Multi-Polynomial Higher Order Neural Network Group Models
For Financial Data and Rainfall Data Simulation & Prediction

By
Hui Qi

A Thesis
Presented to the
Department of Computing and Information Systems,
University of Western Sydney, Macarthur
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science (Honours)

March, 2001
Statement of Authentication

The work presented in this thesis is, to the best of my knowledge and belief, original except as acknowledged in the text. I hereby declare that I have not submitted this material, either in whole or in part, for a degree at this or any other institution.

................................
(Signature)

Hui Qi
Acknowledgments

I’d like to thank many people for supporting and helping my research. In particular, I’d like to express my deepest thanks to my supervisor Dr. Ming Zhang for his guidance and support during my research. Although this work was performed while Dr. Ming Zhang held a USA National Research Council Senior Research Associateship at the National Oceanic and Atmospheric Administration (NOAA); National Environmental Satellite, Data, and Information Service (NESDIS); Office of Research and applications (ORA), many e-mail letters were sent to me from him, which all were very significant for this thesis.

I must thank Professor Athula Ginige and Associate Professor San Murugesan for their help in my research career.

I’d also like to thank the following people for concerning with my life and study: Tony Yi, Director of Barclay Company; Baoyan Pang, Tony Yi’s wife; Jingchun Zhang, Programmer of IBM Australia; Charles Hu, PhD in USWM; and Peter Lu, Msc in UWSM.

Finally, I want to thank my wife Li Wang and my son Geoffrey Qi for their love. With their love companying, I overcame many difficulties in the project of my research. The best parents in the world, Sanyun Qi and Fengzhi Chi, continue to encourage and support me in every way.
Abstract

Multi-Polynomial Higher Order Neural Network Group Models (MPHONNG) program developed by the author will be studied in this thesis, as well the thesis investigates especially using MPHONNG for financial data and rainfall data simulation and prediction. The MPHONNG is combined with characteristics of Polynomial function \[ \sum_{i,j=0}^{n} (a_{ij} x^i y^j) \], Trigonometric polynomial function \[ b_{ij} \sin^{i}(b_{i,j}x) \cos^{j}(b_{j,j}y) \] and Sigmoid polynomial function \[ \frac{c_{ij}}{(1 + \exp(-c_{ij}x))^i (1 + \exp(-c_{ij}y))^j} \]. The models are constructed with three layers Multi-Polynomial Higher Order Neural Network. The weights of the MPHONNG models are derived directly from the coefficients of the Polynomial form, Trigonometric polynomial form and Sigmoid polynomial form. The theoretical principles of the Multi-Polynomial Higher Order Neural Network Group Models are presented and demonstrated in the thesis.


To the best knowledge of the author, it is the first attempt to use Multi-Polynomial Higher Order Neural Network Group Models (MPHONNG) for financial data and rainfall data simulation and prediction. So far we could not find any references to using Multi-Polynomial Higher Order Neural Network Group Models for financial data and rainfall data simulation and prediction in the extensive literature search conducted for this thesis, including a thorough Internet search on this topic.

Multi-Polynomial Higher Order Neural Network Group Models (MPHONNG) program runs on X window on Sun workstation. A user-friendly GUI (Graphical User Interface) system has also been incorporated. When you run the system, any step, data or calculation can be reviewed and modified from different windows during processing. Hence, changing data, network models and comparing results
can be done very easily and efficiently.

In the financial data area, sixteen groups of real-life data extracted from Reserve Bank of Australia Bulletin have been used for testing. The average error of the simulation is from 0.2070% up to 3.3512%. In the rainfall data area, MPHONNG program combining with ANSER-PLUS EXPERT SYSTEM for rainfall simulating has been tested by using satellite picture data from NOAA/NESDIS/ORA-USA (USA National Research Council Senior Research Associateship at the National Oceanic and Atmospheric Administration (NOAA), National Environmental Satellite, Data, and Information Service (NESDIS), Office of Research and applications (ORA).). The average error of the simulation is only about 1.86% * and 6.02% #.

