THE UNIVERSITY OF QUEENSLAND

School of Information Technology and Electrical Engineering

Time Series Forecast With Neural Network and Wavelet Techniques

by

Ruey Hwa Loh

Department of Electrical and Computer Engineering,
University of Queensland.

Submitted for the degree of
Bachelor of Engineering (Pass/Honours)
in the division of Electrical Engginering
October 2003

28th October 2003

The Dean
School of Engineering
University of Queensland
St Lucia, Q 4072

Dear Professor Simmons,

In accordance with the requirements of the degree of Bachelor of Engineering (Pass/Honours) in the division of Computer Systems Engineering / Electrical and Electronic Engineering, I present the following thesis entitled “Time Series Forecast with Neural Network and Wavelet Techniques". This work was performed under the supervision of Dr Z.Y. Dong

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,
Ruey Hwa Loh

Acknowledgements

I would like to express my sincere thanks to my thesis supervisor, Dr Z. Y Dong for provided me with the necessary support, advice and enthusiasm required to successfully complete this thesis. His efforts in helping me in the development of the project, through technical difficulties and in search for relevant literature are much appreciated.

Last but least I would like to take this opportunity to show my gratitude for the people who had assisted me in the completion of this thesis.

Abstract

Forecasting of electricity has always been the essential part of an efficient power system planning and operation, especially short term forecasts as it has becoming increasingly important since the rise of the competitive energy markets. The aim of short term load forecast is to predict future electricity demands based on historical data and other information such as temperature. This thesis proposes designing a model using neural network and wavelet techniques to increase the accuracy of time series load forecast.

The model is created in the form of a program package written with MATLAB®. The time series data used are historical electricity load and pricing data of Queensland obtained from the NEMMCO website. Wavelet technique is implemented to the time series data, decomposing the data into number of wavelet coefficient signals. The decomposed signals are then fed into neural network for training. To obtain the predict forecast, the outputs from the neural network are recombined using the same wavelet technique

The simulation results showed that the model was capable of producing a reasonable forecasting accuracy in short term load forecast.

Contents

Chapter 1

Introduction

Chapter 1

Introduction

A reliable and continues supply of electrical energy is necessary for the functioning of today’s complex societies. Because of the combination of increasing consumption and obstruction of various kinds, and the extension of existing electrical transmission networks and these power systems are operated closer and closer to their limits. It is known for all that electrical energy cannot be stored efficiently, due to this fact that the electrical load could be controlled by utilities only to a very small extent, the forecast of load is an important topic for power generation and transmission.

Therefore for these problems, many developed countries opened a deregulated electricity markets in which proper load forecasting is done and people are now able to get reliable electrical energy. In order to plan efficient operation and economical capital expansion of deregulated electrical markets, the market owner should be able to anticipate the need of power delivery, how much power much must be delivered and where and when it will be needed, these information can be provided only by load forecasting, now a days the load forecasting is done by neural networks. Because forecasting of load demand data forms an important component in planning generation schedules in a power system

The aim of short term load forecast is to predict future electricity demands based on historical data and other information such as temperature. The aim of electricity market clearing price forecast is to provide market participants with price signal in the future so to help them maximize their returns by optimized operational planning based on such signals [2]. Proper demand and price forecast can also help to build up cost effective risk management plans for the participating companies in the electricity market.

For this thesis, the electricity market demand and price data are treated as time series, and analyzed to produce forecasted series with the proposed time. Time series forecast is traditionally based on linear models – e.g. Auto-regressive Moving Average (ARMA) models. These models are straight for implementation but fail to give satisfactory results when dealing with nonlinear, non-stationary time series. The solution to that is Neural Network, for their approximation ability for nonlinear mapping and generalization. However, it suffers from the problem of obtaining monolithic global models for a time series. This introduced the multi-resolution decomposition techniques such as wavelet transform approach.

Combining the learning capabilities of neural networks and time series techniques and using wavelet decomposition technique to explore more details of the time series signals. A multi-layer percepton neural network is applied to each decomposed series to predict the time series. We will study neural network models combined with wavelet transformed data, and show how useful information can be captured on various time scales.

Chapter 2

Neural Networks in Time Series Forecast

Chapter 2

Neural Networks in Time Series Forecast

2.1 Introduction

The working principle of neural network is based on the human brain. The human brain, an unparalleled pattern recognition system, consists of 10^{10} computing elements called neurons. They communicate through a network of axons and synapses. 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 1 below [4]:

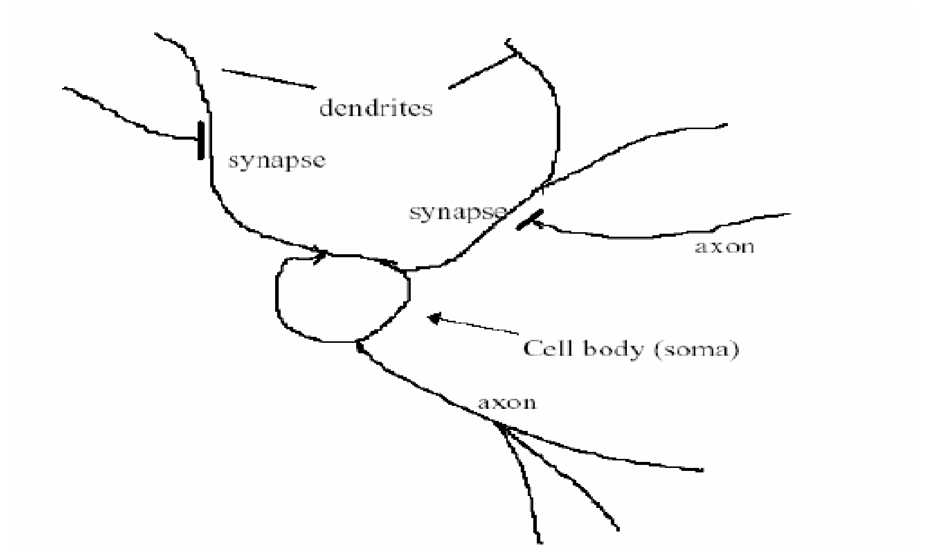

Figure 1: Elements and connectivity of Neural Network

A typical cell has three major regions: the cell body, the axon and the dendrites. The dendrites receive information from neurons through axons that serves as the transmission lines. An axon carries impulse from the neuron. The axon-dendrites connect an organ called synapse. A synapse is situated where the neuron introduces its signal to the neighboring neuron. The neuron responds to the total of its input aggregated within a short time interval known as the period of latent summation. A neuron generates a pulse response and sends it to its axons only if necessary conditions for firing are fulfilled.

