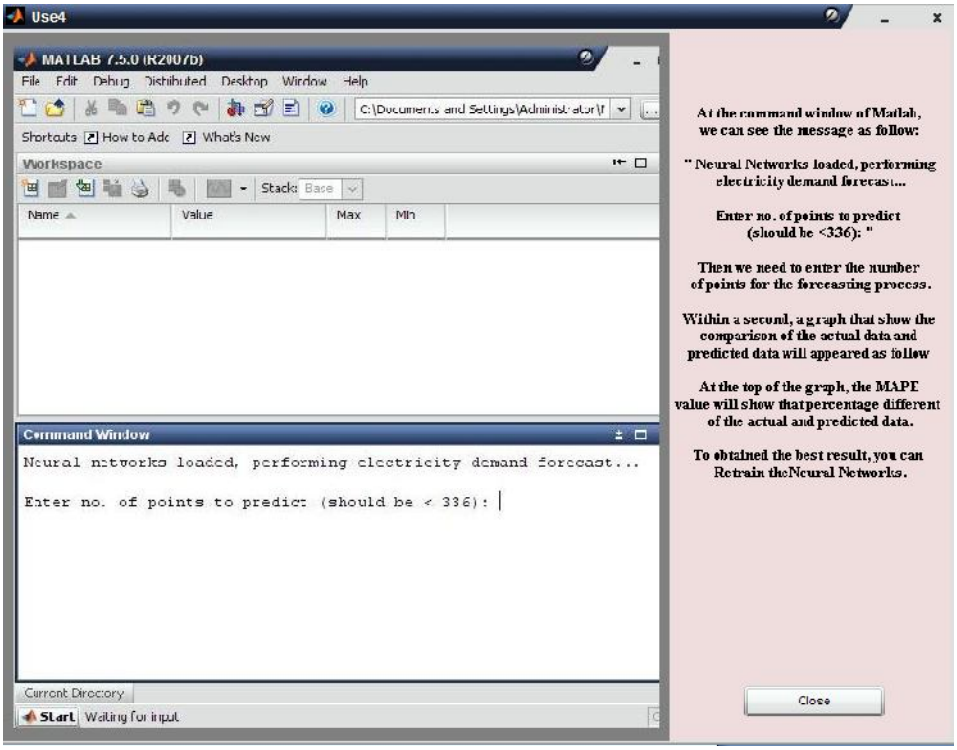

Figure 45: Use4

If the Start Button is clicked, the Short-term Load Forecasting system is executed. On the other hand, if the Credit Button is clicked, the Figure 46 is shown.

Figure 46: Credit

To exit the program, press the Exit button, and a message showing “Are you sure want to quit” will pop out, user can choose to click ‘Yes’ for exit and ‘No’ to return to the Main program.

7.4 Discussions

The table below shows the result obtained by using the Neural Network model which tested with six sets of historical data containing the electricity load and temperature data for the month of January 2009, on a half-hourly basis; one set of electricity load data of Queensland and five sets of temperature data from five different locations (higher power consumption regions) of Queensland.

No. of Demand dataPoints	MAPE(%)
1 day (48 Points)	2.2055%
2 days (96 Points)	1.9467%
3 days (144 Points)	1.8520%
4 days (192 Points)	1.8709%
5 days (240 Points)	1.9171%
6 days (288 Points)	1.9605%
1 week (336 Points)	1.9929%

Table 4: The Mean Absolute Error obtained from the forecasting neural network model

By referring to the Mean Absolute Error obtained from the forecasting of neural network model, we can see that all the MAPE obtained are less than 2.3%. This means that the accuracy of the Short-term neural network obtained have a very high accuracy which is up to 97.7%. Therefore this short-term neural network model can be used to provide a more reliable electricity generation to the load.

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS

8.1 Conclusion

This thesis proposed a STLF model with a high forecasting accuracy. The NWT has been successfully implemented in the model. The implementation of NWT has reasonably enhanced the learning capability of the NNs in the model, thus minimizing their training frequencies as shown in the simulations.

Temperature has a close relationship with electricity load and its feasibility to be included for STLF has been proven with reasonable accuracy achieved from the simulations. However, when temperature data is included, the model is found to be able to forecast only one point ahead. Nevertheless, this limitation can easily be overcome. The details are discussed as future improvements in the next section.

In summary, the inclusion of temperature data (as an additional input variable) and the use of NWT (as the data processing tool) for the proposed STLF model have been a success.

8.2 Recommendations for Future Work

The limitation as mentioned earlier can be easily resolved. Having to forecast more than one point ahead requires additional NNs to forecast the temperature series that are used in the model.

To be able to predict more than one point ahead, the model is suggested to include another fifteen NNs. In other words, instead of training only one NN to predict an output of load series, fifteen additional NNs (three NNs to predict one temperature) are trained to predict the five sets of temperature series as well. In this way, the first set of predicted values (one load and five temperatures) can be used to represent the second set of inputs to predict the second point of load value, and the sequence goes on. The suggested model is also anticipated to produce a further improved accuracy for the short-term load forecasting of one point ahead.

The Neural Network model discusses above is only a suggestion; other types of NNs (e.g. recurrent NNs) should also be explored in detail for future improvements. Technology evolves with time; perhaps new methods and algorithms would be formulated in the near future to improve the STLF model. Thus to predict more points ahead with satisfactory results, future researched methodologies should also be considered.

REFERENCES

[1] R. Polikar, Rowan University website, “The Wavelet Tutorial (Part I),” Fundamental Concepts & An Overview of the Wavelet Theory (2 n), http://engineering.rowan.edu/~polikar/WAVELETS/WTpart1.html (current Jun. 1996) [20] I. Nabney, “Netlab,” Netlab

[2] A. Louis et al, “Wavelets Theory and Applications,” John Wiley & Sons Ltd, England, 1997

[3] W. Charytoniuk, M.S. Chen, and P. Van Olinda. Nonparametric Regression Based Short-Term Load Forecasting. *IEEE Transactions on Power Systems*, 13:725–730, 1998.