All results of simulation and prediction have given satisfactorily, which have been confirmed that the MPHONNG have capable of handling high frequency, high order nonlinear and discontinuous data.
2.2-2 Structure ..................................................................................(13)
2.2-3 Weights ..................................................................................(16)
2.2-4 Learning Techniques .................................................................(17)
2.2-5 Training (Finding the Weights) by way of Back Propagation (BP) ......(21)

Chapter 3. Ordinary Higher Order Neural Networks ...(22)
3.1 Higher Order Neural Network (HONN) ...........................................(22)
3.2 Polynomial Higher Order Neural Networks (PHONN) .................(23)
3.3 Trigonometric Polynomial High Order Neural Network

(THONN) ..........................................................................................(24)
3.4 Polynomial and Trigonometric Polynomial High Order Neural
Network (PT-HONN) ..........................................................................(25)

Chapter 4. The Theoretical Basis of the Multi-Polynomial
Higher Order Neural Network Group Models
Program ............................................................................................(26)
4.1 The aim of introduction ..................................................................(26)
4.2 Artificial Neural Network Sets ......................................................(28)
4.3 Genenalised Artificial Neural Network Set ....................................(31)
4.3-1 Generalised product sets of artificial neural network sets ...........(31)
Chapter 4. Multi-Neural Network Group Models

4.3-2 Additive Generalised Artificial Neural Network Sets ........................................(32)
4.3-3 Generalised ANN Sets ......................................................................................(32)
4.4 Neural Network Group ......................................................................................(34)
4.4-1 Product notation of artificial neural network groups ....................................(34)
4.4-2 Additive notation of artificial neural network groups ....................................(37)
4.4-3 Neural Network Group Set (NNGS) .................................................................(39)
4.5 Neural Network Group Models ........................................................................(41)
4.5-1 Neural Network Algebra Sum Groups .............................................................(41)
4.5-2 Piece-wise Function Groups (PFGs) .................................................................(42)
4.5-3 Neural Network Group Features ....................................................................(43)
4.6 Using MPHONNG Models for Real Data Simulation and Estimation .........................(47)
4.6-1 The Formulation of MPHONNG .................................................................(47)
4.6-2 The Structure of MPHONNG ...........................................................................(47)

Chapter 5. Multi-Polynomial Higher Order Neural Network Group Models (MPHONNG) .................................................................(49)

5.1 The MPHONNG Definition ............................................................................(49)

5.2 The MPHONN Model .....................................................................................(50)
5.2-1 The MPHONN Definition .............................................................................(50)
5.2-2 The Formula of MPHONN Model ...........................................(51)
5.2-3 The Structure of MPHONN Model ...........................................(51)
5.2-4 The Derivatives of MPHONN Model ...........................................(53)

**Chapter 6. MPHONNG Simulator** .............................................(59)

6.1 Overview of the MPHONNG Program ...........................................(59)

6.2 The Interface of MPHONNG simulator and procedures of operating .........................................................(62)

6.3 MPHONNG Software Package ...............................................(80)

**Chapter 7 Experiment of MPHONNG** ....................................(82)

7.1 Simulation-1 .........................................................................(83)

7.2 Simulation-2 .........................................................................(84)

7.3 Simulation-3 .........................................................................(85)

7.4 Simulation-4 .........................................................................(86)

7.5 Simulation-5 .........................................................................(87)

7.6 Simulation-6 .........................................................................(88)

7.7 Simulation-7 .........................................................................(89)

7.8 Simulation-8 .........................................................................(90)
Chapter 8 Comparing MPHONNG with PHONN, THONN and PT-HONN

8.1 Comparison MPHONNG program with PHONN, THONN and PT_HONN

8.1-1 Comparison -1 ........................................(101)
8.1-2 Comparison -2 ........................................(102)
8.1-3 Comparison -3 ........................................(103)
8.1-4 Comparison -4 ........................................(104)
8.1-5 Comparison -5 ........................................(105)
Chapter 9 Using MPHONNG Model for estimating heavy rainfall from satellite data

9.1 Introduction

9.2 ANSER- plus system

9.3 Estimating with MPHONNG and comparison results

Chapter 10 Conclusion

10.1 Original Contribution

10.2 Future Research
Chapter 1. Introduction

In this chapter, I first explain the motivation for this research, and then give an overview the thesis. The major original contributions of the thesis, the abbreviations and the symbols used in the thesis are listed at the end of the chapter.