The condition for firing is that excitation should exceed inhibition by an amount called threshold of neuron. Since a synaptic connection causes the excitatory or inhibitory reactions of receiving neuron, it is practical to assign positive unity weight values to such connections.

It is observed that the biological network performs temporal integration and summation of incoming signals. The resulting spatial-temporal processing performed by a natural neural network is a complex process and much different from digital computation. The characteristic feature of the neuron is that the signal generated does not differ significantly in the magnitude; the signal in the nerve fiber is either absents or has a maximum value. In other words, information is transmitted between the cells by means of a binary signal.

2.2 Artificial Neural Network

Artificial neural networks are mathematical tools originally inspired by the way human brain processes information [6]. The neuron receives information through a number of inputs nodes. The inputs are multiplied by a weight denoted by W. For instance, the input X_1 is multiplied by a weight of W_1. The same is done for the rest of the inputs as well. Finally a weight vector comprising of all the weights is formed.

The result of all the multiplication of the inputs and weights are then fed to the summer $\sum$ where addition is executed. The output of the summer is then fed to the Linear Threshold unit. If the input to the summer is above the threshold level, an output of '1' will take place. Else, an output of '0' will occur. All the data can be presented to the Network in binary ('1' and '0') or in bipolar ('1' and '-1'). Figure 2 [4] illustrates the diagram of a Linear Threshold unit (LTU).

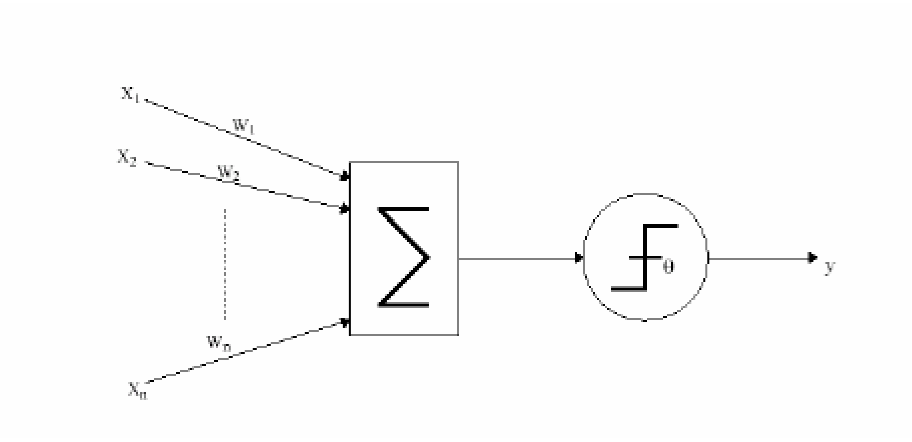

Figure 2: Linear Threshold Unit

2.3 Training of Neural Networks

A neural network is required to go through training before it is actually being applied. Training involves feeding the network with data so that it would be able to learn the knowledge among inputs through its learning rule. There are three types of training algorithms - initialization algorithms, supervised learning and unsupervised learning.

Initialization algorithms are not really training algorithms at all, but methods to initialize weights prior to training proper. They do not require any training data.
In supervised learning, the network is shown a series of input and expected output example. The expected output is used to compare with the actual output from the network.

The network will adjust its weight to accommodate each training examples. The purpose of the weight here is to minimize the difference between the two outputs.

For unsupervised learning, the network is only presented with the inputs but not the output. The weights are updated by the network in response to the input patterns. That implies that there are no training data like supervised learning. For the purpose of this thesis, a variety of supervised learning algorithms will be presented in the following section.

2.3.1 Perceptron Convergence Algorithm

The perceptron was developed by a psychologist known as Rosenblatt in the late 1950s. It was a modification of the LTU which now has the ability to update its weights automatically. The training technique used is called the perceptron convergence algorithm (PCA). 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, when it approaches the desire response, it goes into a saturated form of oscillation around the response rather than terminating itself. Algorithms with such phenomenon are not desired.

2.3.2 Widrow-Hoff Algorithm

The Widrow-Hoff algorithm, also known as the delta rule, was formulated by Widrow and Hoff in 1960. The Widrow-Hoff algorithm was designed to operate with a single neuron and is suitable for solving classification problems that not linearly separable. Although not necessarily give a solution to such a problem it finds a solution close to the problem and terminate when closest to a successful classification unlike PCA. Figure 3 [5] illustrates the diagram of a Widrow-Hoff algorithm.