[4] T. Down, .Introduction to Neural Computing & Pattern Recognition, http://www.csee.uq.edu.au/~comp3700lectures/lecture1.pdf (current Jul. 2003)

[5] G.Janacek, L.Swift, *Time series: forecasting, simulation, applications*, West Sussex: Ellis Horwood Limited, 1993.

[6] H. S. Hippert et al, .Neural Networks for Short-Term Load Forecasting: A Review and Evaluation,. IEEE Trans. on Power Systems, Vol. 16, No. 1, Feb. 2001, pp. 44-54

[7] R.F. Engle, C. Mustafa, and J. Rice. Modeling Peak Electricity Demand. *Journal of Forecasting*, 11:241–251, 1992.

[8] D. K. Ranaweera et al, “Effect of probabilistic inputs on neural network-based electric load forecasting,” IEEE Trans. on Neural Networks, Vol. 7, No. 6, Nov. 1996, pp. 1528-1532 References 90

[9] A. D. Papalexopoulos and T. C. Hesterberg, “A regression-based approach to short-term system load forecasting,” IEEE Trans. on Power Systems, Vol. 5, No. 4, Nov. 1990, pp. 1535-1547

[10] T. Haida and S. Muto. Regression Based Peak Load Forecasting using a Transformation Technique. *IEEE Transactions on Power Systems*, 9:1788–1794, 1994.

[11] K.L. Ho, Y.Y. Hsu, F.F. Chen, T.E. Lee, C.C. Liang, T.S. Lai, and K.K. Chen. Short-Term Load Forecasting of Taiwan Power System using a Knowledge Based Expert System. *IEEE Transactions on Power Systems*, 5:1214–1221, 1990.

[12] O. Hyde and P.F. Hodnett. An Adaptable Automated Procedure for Short-Term Electricity Load Forecasting. *IEEE Transactions on Power Systems*, 12:84–93, 1997.

[13] S.J. Kiartzis and A.G. Bakirtzis. A Fuzzy Expert System for Peak Load Forecasting: Application to the Greek Power System. *Proceedings of the 10th Mediterranean Electrotechnical Conference*, 3:1097– 1100, 2000.

[14] V. Miranda and C. Monteiro. Fuzzy Inference in Spatial Load Forecasting. *Proceedings of IEEE Power Engineering Winter Meeting*, 2:1063–1068, 2000.

[15] M. Peng, N.F. Hubele, and G.G. Karady. Advancement in the Application of Neural Networks for Short-Term Load Forecasting. *IEEE Transactions on Power Systems*, 7:250–257, 1992.

[16] S. Ruzic, A. Vuckovic, and N. Nikolic. Weather Sensitive Method for Short-Term Load Forecasting in Electric Power Utility of Serbia. *IEEE Transactions on Power Systems*, 18:1581–1586, 2003

[17] S.E. Skarman and M. Georgiopoulous. Short-Term Electrical Load Forecasting using a Fuzzy ARTMAP Neural Network. *Proceedings of SPIE*, 181–191, 1998.

[18] T. Down, "Introduction to Neural Computing & Pattern Recognition," COMP3700 Machine Learning website, University of Queensland, Australia, http://www.csee.uq.edu.au/~comp3700/lectures/lecture1.pdf (current Jul. 2002)

[19] S. Haykin, "Neural Networks," Macmillan College Publishing Company, Inc. 1994

[20] N. K. Bose and P. Liang, "Neural Network Fundamentals with Graphs, Algorithms, and Applications," McGraw-Hill Electrical and Computer Engineering Series, McGraw-Hill, Inc. 1996