1.1 Motivation

Until now, in the area of the real world data simulation and prediction, there is no single neural network model that could handle the wide variety of data and perform very well. This is due to the difficulty in dealing with complicated real work data. This problem has become one of the major research topics in the discipline of computing.

In the major motivation for my research, it deals with many financial data and rainfall data, this because these data are representative and practicable. Financial operation system and rainfall simulations-predictions have nowadays access to an extremely large amount of data, both quantitative and qualitative, real-time or historical, and people can use this information for their investment decision-making process and protecting from human’s life.

Firstly, financial data, such as historical price databases or real-time price information is largely processed by computer programs. However, there are only few programs based on artificial intelligence techniques for financial analysis intended for the end-user. Secondly, let us consider the human need for more accurate weather forecast so as to prevent natural disasters. Currently, satellite derived precipitation estimates [1, 4, 119] and 3-hour precipitation outlooks for convective systems, extra-tropical cyclones and tropical cyclones are computed on the NOAA/NESDIS Interactive Flash Flood Analyser (IFFA) system and transmitted to National Weather Service Forecast Offices, and River Forecast Centres in USA. However, this system permits the computation of rainfall estimates for only one convective system and it is very time consuming. This is due to the considerable time needed for image processing, interpretation and the computation involved in the estimation of rainfall. If there were several storms occurring, an automatic estimation technique would be useful in providing rainfall estimates for the entire country. But, the classic methods cannot give good results.

Artificial Intelligent (AI) is another possible way to solve the problems. In financial field, the traditional way of operation is the Questions and Answers (Q&A) method. The neural network model looks like a ‘Black Box’ for the
financial operators. Within Q&A method, financial operators do not need to know much about the underlying model for the problems, because the network would be able to deduce this model without outside intervention, given the relevant training data. This kind of process is called ‘Model-Free Inference’. For situations where it is too difficult or time consuming to derive an accurate mathematical representation for the physical model, such a system would be ideal in practice. In rainfall estimation, some artificial intelligence system for weather forecasting are designed to be objective and automated, some are designed to augment human skill. In knowledge Augmented Severe Storms Predictor (KASSPr), knowledge was elicited in a series of interviews and exchanges of documentation between the developer and an expert in severe weather forecasting. But so far no AI system can solve these problems very well.

Artificial Neural Network (ANN) computing is an area that is receiving increased research interest. Since Richard Lippmann’s [121] tutorial article “An Introduction to Computing with Neural Networks”, Lippmann’s [121] article becomes one of the most widely referenced papers in the neural network literature. In Lippmann’s article, Multi-Layer Perceptron (MLP) neural network has been introduced. So far, the MLP is still the most widely used neural network in the world. But no one had very good results for finance data prediction and rainfall estimation only using simple neural network.

Artificial neuron network-based models are not yet sufficiently powerful to characterize complex systems. Moreover, a gap exists in the research literature between complex systems and general systems. To characterize complex systems, neuron-based and neural network-based group models are studied. Lumer [130] proposed a new mechanism of selective attention among perceptual; groups as part of his computational model of early vision. Lie Groups were used in Tsao’s [122] group theory approach to the computer simulation of 3D rigid motion.

A way of solving the problems is to develop a number of new models with different algorithms. A wider variety of models would give people more chances to find a suitable model and more accurate results when they process their data in financial decision and rainfall estimation. This is the major motivation for my research.

The degree of accuracy is the most important characteristic of a simulation and prediction model. A way to increase the degree of accuracy of a model is provided in this thesis. Group theory with Multi-Polynomial Higher Order Neural Network Models (MPHONNG) is used in the thesis to improve accuracy.

The difficulty is due to the dual nature of the estimation of error in a problem. An incorrect model that has insufficient or inappropriate representational ability will have a high bias. On the other hand, a model able to be truly bias free must have
a high variance to ensure its encoding flexibility, and hence will require a prohibitively large training set to provide a good approximation.

The dilemma is that, the more representational power a neural network model is given, the more difficult it is for it to learn concepts correctly. Each neural network model has an inherent underlying process that is used to construct its internal model and as a consequence, any solutions that are found will be naturally biased by the representational power of the learning system. Such bias includes the architecture type, connection topology and perhaps the input and output representations. Consequently the estimation of these parameters rely on the prior knowledge or biases of the researcher about the problem, annihilating the original goal of bias free learning.