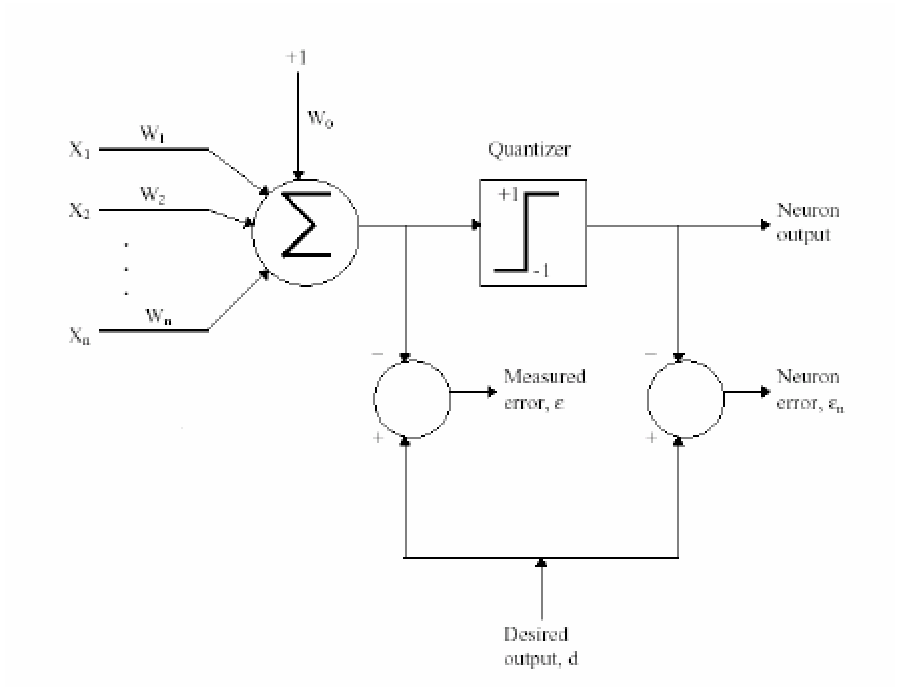

Figure 3: Widow-Hoff algorithm

2.3.3 Back-propagation Algorithm

Though several network architectures and training algorithms are available, the Back-propagation (BP) algorithm is by far the most popular. BP algorithm was created by generalizing the Widrow-Hoff algorithm to multiple-layer perceptron (MPL) and nonlinear differentiable transfer functions [10]. Multilayer perceptrons neural networks trained by BP algorithm consist of several layers of neurons, interconnections, and weights that are assigned to those interconnections. Figure 4 illustrates the diagram of a Multi-layer Perceptron with Two Hidden Layers.

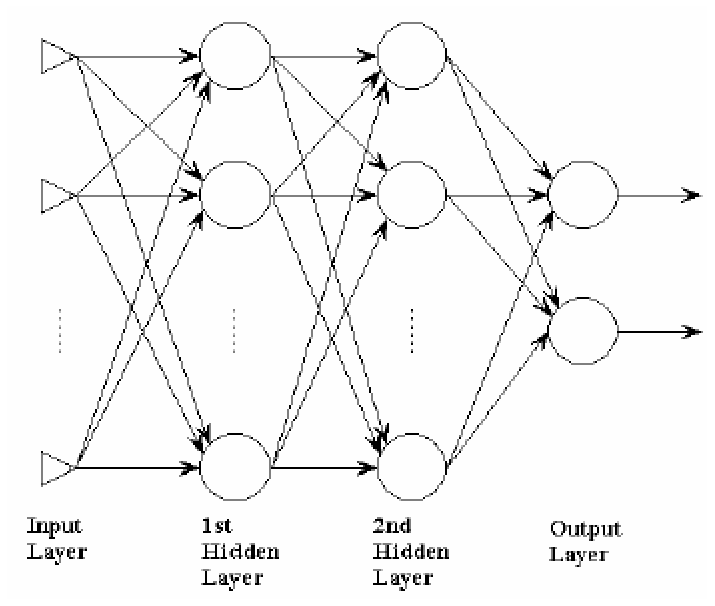

Figure 4: Multi-layer Perceptron with Two Hidden Layers

The algorithm consists of a forward pass and a backward pass. The input signal is first *Forward* through the networks towards the output until a set of the actual response is obtained from the network. From there error signal is then generated based on the difference between the actual and the target response. Lastly the generated error signal is passed *backwards* through the hidden layers, 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

Chapter 3

Wavelet Transform

Chapter 3

Wavelet Transform

3.1 Introduction

Wavelet transform can be viewed as transforming the signal from the time domain to the wavelet domain. This new domain contains more complicated basis functions called wavelets, mother wavelets or analyzing wavelets. The wavelet transform has its roots in the Fourier transform. To have better background information, we will briefly review the Fourier transform and how this transform had led research to develop the Short Time Fourier Transform (STFT) that in turn led to the birth of the wavelet transformation. We will also present an intuitive understanding of the wavelet transform and the usefulness of this transform in time series.

3.2 Fourier Transform

Fourier transform (FT) is probably the most popular mathematical transformation that is being used to extract useful information that is hidden within a raw time series signal (time series signal in its original form) [12]. It can be computed using the following equation:

$$F(\omega) = \int_{-\infty}^{\infty} e^{-j\omega t} f(t)\, dt$$

Eq 3.1

However the FT only gives what frequency components (spectral components) exist in the signal but no information regarding where in time those spectral components appear. Therefore, FT is not a suitable technique for non-stationary signal, time-varying phenomena whose frequency content changes with time.

3.3 Short-Time Fourier Transform

To overcome the limitation of the FT, a window-version of Fourier Transform Known as Short Time Fourier Transform (STFT) was developed. In STFT, the signal is divided into small segments, where these segments (portions) of the signal can assume to be stationary [13]. The expression for a STFT signal can be shown in the following equation:

$$STFT(t, \omega) = \int_{-\infty}^{\infty} e^{-j\omega s} f(s)\, g(s-t)\, ds \qquad \text{Eq 3.2}$$

But STFT face a time–frequency resolution problem due to the width of the window function that is used. Namely, if we use a narrow window, we get good time resolution but poor frequency resolution. Similarly for a wide window, we get good frequency resolution but poor time resolution. The solution to that is to use variable window size which leads to the implementation of wavelet transform.

3.4 Wavelet Transform

The wavelet transform (WT) use a scalable windowing technique. The adjustable window size allows us to trade off the time and frequency resolution in different ways. If we require analyzing a large region of low frequency signal, we will use a long time intervals. Likewise, if we require analyzing a small region of high frequency signal, we will use short time intervals.

Wavelet analysis consists of breaking up the signals into shifted scaled versions of the original wavelet as compared to Fourier analysis, where signals are broken into sine waves of various frequencies instead. And it uses a time-scale region instead of a time-frequency region.