[21] R. Bharath and J. Drosen, "Neural Network Computing," Windcrest, USA, 1994

APPENDIX A

Program for Short-term Load Forecasting System

Main.m file

```
function varargout = Main(varargin)
% MAIN M-file for Main.fig
%      MAIN, by itself, creates a new MAIN or raises the existing
%      singleton*.
%
%      H = MAIN returns the handle to a new MAIN or the handle to
%      the existing singleton*.
%
%      MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MAIN.M with the given input
arguments.
%
%      MAIN('Property','Value',...) creates a new MAIN or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Main_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Main_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Main

% Last Modified by GUIDE v2.5 10-Oct-2009 19:45:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Main_OpeningFcn, ...
                   'gui_OutputFcn',  @Main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Main is made visible.
function Main_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Main (see VARARGIN)

% Choose default command line output for Main
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('PowerPlant','jpg');
image(x)
set(gca,'visible','off')

% UIWAIT makes Main wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Main_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
figure(Introduction)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
figure(Use1)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
clc;
clear;
%Obtain user desired resolution level (Tested: resolution = 2 is the
best)
level = menu('Enter desired resolution level: ', '1', '2 <= (For
testing, please select this!!)', '3', '4');
switch level
case 1, resolution = 1;
case 2, resolution = 2;
case 3, resolution = 3;
case 4, resolution = 4;
```

```
end
msg = ['Resolution level to be used is ', num2str(resolution)];
disp(msg);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Obtain user desired amount of data to use
data = menu('Choose amount of data to use: ', '1 day', '2 days', '3
days', '4 days',...
'5 days', '6 days', '1 week <= (For testing,please select this!!)', '2
weeks');
switch data
case 1, dataPoints = 48; %1 day = 48 points
case 2, dataPoints = 96; %2 days = 96 points
case 3, dataPoints = 144; %3 days = 144 points
case 4, dataPoints = 192; %4 days = 192 points
case 5, dataPoints = 240; %5 days = 240 points
case 6, dataPoints = 288; %6 days = 288 points
case 7, dataPoints = 336; %1 week = 336 points
case 8, dataPoints = 672; %2 weeks = 672 points
end
msg = ['No. of data points to be used is ', num2str(dataPoints)];
disp(msg);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Menu for data set selection
select = menu('Use QLD data of: ', 'Jan09 <= (For testing,please select
this!!)',...
'Feb09', 'Mar09');
switch select
case 1, demandFile = 'DATA200901_QLD1', brisTempFile =
'BRISBANE200901'...
, goldcstTempFile = 'GOLDCST200901'...
, townsvilTempFile = 'TOWNSVIL200901'...
, cairnsTempFile = 'CAIRNS200901', rockyTempFile = 'ROCKY200901';
case 2, demandFile = 'DATA200902_QLD1', brisTempFile =
'BRISBANE200902'...
, goldcstTempFile = 'GOLDCST200902'...
, townsvilTempFile = 'TOWNSVIL200902'...
, cairnsTempFile = 'CAIRNS200902', rockyTempFile = 'ROCKY200902';
case 3, demandFile = 'DATA200903_QLD1', brisTempFile =
'BRISBANE200903'...
, goldcstTempFile = 'GOLDCST200903'...
, townsvilTempFile = 'TOWNSVIL200903'...
, cairnsTempFile = 'CAIRNS200903', rockyTempFile = 'ROCKY200903';
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical DEMAND data into tDemandArray
selectedDemandFile = [demandFile, '.csv'];
[regionArray, sDateArray, tDemandArray, rrpArrray, pTypeArray,tTime]
...
= textread(selectedDemandFile, '%s %q %f %f %s %f','headerlines', 1,
'delimiter',',');
%Display no. of points in the selected time series demand data

[demandDataPoints,y] = size(tDemandArray);
msg = ['The no. of points in the selected Demand data is ',
num2str(demandDataPoints)];
disp(msg);
%Decompose historical demand data signal
```

```
[dD, l] = swt(tDemandArray, resolution, 'db2');
approx = dD(resolution, :);
%Plot the original demand data signal
figure (1);
subplot(resolution + 2, 1, 1); plot(tDemandArray(1: dataPoints))
legend('Demand Original');
title('QLD Demand Data Signal');
%Plot the approximation demand data signal
for i = 1 : resolution
subplot(resolution + 2, 1, i + 1); plot(approx(1: dataPoints))
legend('Demand Approximation');
end
%After displaying approximation signal, display detail x
for i = 1: resolution
if( i > 1 )
detail(i, :) = dD(i-1, :)- dD(i, :);
else
detail(i, :) = tDemandArray' - dD(1, :);
end
subplot(resolution + 2, 1, resolution - i + 3); plot(detail(i, 1:
dataPoints))
legendName = ['Demand Detail ', num2str(i)];
legend(legendName);
i = i + 1;
end
%Normalising approximation demand data
maxDemand = max(approx'); %Find largest component
normDemand = approx ./ maxDemand; %Right divison
maxDemandDetail = [ ];
normDemandDetail = [ ];
detail = detail + 4000;
for i = 1: resolution
maxDemandDetail(i) = max(detail(i, :));
normDemandDetail(i, :) = detail(i, :) ./maxDemandDetail(i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical Brisbane's TEMPERATURE data into temp1Array
selectedTempFile1 = [brisTempFile, '.csv'];
[dTimeArray, temp1Array] ...
= textread(selectedTempFile1, '%s %f', 'headerlines', 1, 'delimiter',
',');
%Display no. of points in Brisbane's temperature data
[noOfDataPoints, y] = size(temp1Array);
msg = ['The no. of points in Brisbane Temperature data is ',
num2str(noOfDataPoints)];
disp(msg);
%Decompose historical Brisbane's temperature data signal
[dT1, l] = swt(temp1Array, resolution, 'db2');
approxT1 = dT1(resolution, :);
%Plot the original Brisbane's temperature data signal
figure (2);
subplot(resolution + 2, 1, 1); plot(temp1Array(1: dataPoints))
legend('T1 Original');
title('BRISBANE Temperature Data Signal');
%Plot the approximation Brisbane's temperature data signal
for i = 1 : resolution
subplot(resolution + 2, 1, i + 1); plot(approxT1(1: dataPoints))
```

```
legend('T1 Approximation');
end
%After displaying approximation signal, display detail x
for i = 1: resolution
if( i > 1 )
detailT1(i, :) = dT1(i-1, :)- dT1(i, :);
else
detailT1(i, :) = temp1Array' - dT1(1, :);
end
subplot(resolution + 2, 1, resolution - i + 3); plot(detailT1(i, 1:
dataPoints))
legendName = ['T1 Detail ', num2str(i)];
legend(legendName);
i = i + 1;
end
%Normalising approximation Brisbane's temperature data
maxBrisTemp = max(approxT1'); %Find largest component
normBrisTemp = approxT1 ./ maxBrisTemp; %Right divison
maxBrisTempDetail = [ ];
normBrisTempDetail = [, ];
detailT1 = detailT1 + 40;
for i = 1: resolution
maxBrisTempDetail(i) = max(detailT1(i, :));
normBrisTempDetail(i, :) = detailT1(i, :) ./maxBrisTempDetail(i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical GoldCoast's TEMPERATURE data into temp2Array
selectedTempFile2 = [goldcstTempFile, '.csv'];
[dTimeArray, temp2Array] ...
= textread(selectedTempFile2, '%s %f', 'headerlines', 1, 'delimiter',
',');
%Display no. of points in GoldCoast's temperature data
[noOfDataPoints, y] = size(temp2Array);
msg = ['The no. of points in Gold Coast Temperature data is ',
num2str(noOfDataPoints)];
disp(msg);
%Decompose historical GoldCoast's temperature data signal
[dT2, l] = swt(temp2Array, resolution, 'db2');
approxT2 = dT2(resolution, :);
%Plot the original GoldCoast's temperature data signal
figure (3);
subplot(resolution + 2, 1, 1); plot(temp2Array(1: dataPoints))
legend('T2 Original');
title('GOLD COAST Temperature Data Signal');
%Plot the approximation GoldCoast's temperature data signal
for i = 1 : resolution
subplot(resolution + 2, 1, i + 1); plot(approxT2(1: dataPoints))
legend('T2 Approximation');