To achieve low variance while simultaneously estimating a large variety of parameters require an impractical number of training examples.

One possible solution to this problem is to develop a new model that is visible to the users. The author’s MPHONNG program allows the users to watch every aspect of the model during the training process. We call this kind of model training as an ‘Open-Vision Box’.

1.2 Original Contributions

The original contributions of the thesis in the field of the neural networks for financial data simulation and rainfall estimation are:

1. On neural networks:

   a) Using Multi-Polynomial Higher Order Neural Network Group Models, which have been developed and theoretically analysed.

   b) The thesis is the first attempt to use Multi-Polynomial Higher Neural Network Group Models for finance data simulation and prediction, and rainfall estimation and prediction.

   c) Use of group theory in the Multi-Polynomial Higher Order Neural Network Group Models.

   d) Combining MPHONNG with ANSER-PLUS EXPERT SYSTEM for rainfall estimation and prediction, making results visible directly.

2. On system architecture:

   a) A complete new computer program has been developed. It is called MPHONNG, for applying Multi-Polynomial Higher Order Neural Network Group Models for financial data simulation and prediction, and rainfall
estimation.

b) The program has been developed in a user-friendly GUI system, easily controlled with drop-down menus and the mouse. Both system operation and system mode can be viewed during the process of data simulation and estimation.

3. On testing results:

a) In the financial data area, MPHONNG program has been tested by using real life data extracted from Reserve Bank of Australia Bulletin. The average error of the simulation is from 0.2070% up to 3.3512%

b) In the rainfall data area, MPHONNG program Combining with ANSER-PLUS EXPERT SYSTEM for rainfall has been tested by using satellite picture data from NOAA/NESDIS/SAL-USA. The average error of the simulation is only about 1.86% * and 6.02% #.

1.3 Abbreviations

ANN  Artificial Neural Networks.
MLP  Multi-layer Perceptrons.
HONN Higher Order Neural Networks.
PHONN Polynomial Higher Order Neural Networks.
THONN Trigonometric Polynomial Higher Order Neural Networks.
PT-HONN Polynomial and Trigonometric Polynomial Higher Order Neural Networks.
M-PHONN Multi-Polynomial Higher Order Neural Networks.
MPHONNG Multi-Polynomial Higher Order Neural Network Group Models.
GUI  Graphical User Interface.
BPN  Back Propagation Network
SPHONN Sigmoid Polynomial Higher Order Neural Network
1.4 Symbols

\( A \) \quad a \text{ set}

\( a \) \quad an \text{ element of set } A \n
\( \eta \) \quad \text{learning-rate parameter}

\( j \) \quad \text{input index } j=1, \ldots, L \text{ means } L \text{ input for output neuron}

\( k \) \quad \text{the } k^{th} \text{ output neuron}

\( E \) \quad \text{error}

\( t \) \quad \text{training time}

\( o \) \quad \text{output layer}

\( p \) \quad \text{the } p^{th} \text{ training vector}

\( w_{kj}^{\theta} \) \quad \text{weight}

\( i_{pj} \) \quad \text{input to the output neuron (output of second hidden layer)}

\( f_k^{\theta} \) \quad \text{activity function of output neuron}

\( o_{pk} \) \quad \text{output of output neuron}

\( \theta_k^{\theta} \) \quad \text{bias term}

\( y_{pk} \) \quad \text{desired output value,}\n
\( o_{pk} \) \quad \text{actual output from the } k^{th} \text{ unit}
1.5 Overview of the thesis

The essence of this thesis is presented in the three papers presented by the author at three international conferences, and include as Appendix A. The author has successfully developed a new neural network group model, which is named the Multi-Polynomial Higher Order Neural Network Models (MPHONNG). The program for the financial data simulation and rainfall estimation is based on the MPHONNG.

Chapter 2 starts by providing a general review of knowledge acquisition. The review is based on an extensive literature search and on an experimental study of representation in machine learning. Prior knowledge and its relationship to a posteriori knowledge are introduced, followed by a summary of previous attempts at incorporation prior knowledge in neural systems.

Statistical inference, a general framework for the analysis of this knowledge in machine learning systems, is then discussed. This provides a basis for many of the terms used in this thesis. The more specific case of feed-forward neural networks is then examined with regard to the density of solutions in the weight space.