3.4.1 Continuous Wavelet Transform

Continuous wavelet transform (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function [10]. The result of the CWT is the continuous set of wavelet coefficients. When the wavelet coefficients are multiplied with the scaled and shifted wavelet, the constituent wavelets of the original signal are produced. The expression for a CWT signal can be shown in the following equation:

$$\gamma(s,\tau) = \int f(t)\psi^{*}_{s,\tau}(t)dt$$

Eq 3.3

The function, $\psi(t)$, is called the mother wavelet. This function serves as a prototype for generating other window functions. The term translation, τ , refers to the location of the window. As the window shifts through the signal, the time information in the transform domain is obtained. The term scaling, *s*, refers to dilating or compressing the wavelet.

In CWT, an analyzing window is shifted along the time domain to pick upinformation about the signal. However, this process is difficult to implement and the information that has been picked up may overlap and result in redundancy

3.4.2 Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT) involves choosing scales and positions based on powers of two - so called dyadic scales and translation. The mother wavelet is rescaled or “dilated”, by powers of two and translated by integers. The results are more efficient and are just as accurate [10]. 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. The DWT with respect to a mother wavelet, $\Psi(t)$, is defined as:

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

Eq 3.4

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

The DWT has many advantages in compressing a wide range of signals. With DWT technique, a very large proportion of the coefficients of the transform can be set to zero without appreciable loss of information. Be side that if more properties other than the stationary properties of a signal are desired, DWT would certainly be a better choice as compared with the traditional technique of FT [17].

One problem with the application of DWT in time-series analysis is that it suffers from a lack of translation invariance.

3.4.3 Non-Decimated Wavelet Transform

The problem to DWT can be tackled by non-decimated wavelet transform (NWT). NWT uses a redundant basis transformation based on an *n*-length input time series has an *n*-length resolution scale for each of the resolution levels of interest. Therefore, information at each resolution scale is directly related at each time point [3].

If we consider a given time series signal *c*, the à trous wavelet transform is performed by passing the signal through a series of low pass filters *h*. 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*, obtained at each of the different resolution levels. The mathematical expression that describes this process is given as:

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

Eq 3.5

Together with the approximation (low frequency information) coefficient series, *ătrous* wavelet transform also generates the wavelet (high frequency information) coefficient series. As shown in below:

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

Eq 3.6

Finally, the signal can be reconstructed using the mathematical expression given:

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

Eq 3.7

Chapter 4

Forecast Model

Chapter 4

Forecast Model

4.1 Design Overview

The data used in the model to train neural networks are historical electricity load and pricing data. As for the data process tool Wavelet Technique is used in the model.

The model consist of three stages

Stage 1: Data Pre-processing

Stage 2: Data Prediction

Stage 3: Data Post-processing

The reason for addition filter is to further smoother the signal for more forecast accuracy. Figure 5 illustrates the diagram of the forecast model

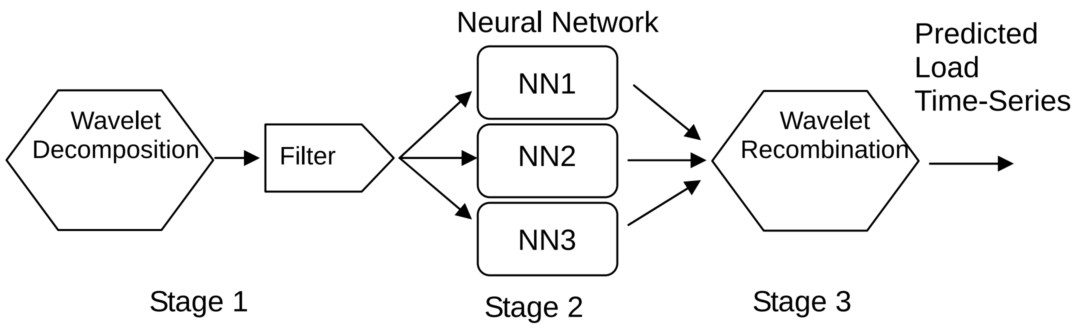

Figure 5: Forecast Model

4.1.1 Stage 1: Data Pre-processing

The data used in the pre-processing are historical electricity load and pricing data. They are fed to model as time-series signals. And Non-decimated Wavelet Transform

(NWT) is used as the data pre processor. Depending on the selected resolution levels, the time-series signals are decomposed into a number of wavelet coefficients. If the resolution level is defined as *n*, after decomposing the signal, there will be one approximation coefficient series with *n* number of detail coefficient series. Figure 6 illustrates the diagram of the forecast model

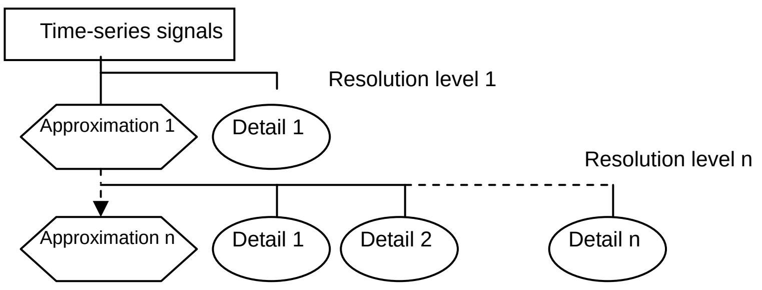

Figure 6: Wavelet Decomposition Process

From here, the wavelet coefficient with the most irregularities will be filter using MATLAB® software Butterworth digital and analog filter design before they are fed as input to neural networks for training or forecasting use.

4.1.2 Stage 2: Data Prediction

Neural networks are used for data prediction in the forecast model. The number of neural networks 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, one neural network is required to perform the corresponding prediction.

4.1.3 Stage 3: Data Post- Processing

For data post- processing, we use the same wavelet technique and resolution level as data pre-processing. The outputs from the neural networks are recombined to form the final predicted output.

4.2 Training Algorithm

As discuss in Chapter 2, 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. Since the standard BP algorithm is a gradient descent algorithm. The *SCG* algorithm is found to be suitable. This is due to its advantage of fast computational time for a large neural networks size. Using this algorithm will significantly shorten the time needed to train the neural networks.