end
%After displaying approximation signal, display detail x
for i = 1: resolution
if( i > 1 )
detailT2(i, :) = dT2(i-1, :)- dT2(i, :);
else
detailT2(i, :) = temp2Array' - dT2(1, :);
end
```

```
subplot(resolution + 2, 1, resolution - i + 3); plot(detailT2(i, 1:
dataPoints))
legendName = ['T2 Detail ', num2str(i)];
legend(legendName);
i = i + 1;
end
%Normalising approximation GoldCoast's temperature data
maxGoldCstTemp = max(approxT2'); %Find largest component
normGoldCstTemp = approxT2 ./ maxGoldCstTemp; %Right divison
maxGoldCstTempDetail = [ ];
normGoldCstTempDetail = [, ];
detailT2 = detailT2 + 40;
for i = 1: resolution
maxGoldCstTempDetail(i) = max(detailT2(i, :));
normGoldCstTempDetail(i, :) = detailT2(i, :) ./maxGoldCstTempDetail(i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical Townsville's TEMPERATURE data into temp3Array
%and applied a weightage of 0.95
selectedTempFile3 = [townsvilTempFile, '.csv'];
[dTimeArray, temp3Array] ...
= textread(selectedTempFile3, '%s %f', 'headerlines', 1, 'delimiter',
',');
%Display no. of points in Townsville's temperature data
[noOfDataPoints, y] = size(temp3Array);
msg = ['The no. of points in Townsville Temperature data is ',
num2str(noOfDataPoints)];
disp(msg);
%Decompose historical Townsville's temperature data signal
[dT3, l] = swt(temp3Array, resolution, 'db2');
approxT3 = dT3(resolution, :);
%Plot the original Townsville's temperature data signal
figure (4);
subplot(resolution + 2, 1, 1); plot(temp3Array(1: dataPoints))
legend('T3 Original');
title('TOWNSVILLE Temperature Data Signal');
%Plot the approximation Townsville's temperature data signal
for i = 1 : resolution
subplot(resolution + 2, 1, i + 1); plot(approxT3(1: dataPoints))
legend('T3 Approximation');
end
%After displaying approximation signal, display detail x
for i = 1: resolution
if( i > 1 )
detailT3(i, :) = dT3(i-1, :)- dT3(i, :);
else
detailT3(i, :) = temp3Array' - dT3(1, :);
end
subplot(resolution + 2, 1, resolution - i + 3); plot(detailT3(i, 1:
dataPoints))
legendName = ['T3 Detail ', num2str(i)];
legend(legendName);
i = i + 1;
end
%Normalising approximation Townsville's temperature data
maxTownsvilTemp = max(approxT3'); %Find largest component
normTownsvilTemp = approxT3 ./ maxTownsvilTemp; %Right divison
```

```
normTownsvilTemp = normTownsvilTemp * 0.95; % Apply weightage
maxTownsvilTempDetail = [ ];
normTownsvilTempDetail = [, ];
detailT3 = detailT3 + 40;
for i = 1: resolution
maxTownsvilTempDetail(i) = max(detailT3(i, :));
normTownsvilTempDetail(i, :) = detailT3(i, :)
./maxTownsvilTempDetail(i);
end
normTownsvilTempDetail = normTownsvilTempDetail * 0.95;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical Cairns's TEMPERATURE data into temp4Array
selectedTempFile4 = [cairnsTempFile, '.csv'];
[dTimeArray, temp4Array] ...
= textread(selectedTempFile4, '%s %f', 'headerlines', 1, 'delimiter',
',');
%Display no. of points in Cairns's temperature data
[noOfDataPoints, y] = size(temp4Array);
msg = ['The no. of points in Cairns Temperature data is ' ,
num2str(noOfDataPoints)];
disp(msg);
%Decompose historical Cairns's temperature data signal
[dT4, l] = swt(temp4Array, resolution, 'db2');
approxT4 = dT4(resolution, :);
%Plot the original Cairns's temperature data signal
figure (5);
subplot(resolution + 2, 1, 1); plot(temp4Array(1: dataPoints))
legend('T4 Original');
title('CAIRNS Temperature Signal');
%Plot the approximation Cairns's temperature data signal
for i = 1 : resolution
subplot(resolution + 2, 1, i + 1); plot(approxT4(1: dataPoints))
legend('T4 Approximation');
end
%After displaying approximation signal, display detail x
for i = 1: resolution
if( i > 1 )
detailT4(i, :) = dT4(i-1, :)- dT4(i, :);
else
detailT4(i, :) = temp4Array' - dT4(1, :);
end
subplot(resolution + 2, 1, resolution - i + 3); plot(detailT4(i, 1:
dataPoints))
legendName = ['T4 Detail ', num2str(i)];
legend(legendName);
i = i + 1;
end
%Normalising approximation Cairns's temperature data
maxCairnsTemp = max(approxT4'); %Find largest component
normCairnsTemp = approxT4 ./ maxCairnsTemp; %Right divison
normCairnsTemp = normCairnsTemp * 0.90; % Apply weightage
maxCairnsTempDetail = [ ];
normCairnsTempDetail = [, ];
detailT4 = detailT4 + 40;
for i = 1: resolution
maxCairnsTempDetail(i) = max(detailT4(i, :));
normCairnsTempDetail(i, :) = detailT4(i, :) ./maxCairnsTempDetail(i);
```

```
end
normCairnsTempDetail = normCairnsTempDetail * 0.90;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical Rockhampton's TEMPERATURE data into temp5Array
selectedTempFile5 = [rockyTempFile, '.csv'];
[dTimeArray, temp5Array] ...
= textread(selectedTempFile5, '%s %f', 'headerlines', 1, 'delimiter',
',');
%Display no. of points in Rockhampton's temperature data
[noOfDataPoints, y] = size(temp5Array);
msg = ['The no. of points in Rockhampton Temperature data is ',
num2str(noOfDataPoints)];
disp(msg);
%Decompose historical Rockhampton's temperature data signal
[dT5, l] = swt(temp5Array, resolution, 'db2');
approxT5 = dT5(resolution, :);
%Plot the original Rockhampton's temperature data signal
figure (6);
subplot(resolution + 2, 1, 1); plot(temp5Array(1: dataPoints))
legend('T5 Original');
title('ROCKHAMPTON Temperature Signal');
%Plot the approximation Rockhampton's temperature data signal
for i = 1 : resolution
subplot(resolution + 2, 1, i + 1); plot(approxT5(1: dataPoints))
legend('T5 Approximation');
end
%After displaying approximation signal, display detail x
for i = 1: resolution
if( i > 1 )
detailT5(i, :) = dT5(i-1, :)- dT5(i, :);
else
detailT5(i, :) = temp5Array' - dT5(1, :);
end
subplot(resolution + 2, 1, resolution - i + 3); plot(detailT5(i, 1:
dataPoints))
legendName = ['T5 Detail ', num2str(i)];
legend(legendName);
i = i + 1;
end
%Normalising approximation Rockhampton's temperature data
maxRockyTemp = max(approxT5'); %Find largest component
normRockyTemp = approxT5 ./ maxRockyTemp; %Right divison
normRockyTemp = normRockyTemp * 0.85; % Apply weightage
maxRockyTempDetail = [ ];
normRockyTempDetail = [, ];
detailT5 = detailT5 + 40;
for i = 1: resolution
maxRockyTempDetail(i) = max(detailT5(i, :));
normRockyTempDetail(i, :) = detailT5(i, :) ./maxRockyTempDetail(i);
end
normRockyTempDetail = normRockyTempDetail * 0.85;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function here allows repetitive options to,
% 1) Create new NNs, 2) Retrain the existing NNs,
% 3) Perform load demand forecasting and 4) Quit session
resultGraphNum = 8; %For multiple forecast use
```

```
while (1)
%Main MENU
choice = menu('Please select one of the following options: ',...
'CREATE new Neural Networks',...
'RETRAIN existing Neural Networks',...
'Perform FORECASTING of load demand', 'QUIT session...');
switch choice
case 1, scheme = 'c';
case 2, scheme = 'r';
case 3, scheme = 'f';
case 4, scheme = 'e';
end
%If scheme is 'c', call <nnCreate> to create new NNs, train them then
perform forecast
if(scheme == 'c')
nnCreate;
end
%If scheme is 'r', call <nnRetrain> to retrain the existing NNs
if(scheme == 'r')
nnRetrain;
end
%If scheme is 'f', call <nnForecast> to load the existing NN model
if(scheme == 'f')
nnforecast;
end
%If scheme is 'e', verifies and quit session if 'yes' is selected else
continue
if(scheme == 'e')
button = questdlg('Quit session?', 'Exit Dialog','Yes','No','No');
switch button
case 'Yes', disp(' ');
disp('Session has ended!!');
disp(' ');
break;
case 'No', quit cancel;
end
end
end
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 % --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
figure(Credit)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
button=questdlg('Are you sure want to quit?','Exit','Yes','No','No');
switch button
    case 'Yes',
        close all
case 'No',
        quit cancel;end
```