Chapter 3 presents some ordinary higher order neural networks models, which are basis of my research. During my studying, I used these models for comparing with my new model MPHONNG. They are HONN, PHONN, THONN and PT_HONN.

Chapter 4 presents the theoretical basis of the models that I have developed. The key idea in this chapter is to use group theory to study the structure of neural network group models. The reasons are given for using neural network group models for setting up my new model MPHONNG. The definition of a neural network group is introduced. The neural network group models and features are then discussed in some detail.

Chapter 5 presents the new model which I developed the Multi-Polynomial Higher Order Neural Network Group Models. It includes the MPHONNG definition, the formula of MPHONN, the structure of MPHONN and the derivatives of MPHONN.

Chapter 6 introduces the interface of the MPHONNG, which includes procedures for using this program and its package.
Chapter 7 provides sixteen sample applications of the MPHONNG program for experiment of data simulation.

Chapter 8 provides the results of comparative analysis experiments among PHONN, THONN, PT_HONN and MPHONNG. The accuracy of the MPHONNG results is quite satisfactory.

Chapter 9 presents using MPHONNG model for estimating heavy rainfall from satellite data, introduces ANSER-plus system and using the results of estimating with MPHONNG to compare with others models.

Chapter 10 provides the conclusion of this thesis. It includes the recapitulation of the contributions of the thesis and lists the most significant results from the thesis. It also provides a perspective on avenues for my future research for the issue of electricity supply net, the problem of Power Factor Controlling with Neural Networks.

Appendix A consists of three papers, which have been published in different international conferences.
Chapter 2. Knowledge Acquisition in Neural Network Field

Firstly a general review of knowledge acquisition in Neural Networks will be provided in this chapter, because this review has provided the background of my research. It includes prior knowledge and its relationship to a posteriori knowledge. Most of the review is based on an extensive literature search and an experimental study of representation in machine learning.

2.1 General Introduction to Artificial Neural Network

2.1-1 Artificial Neural Networks (ANNs)

It is more than fifty years that since McCulloch and Pitts developed their simplified single neuron model. Artificial Neural Networks (ANNs) are a set of information analysis tools based on a physiological understanding of the nervous systems and how it processes information. ANNs complement other information analysis tools such as multivariate statistics, clustering algorithms, probabilistic modeling, fuzzy logic, expert or knowledge-based systems, signal processing, conventional image processing, Fourier analysis, and wavelet analysis to name a few.

ANNs are mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements that are analogous to neurons and are tied together with weighted connections that are analogous to synapses. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Learning often occurs by example through training, or exposure to a truth set of input/output data where the training algorithm iteratively adjusts the connection weights (synapses). Learning can also occur through clustering of input data by self-organizing the connections. In both cases, these connection weights store the knowledge necessary to solve specific problems.
2.1-2 The History of Neural Network

Actually the development of ANNs has been undergo a long time. McCulloch & Pits first developed their simplified single neuron model in 1943. Widrow developed his ‘ADALINE’ and Posenblatt the ‘PERCEPTRON’ during the 1960’s. Multi-layer feed-forward networks (Multi-Layer Perceptrons or MLPs) and the back-propagation algorithm were developed during the late 1970’s, and Hopfield devised his recurrent network during the early 1980’s. The development of MLPs and ‘Hopfiled nets’ heralded a resurgence of worldwide interest in ANNs, which has continued unabated ever since.

ANNs are new types of computers based on models of biological neural networks. It should be emphasized that nobody fully understands how biological neural networks work. Despite this, ANNs have captured the imagination of both research scientists and practitioners alike - the prospect of producing computers based on the workings of the human brain is truly inspiring.

Despite a flurry of activity during the previous decade, ANNs remains a young field of research. It offers a new approach to computing which develops mathematical structures with the ability to learn. The methods are loosely inspired by academic investigations into modeling the nervous system’s learning processes. It has been repeatedly demonstrated that ANNs can be used to solve many real-world problems, and indeed are excellent for pattern recognition/classification tasks in particular.