4.3 Number of Input Nodes and Output Neurons

As mentioned earlier, the training time-series data consist of price and electricity load demand of Queensland. Therefore, the total number of input nodes for the proposed model is fixed at two. The forecast model is set to have one output neuron that takes one value of the time-series signal as the target.

4.4 Number of Hidden Neurons

The learning capability of neural networks depends on the number of neurons in its hidden layer. If the number of hidden neuron is too few, the model will not be flexible enough to model the data well. On the other hand if there are too many, the model will overfit the data. By selecting a few alternatives numbers for testing, the most suitable number of neurons for the neural network is explored.

Chapter 5

Results

Chapter 5

Results

5.1 Methodology

The proposed model is tested with two sets of historical data containing the electricity load and price data of Queensland for the month of March 2002, on a half-hourly basis. The sets of electricity load and price data is downloaded from the NEMMCO website.

Even though price data is included with the electricity load data in this forecast model, the model is meant to forecast only the electricity load. As a result, the final predicted series is formed by setting the training target of the neural networks to be a later load series. Price is only used with load to form a particular time-series pattern as the inputs for training the neural networks or for load predictions.

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. For the proposed model, different resolution level are tested and found that 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. Price wavelet coefficient (detail 1) with the most irregularities is put through the Butterworth filter to further smoother the signal for more forecast accuracy.

The proposed model is tested with a number of different values of hidden neurons, however no significant changes are observed with the predicted results. Therefore for this thesis, the number of hidden neurons is set to two

5.2 Simulation Result

Queenlands demand historical time-series data are decomposed at resolution level 2 as follows:

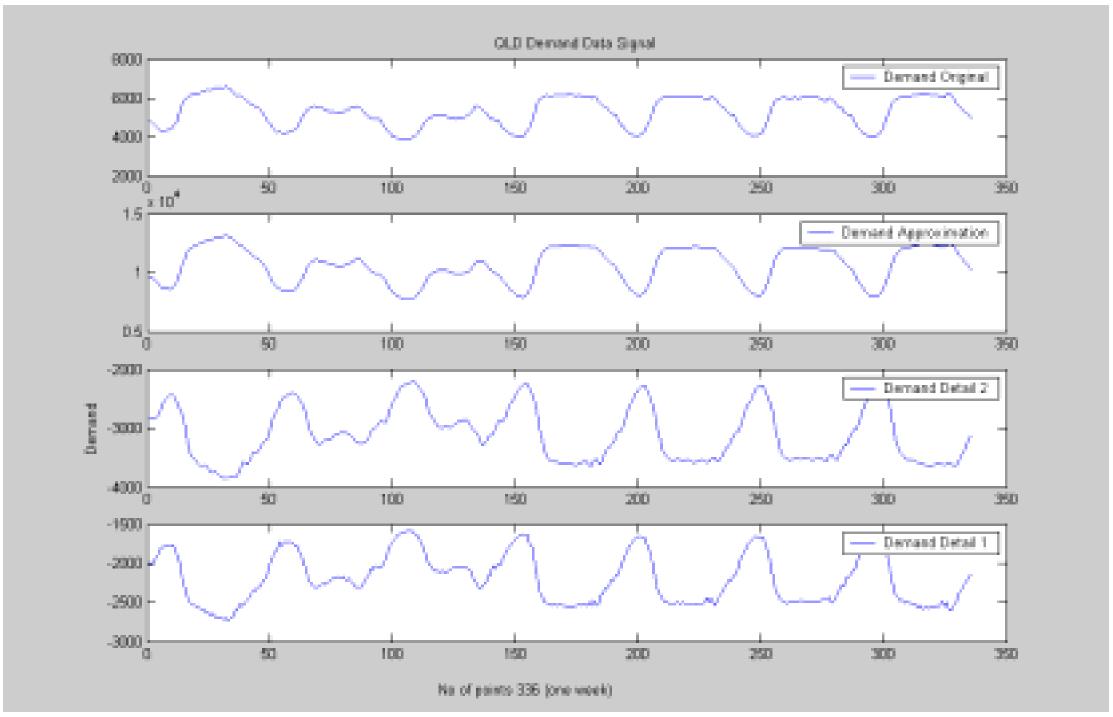

Figure 7: Decomposition of Queensland Demand Series

Queenlands price historical time-series data are decomposed at resolution level 2 with the filter applied to the Detail 1 as follows:

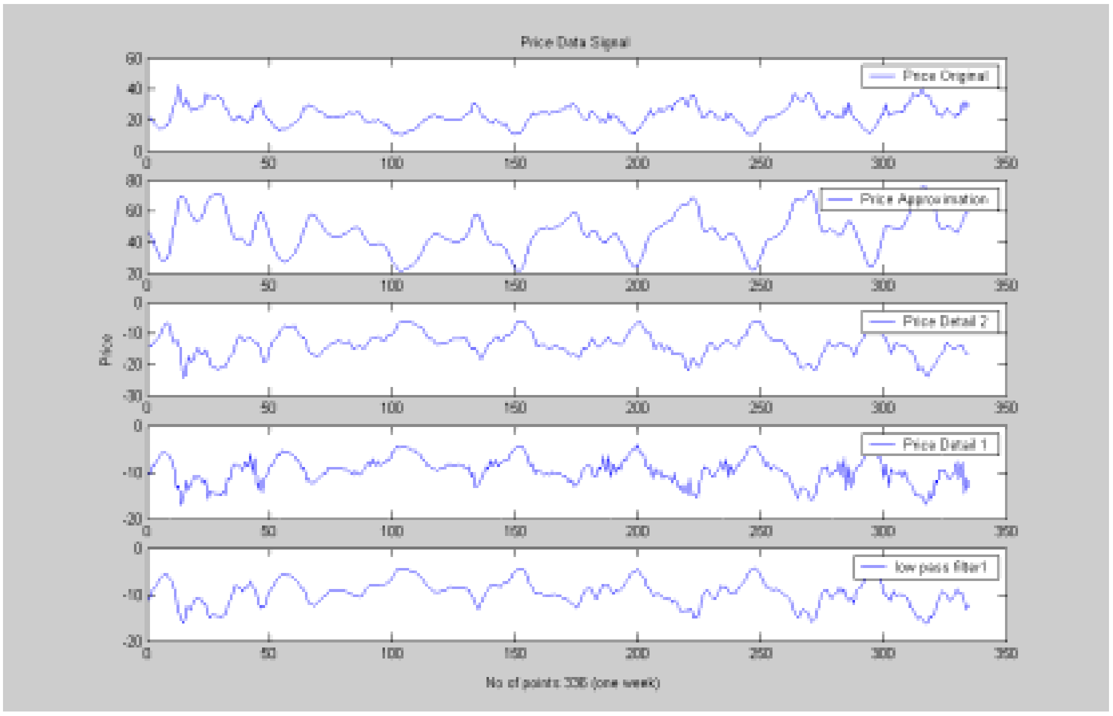

Figure 8: Decomposition of Queensland Price Series

As depicted in forecast model, three neural networks were created for the forecast model - one for the approximation series and one for each of the two wavelet coefficient series. The neural networks were trained with one week (336 points) of combined load and price data for fifty cycles. The performance of the forecast model was evaluated and the results are as follows:

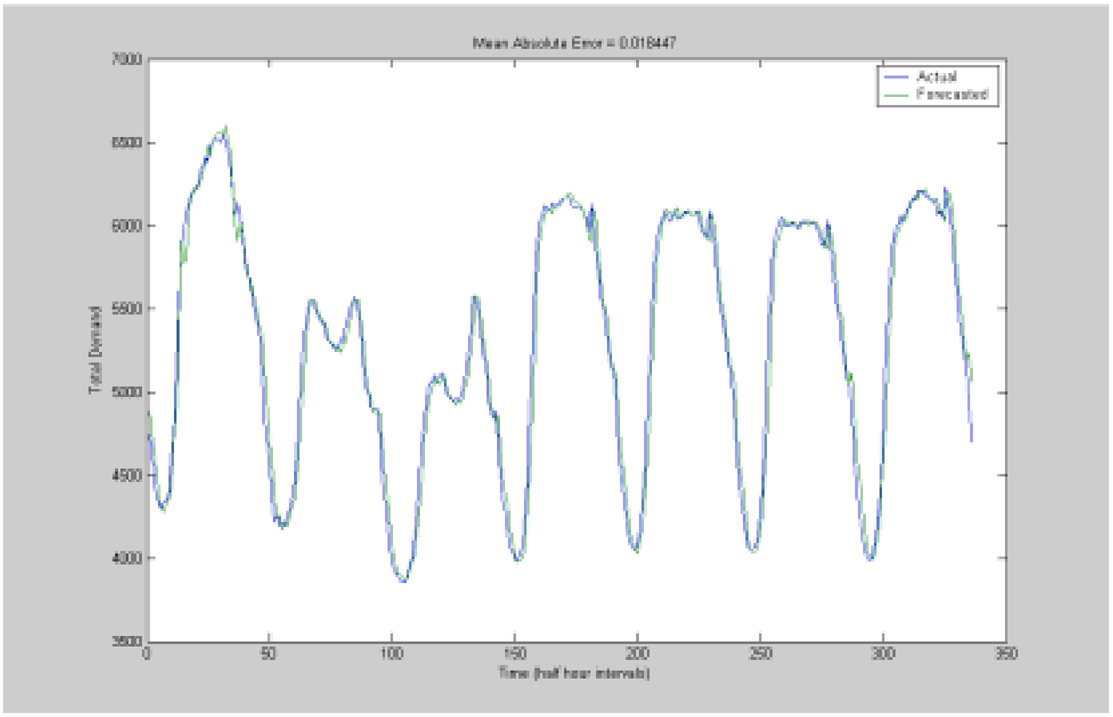

Figure 9: Load Forecasted for 7 days (336 points)

The model is evaluated based on it prediction errors. A successful model would give an accurate time-series forecast. The performance of the model is hence measured using the mean absolute percentage error (MAPE) which is defined as:

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

Eq 5.1

where N is the number of points measured, x is the actual values and y is the predicted values.

5.3 Discussions

The simulation results showed that the model was capable of producing a reasonable accuracy in short team load forecast. The proposed model requires only 336 points (one week) of the pricing and electricity historical data in every training cycle. Once the model is trained, it can be used to forecast the electricity load data for 48 times, each time forecasting one point ahead, without the need to be retrained.

Chapter 6

Conclusions and Recommendations

Chapter 6

Conclusion and Recommendations

6.1 Conclusion

With the implementation of the wavelet decomposition into the time series and neural network, its help to gain deeper insight into the demand data series and help to get better prediction result. The addition filtering of price wavelet coefficient (detail 1) is done to further smoother the signal for more forecast accuracy.

From the simulation results the propose model is capable of producing a reasonable accuracy in short team load forecast. However, the model is found to be able to forecast only one point ahead. To overcome this limitation the model need to be modified to predict more points ahead.

To sum up, the inclusion of price 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.

6.2 Recommendations for Future Work

In this thesis, even though the price data is included with the electricity load data. The model is only capable to predict the electricity load but not the electricity prices. Thus, in order to predict future electricity price. One approach may be using statistical method based on time-series analysis. The statistical methods employ mathematical models to identify the parameters to fit a predefined formula based on the historical data. The statistical method is only a suggestion; other types of method (e.g. simulation based methods) should also be explored in detail for future improvements.

References

[1] National Electricity Market Management Company Limited (NEMMCO) website, http://www.nemmco.com.au/publications/whitebook/market.htm (current mar. 2003)

[2] D. Sansom and T. K. Saha, “Neural networks for forecasting electricity pool price in a deregulated electricity supply industry”, *Proceedings of the 1999 Australasian Universities Power Engineering Conference*, Darwin, Australia, Sep. 1999, pp. 214-219.