nnCreate.m file

```
%Subsystem : Data pre-processing / Creates then matures the NNs
%File name : nnCreate.m
%Author : Tai Hein Fong
%Date : July 2009
%Description : This file prepares the input & output data for the NNs.
It creates %then trains the NNs to mature them. When all is done,
program will return to the Main %Menu for other options.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clear command screen and set target demand ouput to start at point 2
clc;
targetStartAt = 2;
disp('Program will now CREATE a Neural Network for training and
forecasting...');
disp(' ');
disp('To capture the pattern of the signal, the model is ')
disp('set to accept dataPoints x 8 sets of training examples; ');
disp('1 set of demand + 7 sets of temp (1 set for each 7 different
locations). ');
disp(' ');
disp('The normalised demand data <point 2>, is to be taken as the ')
disp('output value for the first iteration of training examples. ');
disp(' ');
disp('Press ENTER key to continue...');
numOfNNs = resolution + 1;
%Creating and training the NNs for the respective
%demand and temperature coefficient signals
for x = 1: numOfNNs
%Clearing variables
clear targetDemand;
clear inputs;
clear output;
clc;
if(x == 1)
    neuralNetwork = ['Neural network settings for approximation level
',...
num2str(resolution)];
else
neuralNetwork = ['Neural network settings for detail level ', num2str(x
- 1)];
end
disp(neuralNetwork);
disp(' ');
%Set no. of input nodes and hidden neurons for the
%respective demand and temperature coefficient signal
numOfInputs = 6;
inputValue = ['Number of neural network INPUT units is set at ',...
num2str(numOfInputs)];
disp(inputValue);
disp(' ');
numOfOutput = 1;
outValue = ['Output is set to ', num2str(numOfOutput)];
disp(outValue);
disp(' ');
numOfHiddens =3;
```

```
hiddenValue = ['Number of neural network HIDDEN units is set at
',num2str(numOfHiddens)];
disp(hiddenValue);
disp(' ');
%Setting no. of training examples
trainingLength = dataPoints;
%Set target outputs of the training examples
if(x == 1)
targetDemand = normDemand(targetStartAt: 1 + trainingLength);
else
targetDemand = normDemandDetail(x - 1, targetStartAt: 1 +
trainingLength);
end
%Preparing training examples
%Setting training i/ps to be 8 with user defined no. of iterations
(dataPoints)
y = 0;
while y < trainingLength
if(x == 1)
inputs(1, y + 1) = normDemand(y + 1);
inputs(2, y + 1) = normBrisTemp(y + 1);
inputs(3, y + 1) = normGoldCstTemp(y + 1);
inputs(4, y + 1) = normTownsvilTemp(y + 1);
inputs(5, y + 1) = normCairnsTemp(y + 1);
inputs(6, y + 1) = normRockyTemp(y + 1);
else
inputs(1, y + 1) = normDemandDetail(x - 1, y + 1);
inputs(2, y + 1) = normBrisTempDetail(x - 1, y + 1);
inputs(3, y + 1) = normGoldCstTempDetail(x - 1, y + 1);
inputs(4, y + 1) = normTownsvilTempDetail(x - 1, y + 1);
inputs(5, y + 1) = normCairnsTempDetail(x - 1, y + 1);
inputs(6, y + 1) = normRockyTempDetail(x - 1, y + 1);
end
output(y + 1, :) = targetDemand(y + 1);
y = y + 1;
end
inputs = (inputs');
%With reference to line 161
%Create new neural network for respective coefficient signal
%NET = MLP(NIN, NHIDDEN, NOUT, FUNC)
net =mlp(numOfInputs,numOfHiddens,numOfOutput,'linear');
%NN options
options = zeros(1, 18);
options(1) = 1; %Provides display of error values
options(14) = 100;%No of cycles
%Training the neural network
%nnopt(net, options, x, t, alg, len);
[net,options] = netopt(net, options, inputs, output, 'scg');
%Save the neural network
filename = ['net', num2str(x)];
save(filename, 'net');
disp(' ');
msg = ['Neural network successfully CREATED and saved as => ',
filename];
disp(msg);
if(x < 1)
disp(' ');
```

```
disp('Press ENTER key to continue training the next NN...');
else
disp(' ');
disp('Model is now ready to forecast!!');
disp(' ');
end
```

nnRetrain.m file

```
%Subsystem : Retrain the NNs
%File name : nnRetrain.m
%Date : July 2009
%Description : This file loads the existing NNs and trains them again.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clear command screen and display msgs
clc;
disp('Program will now RETRAIN the Neural Networks ')
disp('with the SAME initial data series again...');
disp(' ');
disp('To capture the pattern of the signal, the model is ')
disp('set to accept dataPoints x 8 sets of training examples; ');
disp('1 set of demand + 7 sets of temp (1 set for each 7 different
locations). ');
disp(' ');
disp('The normalised demand data <point 2>, is to be taken as the ')
disp('output value for the first iteration of training examples. ');
disp(' ');
msg = ['Data points to be used for reTraining the NNs is from 1 to
',...
    num2str(dataPoints)];
disp(msg);
disp(' ');
disp('Press ENTER key to continue...');
pause;
%Clear command screen
clc;
%Prompt for no. of training cycles
%For current program, 1 cycle = user set no. of iterations (ie:
dataPoints)
cycle = input('Input number of cycles to retrain the NNs: ');
numOfNNs = resolution + 1;
%Loading existing NNs for training
for x = 1: numOfNNs
%Re-initialising variables
clear targetDemand;
clear inputs;
clear output;
clc;
%Loading NN for the respective demand and temperature coefficient
signals
filename = ['net', num2str(x)];
clear net
load(filename);
%Getting the size of NN
numOfInputs = net.nin;
```

```
numOfHiddens = net.nhidden;
numOfOutput = 1;
%Setting length of reTraining examples and target outputs
reTrainLength = dataPoints;
targetLength = reTrainLength;
targetStartAt = 2;
%Set target outputs of the training examples
if(x == 1)
targetDemand = normDemand(targetStartAt: 1 + targetLength);
else
targetDemand = normDemandDetail(x - 1, targetStartAt: 1 +
targetLength);
end
%Preparing training examples
%Setting training i/ps to be 8 with user set no. of iterations
(dataPoints)
y = 0;
while y < reTrainLength
if(x == 1)
inputs(1, y + 1) = normDemand(y + 1);
inputs(2, y + 1) = normBrisTemp(y + 1);
inputs(3, y + 1) = normGoldCstTemp(y + 1);
inputs(4, y + 1) = normTownsvilTemp(y + 1);
inputs(5, y + 1) = normCairnsTemp(y + 1);
inputs(6, y + 1) = normRockyTemp(y + 1);

else
inputs(1, y + 1) = normDemandDetail(x - 1, y + 1);
inputs(2, y + 1) = normBrisTempDetail(x - 1, y + 1);
inputs(3, y + 1) = normGoldCstTempDetail(x - 1, y + 1);
inputs(4, y + 1) = normTownsvilTempDetail(x - 1, y + 1);
inputs(5, y + 1) = normCairnsTempDetail(x - 1, y + 1);
inputs(6, y + 1) = normRockyTempDetail(x - 1, y + 1);

end
output(y + 1, :) = targetDemand(y + 1);
y = y + 1;
end
inputs = (inputs');
%Setting no. of training cycles
[ni, np] = size(targetDemand); % <== [ni, np] tells the NN how long is
1 cycle;
size(targetDemand) %With reference to line 106
%NN options
options = zeros(1, 18);
options(1) = 1; %Provides display of error values
options(14) = 100
%Training the neural network
%nnopt(net, options, x, t, alg, len);
net = netopt(net, options, inputs, output, 'scg');
%Save the neural network
filename = ['net', num2str(x)];
save(filename, 'net');
disp(' ');
msg = ['Neural network => ', filename, ' <= successfully RETRAINED and
saved!! '];
disp(msg);
```

```
if(x < 3)
disp(' ');
disp('Press ENTER key to continue training the next NN...');
else
disp(' ');
disp('Model is now ready to forecast again!!');
disp(' ');
disp('Press ENTER key to continue...');
end
pause;
end
```