[3] B. L. Zhang and Z. Y. Dong, “An Adaptive Neural-Wavelet Model for Short Term Load Forecasting,” *Electric Power Systems Research* 59 (2001) 121-129

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

[5] T. Down, “Introduction to Neural Computing & Pattern Recognition,” *COMP3700* http://www.csee.uq.edu.au/~comp3700/lectures/lecture2.pdf (current Jul. 2003)

[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. Bharath and J. Drosen, “Neural Network Computing,” Windcrest, USA, 1994

[8] H. Maier, “A Review of Artificial Neural Networks,” Department of Civil and Environmental Engineering, University of Adelaide, Research Report No. R 131, Aug. 1995

[9] S. Haykin, “Neural Networks,” Macmillan College Publishing Company, Inc. 1994

[10] The MathWorks Inc., Matlab, Version 6.5

[11] A. Graps, "An Introduction to Wavelets"**,** *IEEE Computational Sciences and Engineering*, Vol. 2, No. 2, Summer 1995, pp 50-61.

[12] R. Polikar, " The Wavelet Tutorial", http://engineering.rowan.edu/~polikar/WAVELETS/WTpart1.html, (current Aug 2003)

[13] R. Polikar, " The Wavelet Tutorial", http://engineering.rowan.edu/~polikar/WAVELETS/WTpart2.html, (current Aug 2003)

[14] C.S. Burrus (Contributor), R.A. Gopinath and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer*, Prentice Hall, New Jersey, 1998

[15] C. Valens, "A Really Friendly guide to Wavelets", http://perso.wanadoo.fr/polyvalens/clemens/wavelets/wavelets.html, (current Aug 2003)

[16] Randy K. Young, "Wavelet Theory And Its Applications", Kluwer Academic Publishers, 1993

[17] B. N. Tran et al, "Wavelets," *J. Webster (ed.) Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, Inc. 1999

[18] G. Zhang et al, "The Wavelet Transform for Filtering Financial Data Streams," *Journal of Computational Intelligence in Finance*, Vol. 7, No. 3, May/Jun. 1999

[19] A. Aussem et al, "Wavelet-Based Feature Extraction and Decomposition Strategies for Financial Forecasting," *Journal of Computational Intelligence in Finance*, Mar./Apr. 1998

[20] 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-

[21] D. W. Bunn, "Forecasting loads and prices in competitive power markets," *Proceedings of the IEEE*, Vol. 88, No. 2, Feb. 2000, pp. 163-169

[22] W. R. Foster et al, "Neural network forecasting of short, noisy time series," *Computers Chem. Engng.*, Vol. 16, No. 4, 1992, pp. 293-297

[23] I. Nabney, "Netlab," *Netlab Neural Network Software*, http://www.ncrg.aston.ac.uk/netlab/ (current Aug 2003)