nnForecast.m file

```
%Subsystem : Forecasting / Data post-processing
%File name : nnForecast.m
%Date : July 2009
%Description : This file loads the trained NNs for load forcasting and
%recombines the predicted outputs from the NNs to form the final
predicted load series.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clear command screen and variables
clc;
clear forecastResult;
clear actualDemand;
clear predicted;
clear actualWithPredicted;
disp('Neural networks loaded, performing electricity demand
forecast...');
disp(' ');
pointsAhead = input('Enter no. of points to predict (should be < 336):
');
numOfTimes = resolution + 1;
%Predict coefficient signals using respective NNs
for x = 1 : numOfTimes
%Loading NN
filename = ['net', num2str(x)];
clear net
load(filename);
clear in;
numOfInputs = net.nin;
y = 0;
%Preparing details to forecast
while y < pointsAhead
if(x == 1)
propData(1, y + 1) = normDemand(y + 1);
propData(2, y + 1) = normBrisTemp(y + 1);
propData(3, y + 1) = normGoldCstTemp(y + 1);
propData(4, y + 1) = normTownsvilTemp(y + 1);
propData(5, y + 1) = normCairnsTemp(y + 1);
propData(6, y + 1) = normRockyTemp(y + 1);

else
propData(1, y + 1) = normDemandDetail(x - 1, y + 1);
```

```
propData(2, y + 1) = normBrisTempDetail(x - 1, y + 1);
propData(3, y + 1) = normGoldCstTempDetail(x - 1, y + 1);
propData(4, y + 1) = normTownsvilTempDetail(x - 1, y + 1);
propData(5, y + 1) = normCairnsTempDetail(x - 1, y + 1);
propData(6, y + 1) = normRockyTempDetail(x - 1, y + 1);

end
y = y + 1;
end
if(x == 1)
forecast = mlpfwd(net, propData') * maxDemand;
else
forecast = mlpfwd(net, propData') * maxDemandDetail(x - 1)-4000;
end
forecastResult(x, :) = forecast;
end
actualDemand = tDemandArray(2: 1 + pointsAhead);
predicted = sum(forecastResult, 1)';
%Calculating Mean Absolute Error
AbsError = abs(actualDemand - predicted(1: pointsAhead)) ./
actualDemand;
msg = ['Mean Absolute Percentage Error = ',...
num2str(mean(AbsError(1: pointsAhead))*100), ' !!'];
disp(' ');
disp(msg);
%Plot actual time series against predicted result
resultGraphNum = resultGraphNum + 1;
figure(resultGraphNum)
actualWithPredicted(:, 1) = actualDemand;
actualWithPredicted(:, 2) = predicted(1: pointsAhead);
plot(actualWithPredicted(:, 1), 'b');
hold on;
plot(actualWithPredicted(:, 2), 'r:');
hold off;
graph = ['Actual VS Forecasted ', ' < MAPE = ',
num2str(mean(AbsError)*100), '% >'];
title(graph);
%xaxis = ['No. of Points shown: ' , num2str(pointsAhead)];
xaxis = ['Time (half hour intervals)'];
xlabel(xaxis);
yaxis = ['Total Demand'];
ylabel(yaxis);
legend('Actual', 'Forecasted');
```

Introduction.m file

```
function varargout = Introduction(varargin)
% INTRODUCTION M-file for Introduction.fig
%      INTRODUCTION, by itself, creates a new INTRODUCTION or raises
the existing
%      singleton*.
%
%      H = INTRODUCTION returns the handle to a new INTRODUCTION or the
handle to
%      the existing singleton*.
%
%      INTRODUCTION('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in INTRODUCTION.M with the given input
arguments.
%
%      INTRODUCTION('Property','Value',...) creates a new INTRODUCTION
or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Introduction_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Introduction_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Introduction

% Last Modified by GUIDE v2.5 10-Oct-2009 20:07:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Introduction_OpeningFcn, ...
                   'gui_OutputFcn',  @Introduction_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before Introduction is made visible.
function Introduction_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Introduction (see VARARGIN)

% Choose default command line output for Introduction
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('Transmissionline','jpg');
image(x)
set(gca,'visible','off')

% UIWAIT makes Introduction wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Introduction_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Use1.m file

```
function varargout = Use1(varargin)
% USE1 M-file for Use1.fig
%      USE1, by itself, creates a new USE1 or raises the existing
%      singleton*.
%
%      H = USE1 returns the handle to a new USE1 or the handle to
%      the existing singleton*.
%
%      USE1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in USE1.M with the given input
arguments.
%
```

```
%      USE1('Property','Value',...) creates a new USE1 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Use1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Use1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Use1

% Last Modified by GUIDE v2.5 27-Oct-2009 22:24:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Use1_OpeningFcn, ...
                   'gui_OutputFcn',  @Use1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Use1 is made visible.
function Use1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Use1 (see VARARGIN)

% Choose default command line output for Use1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('Use1','jpg');
image(x)
set(gca,'visible','off')

% UIWAIT makes Use1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Use1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
figure(Use2)

% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as
a double
 % --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');end
```

Use2.m file

```
function varargout = Use2(varargin)
% USE2 M-file for Use2.fig
%      USE2, by itself, creates a new USE2 or raises the existing
%      singleton*.
%
%      H = USE2 returns the handle to a new USE2 or the handle to
%      the existing singleton*.
%
%      USE2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in USE2.M with the given input
arguments.
```

```
%
%      USE2('Property','Value',...) creates a new USE2 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Use2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Use2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Use2

% Last Modified by GUIDE v2.5 28-Oct-2009 21:07:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Use2_OpeningFcn, ...
                   'gui_OutputFcn',  @Use2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Use2 is made visible.
function Use2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Use2 (see VARARGIN)

% Choose default command line output for Use2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('Use2','jpg');
image(x)
set(gca,'visible','off')

% UIWAIT makes Use2 wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Use2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
figure(Use3)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Use3.m file

```
function varargout = Use3(varargin)
% USE3 M-file for Use3.fig
%      USE3, by itself, creates a new USE3 or raises the existing
%      singleton*.
%
%      H = USE3 returns the handle to a new USE3 or the handle to
%      the existing singleton*.
%
%      USE3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in USE3.M with the given input
arguments.
%
%      USE3('Property','Value',...) creates a new USE3 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Use3_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Use3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Use3
```

```
% Last Modified by GUIDE v2.5 27-Oct-2009 23:56:49

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Use3_OpeningFcn, ...
                   'gui_OutputFcn',  @Use3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Use3 is made visible.
function Use3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Use3 (see VARARGIN)

% Choose default command line output for Use3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('Use3','jpg');
image(x)
set(gca,'visible','off')

% UIWAIT makes Use3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Use3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as
a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
figure(Use4)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Use4.m file

```
function varargout = Use4(varargin)
% USE4 M-file for Use4.fig
%      USE4, by itself, creates a new USE4 or raises the existing
%      singleton*.
%
%      H = USE4 returns the handle to a new USE4 or the handle to
%      the existing singleton*.
%
%      USE4('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in USE4.M with the given input
arguments.
%
%      USE4('Property','Value',...) creates a new USE4 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Use4_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Use4_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help Use4

% Last Modified by GUIDE v2.5 28-Oct-2009 00:31:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Use4_OpeningFcn, ...
                   'gui_OutputFcn',  @Use4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Use4 is made visible.
function Use4_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Use4 (see VARARGIN)

% Choose default command line output for Use4
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('Use4','jpg');
image(x)
set(gca,'visible','off')


% UIWAIT makes Use4 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Use4_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
close all
figure (Main)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Credit.m file

```
function varargout = Credit(varargin)
% CREDIT M-file for Credit.fig
%      CREDIT, by itself, creates a new CREDIT or raises the existing
%      singleton*.
%
%      H = CREDIT returns the handle to a new CREDIT or the handle to
%      the existing singleton*.
%
%      CREDIT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CREDIT.M with the given input
arguments.
%
%      CREDIT('Property','Value',...) creates a new CREDIT or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Credit_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Credit_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Credit

% Last Modified by GUIDE v2.5 27-Oct-2009 19:28:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Credit_OpeningFcn, ...
                   'gui_OutputFcn',  @Credit_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
```

```
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Credit is made visible.
function Credit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Credit (see VARARGIN)

% Choose default command line output for Credit
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('Me','jpg');
image(x)
set(gca,'visible','off')

% UIWAIT makes Credit wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Credit_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as
a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Exit.m file

```
function varargout = Exit(varargin)
% EXIT M-file for Exit.fig
%      EXIT by itself, creates a new EXIT or raises the
%      existing singleton*.
%
%      H = EXIT returns the handle to a new EXIT or the handle to
%      the existing singleton*.
%
%      EXIT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in EXIT.M with the given input
arguments.
%
%      EXIT('Property','Value',...) creates a new EXIT or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Exit_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Exit_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Exit

% Last Modified by GUIDE v2.5 10-Oct-2009 21:33:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Exit_OpeningFcn, ...
                   'gui_OutputFcn',  @Exit_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Exit is made visible.
function Exit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to Exit (see VARARGIN)

% Choose default command line output for Exit
handles.output = 'Yes';

% Update handles structure
guidata(hObject, handles);

% Insert custom Title and Text if specified by the user
% Hint: when choosing keywords, be sure they are not easily confused
% with existing figure properties.  See the output of set(figure) for
% a list of figure properties.
if(nargin > 3)
    for index = 1:2:(nargin-3),
        if nargin-3==index, break, end
        switch lower(varargin{index})
         case 'title'
          set(hObject, 'Name', varargin{index+1});
         case 'string'
          set(handles.text1, 'String', varargin{index+1});
        end
    end
end

% Determine the position of the dialog - centered on the callback
figure
% if available, else, centered on the screen
FigPos=get(0,'DefaultFigurePosition');
OldUnits = get(hObject, 'Units');
set(hObject, 'Units', 'pixels');
OldPos = get(hObject,'Position');
FigWidth = OldPos(3);
FigHeight = OldPos(4);
if isempty(gcbf)
    ScreenUnits=get(0,'Units');
    set(0,'Units','pixels');
    ScreenSize=get(0,'ScreenSize');
    set(0,'Units',ScreenUnits);

    FigPos(1)=1/2*(ScreenSize(3)-FigWidth);
    FigPos(2)=2/3*(ScreenSize(4)-FigHeight);
else
    GCBFOldUnits = get(gcbf,'Units');
    set(gcbf,'Units','pixels');
    GCBFPos = get(gcbf,'Position');
    set(gcbf,'Units',GCBFOldUnits);
    FigPos(1:2) = [(GCBFPos(1) + GCBFPos(3) / 2) - FigWidth / 2, ...
                   (GCBFPos(2) + GCBFPos(4) / 2) - FigHeight / 2];
end
FigPos(3:4)=[FigWidth FigHeight];
set(hObject, 'Position', FigPos);
set(hObject, 'Units', OldUnits);

% Show a question icon from dialogicons.mat - variables questIconData
% and questIconMap
load dialogicons.mat
```

```
IconData=questIconData;
questIconMap(256,:) = get(handles.figure1, 'Color');
IconCMap=questIconMap;

Img=image(IconData, 'Parent', handles.axes1);
set(handles.figure1, 'Colormap', IconCMap);

set(handles.axes1, ...
    'Visible', 'off', ...
    'YDir'   , 'reverse'       , ...
    'XLim'   , get(Img,'XData'), ...
    'YLim'   , get(Img,'YData')  ...
    );

% Make the GUI modal
set(handles.figure1,'WindowStyle','modal')

% UIWAIT makes Exit wait for user response (see UIRESUME)
uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Exit_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% The figure can be deleted now
delete(handles.figure1);

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.output = get(hObject,'String');

% Update handles structure
guidata(hObject, handles);

% Use UIRESUME instead of delete because the OutputFcn needs
% to get the updated handles structure.
uiresume(handles.figure1);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.output = get(hObject,'String');

% Update handles structure
```

```
guidata(hObject, handles);

% Use UIRESUME instead of delete because the OutputFcn needs
% to get the updated handles structure.
uiresume(handles.figure1);


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if isequal(get(handles.figure1, 'waitstatus'), 'waiting')
    % The GUI is still in UIWAIT, us UIRESUME
    uiresume(handles.figure1);
else
    % The GUI is no longer waiting, just close it
    delete(handles.figure1);
end


% --- Executes on key press over figure1 with no controls selected.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Check for "enter" or "escape"
if isequal(get(hObject,'CurrentKey'),'escape')
    % User said no by hitting escape
    handles.output = 'No';

    % Update handles structure
    guidata(hObject, handles);

    uiresume(handles.figure1);
end

if isequal(get(hObject,'CurrentKey'),'return')
    uiresume(handles.figure1);
end
```