Appendix - Source Code Listing

```
%File name :      nprogram.m
%Description : This file reads the data from its source into their respective matrices prior to
               performing wavelet decomposition.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clear command screen and variables
clc;
clear;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% user desired resolution level (Tested: resolution = 2 is best)
level = menu('Enter desired resolution level: ', '1',...
             '2 (Select this for testing)', '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);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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 (Select this for testing)');
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 weeks = 336 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: ', 'Jan02',...
              'Feb02', 'Mar02 (Select this for testing)', 'Apr02', 'May02');
switch select
    case 1, demandFile = 'DATA200201_QLD1';
    case 2, demandFile = 'DATA200202_QLD1';

    case 3, demandFile = 'DATA200203_QLD1';

    case 4, demandFile = 'DATA200204_QLD1';

    case 5, demandFile = 'DATA200205_QLD1';
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Reading the historical DEMAND data into tDemandArray
selectedDemandFile = [demandFile, '.csv'];
[regionArray, sDateArray, tDemandArray, rrpArray, pTypeArray] ...
= textread(selectedDemandFile, '%s %q %f %f %s', '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] = swtmat(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

        if i == 1
            subplot(resolution + 2, 1, resolution - i + 3); plot(detail(i, 1: dataPoints))
            legendName = ['Demand Detail ', num2str(i)];
            legend(legendName);
            else
                  subplot(resolution + 2, 1, resolution - i + 3); plot(detail(i, 1: dataPoints))
                  legendName = ['Demand Detail ', num2str(i)];
                  legend(legendName);

            end

     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 historical PRICE data into rrpArray
selectedPriceFile = [demandFile, '.csv'];
[regionArray, sDateArray, tDemandArray, rrpArray, pTypeArray] ...
= textread(selectedDemandFile, '%s %q %f %f %s', 'headerlines', 1, 'delimiter', ',');


%Display no. of points in Price data
[noOfDataPoints, y] = size(rrpArray);
msg = ['The no. of points in Price data data is ', num2str(noOfDataPoints)];
disp(msg);


%Decompose historical Price data signal
[dP, l] = swtmat(rrpArray, resolution, 'db2');
approxP = dP(resolution, :);


%Plot the original Price data signal
figure (2);
subplot(resolution + 3, 1, 1); plot(rrpArray(2: dataPoints))
legend('Price Original');
title('Price Data Signal');


%Plot the approximation Price data signal
for i = 1 : resolution
       subplot(resolution + 3, 1, i + 1); plot(approxP(2: dataPoints))
       legend('Price Approximation');
```

```
end


%After displaying approximation signal, display detail x
for i = 1: resolution
   if( i > 1 )
         detailP(i, :) = dP(i-1, :)- dP(i, :);
    else
         detailP(i, :) = rrpArray' - dP(1, :);
   end

if i == 1
    [B,A]=butter(1,0.65,'low');
    result =filter(B,A, detailP(i, 1: dataPoints));

    subplot(resolution + 3, 1, resolution - i + 4);plot(result(i, 2: dataPoints))
    legendName = ['low pass filter', num2str(i)];
    legend(legendName);

    subplot(resolution + 3, 1, resolution - i + 3); plot(detailP(i, 2: dataPoints))
    legendName = ['Price Detail ', num2str(i)];
    legend(legendName);

else
    subplot(resolution + 3, 1, resolution - i + 3); plot(detailP(i, 2: dataPoints))
    legendName = ['Price Detail ', num2str(i)];
    legend(legendName);

end
   i = i + 1;
end


%Normalising approximation Price data
maxPrice = max(approxP'); %Find largest component
normPrice = approxP ./ maxPrice; %Right divison

maxPriceDetail = [ ];
normPriceDetail = [, ];

detailP = detailP + 40;

for i = 1: resolution
    maxPriceDetail(i) = max(detailP(i, :));
    normPriceDetail(i, :) = detailP(i, :) ./maxPriceDetail(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function here allows repetitive options to,
%     1) Create new NNs, 2) Retrain the existing NNs,
%     3) Perform load demand forecasting and 4) Quit session

while (1)

    choice = menu('Please select one of the following options: ',...
                            'CREATE new Neural Networks',...
                            'Perform FORECASTING of load demand', 'QUIT session...');
    switch choice
        case 1, scheme = '1';
        case 2, scheme = '2';
        case 3, scheme = '3';
        case 4, scheme = '4';
end


    %If scheme is 'c', call <nCreate> to create new NNs, train them then perform forecast
    if(scheme == '1')
        nCreate;
    end

    %If scheme is 'r', call <nRetrain> to retrain the existing NNs
    if(scheme == '2')
        nRetrain;
    end

    %If scheme is 'f', call <nForecast> to load the existing NN model
    if(scheme == '3')
        nForecast;
    end


    %If scheme is 'e', verifies and quit session if 'yes' is selected else continue
    if(scheme == '4')
        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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%File name :   ncreate.m
%Description : This file prepares the input & output data for the NNs. It creates then trains the
               NNs to mature them.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%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 2 sets of training examples; ');
disp('1 set of demand + 1 sets of price. ');
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...');
pause;


%Clear command screen then prompt for no. of training cycles
%For current program, 1 cycle = user set no. of iterations (ie: dataPoints)
clc;
cycle = input('Input the number of training cycles: ');


numOfTimes = resolution + 1;
%Creating and training the NNs for the respective
%demand and price coefficient signals
for x = 1: numOfTimes

    %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 price coefficient signal
    numOfInputs = 2;
    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 = input('Enter the no. of HIDDEN units for the NN hidden : ');
    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 2 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) = normPrice(y + 1);

      else
            inputs(1, y + 1) = normDemandDetail(x - 1, y + 1);
            inputs(2, y + 1) = normPriceDetail(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)


clear nn;


%Create new neural network for respective coefficient signal
%NET = MLP(NIN, NHIDDEN, NOUT, FUNC)
nn = mlp(numOfInputs, numOfHiddens, numOfOutput, 'linear');


%NN options
options = zeros(1, 18);
options(1) = 1; %Provides display of error values
options(14) = cycle * ni * np;


%Training the neural network
```

```
    %netopt(net, options, x, t, alg);
    nn = netopt(nn, options, inputs, output, 'scg');

    %Save the neural network
    filename = ['nn', num2str(x)];
    save(filename, 'nn');

    disp(' ');
    msg = ['Neural network successfully CREATED and saved as => ', filename];
    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!!');
        disp(' ');
        disp('Press ENTER key to continue...');
    end

    pause;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




%File name :   nforecast.m
%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 = ['nn', num2str(x)];
    clear nn
    load(filename);
    clear in;
    numOfInputs = nn.nin;
    y = 0;
    %Preparing details to forecast
    while y < pointsAhead
        if(x == 1)
            propData(1, y + 1) = normDemand(y + 1);
            propData(2, y + 1) = normPrice(y + 1);

        else
            propData(1, y + 1) = normDemandDetail(x - 1, y + 1);
            propData(2, y + 1) = normPriceDetail(x - 1, y + 1);

        end

        y = y + 1;
    end

    if(x == 1)
        forecast = mlpfwd(nn, propData') * maxDemand;
    else
        forecast = mlpfwd(nn, propData') * maxDemandDetail(x - 1) - 4000;
    end
    forecastResult(x, :) = forecast';
end

%For debugging
% forecastResult

actualDemand = tDemandArray(2: 1 + pointsAhead);
predicted = sum(forecastResult, 1)';


%Calculating Mean Absolute Error
AbsError = abs(actualDemand - predicted(1: pointsAhead)) ./ actualDemand;
msg = ['Mean Absolute Error = ', num2str(mean(AbsError(1: pointsAhead))), ' !!'];
disp(' ');
disp(msg);


%Plot actual time series against predicted result
figure(3)
actualWithPredicted(:, 1) = actualDemand;
actualWithPredicted(:, 2) = predicted(1: pointsAhead);
plot(actualWithPredicted);
graph = ['Mean Absolute Error = ', num2str(mean(AbsError))];
title(graph);
legend('Actual', 'Forecasted');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%File name :    nretrain.m
%Description : This file loads the existing NNs and trains them again.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
%Prompt for the starting point for training
disp('Program will now RETRAIN the Neural Networks ')
disp('with the SAME intial data series again...');
disp(' ');
disp('To capture the pattern of the signal, the model is ')
disp('set to accept dataPoints x 2 sets of training examples; ');
disp('1 set of demand + 1 sets of price. ');
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: ');

numOfTimes = resolution + 1;
%Loading existing NNs for training
for x = 1: numOfTimes

    %Re-initialising variables
    clear targetDemand;
    clear inputs;
    clear output;
    clc;

    %Loading NN for the respective demand and temperature coefficient signals
    filename = ['nn', num2str(x)];
    clear nn
    load(filename);

    %Getting the size of NN
    numOfInputs = nn.nin;
    numOfHiddens = nn.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 2 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) = normPrice(y + 1);

      else
            inputs(1, y + 1) = normDemandDetail(x - 1, y + 1);
            inputs(2, y + 1) = normPriceDetail(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) = cycle * ni * np;


%Training the neural network
%netopt(net, options, x, t, alg);
nn = netopt(nn, options, inputs, output, 'scg');


%Save the neural network
filename = ['nn', num2str(x)];
save(filename, 'nn');

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
```