2.2 Techniques for Prior Knowledge Usage

For any learner with a representational ability, learning can be viewed as a search through the range of implementable functions, to find the function that most closely approximates the desired problem. Methods for utilizing the information contained in prior knowledge can thus be viewed as attempting to restrict or bias the space of implementable function for a particular learner.

Previous attempts at practical methods for the incorporation or transferal of prior knowledge in neural networks can be divided roughly into three groups: weight techniques, structural techniques and learning techniques.

a) Weight techniques, where the prior knowledge to be used is encoded in the weights of trained neural networks.

b) Structural techniques in which the prior knowledge is hard-coded into
the network architecture.

c) Learning techniques, which attempt to modify the way learning is conducted on the basis of prior knowledge.

The multiplayer perceptron neural network is the most widely used type of neural network. There are numerous successful applications in various fields. To mention a few: Hong C. Leung and Victor W. Zue (1989), Yeshwant K. Muthusamy and Ronald A. Cole (1992) have done some applications in speech recognition. Timothy S. Wilkinson, Dorothy A. Mighell and Joseph W. Goodman (1989) have done a good job by applying it to image processing. Charles Schlay, Yves Chanvin, Van Henkle and Richard Golden (1991) have used it in control applications.

In finance, Apostolos Nicholas Refenes, Achileas Zapranis and Gavin Francis (1994) applied it to assets allocations. In this application, the multi-layer perceptron network is compared with a classical method for stock ranking, i.e., multiple linear regression. It was found that the network outperforms regression in terms of out-sample mean-square-error. Also, the performance was tested and found to be stable for various network architectures and training parameters. The sensitivity of the output to various inputs is also examined in detail in this work.

In rainfall estimation, since Ming Zhang and Roderick R. Scofield (1994) applied Artificial Neural Network for rainfall estimation, achievements have been gotten. They tried to use many methods in this area [1.4.119.126].

2.2-1 Basic Principles

There are many different types of models that can be labeled ‘artificial neural networks’. Before going into each specific network type, I will introduce some notations and graphical representations in networks commonly used in the literature. It is best to start with the most basic processing unit in the network: the ‘neuron’. As a processing unit, it will receive ‘inputs’. Then, some transformation will be made to the inputs to obtain an ‘output’.

The transformation can be carried out in two stages. In the first stage, either a linear combination of all the inputs or a norm of the difference between the inputs and the center of the hidden unit will be applied to obtain a scalar, called the ‘net’.

The coefficients of the linear transformation or the center of the hidden units are called the ‘weights’. The processing structure of the neuron can be divided into
two different types and two different modeling functions. The global approximation and the local approximation can also be introduced here. The details will be discussed in later sections.

In the second stage, a non-linear transformation will be carried out on the net to obtain the output. The function used for the non-linear transformation is called the transfer function. To sum up, the whole process is stated in equation 2.1.

\[
y = f\left(\sum_{i=1}^{n} x_i w_i + w_0 \right)
\]

Or

\[
y = f\left\{\left[\sum_{i=1}^{n} (x_i - w_i)^2 \right]/w_0 \right\}
\]

(2.1)

Where \( y \) is the output of the unit, \( x_i \) is its \( i^{th} \) input, \( w_i \) is the weight associated with the \( i^{th} \) input, \( n \) is the number of inputs and \( f \) is the transfer function. The graphical representation of the process is given in Figure 2.1.

![Figure 2.1 A single node with weighted inputs](image)

Most network structures can be organized in layers. A layer consists of a number of neurons. There are no connections between the neurons in the same layer but the neurons belonging to different layers are connected. A typical network structure with two layers is given by equation 2.2.

The first layer is called the input layer, and contains one node (or neuron) for
each of the training data entry. The last layer is called the output layer and contains one neuron for each of the network outputs. Between the input and output layers are an arbitrary number of hidden layers each containing an arbitrary number of neurons. Each neuron is connected to every other neuron in adjacent layers by a set of weights.

The weights define the ‘strength’ of the flow of information from one layer to the next through the network. Each weight can take on any positive or negative value. ‘Training’ a neural network is simply the process of determining an appropriate set of weights so that the network accurately approximates the input/output relationship of the training data.

Equation 2.2 corresponds to the graphical representation shown in Figure 2.2.

First layer:

\[ y_j = f_2 \left( \sum_{k=1}^{n} w_{k,j}^2 y_k^1 + w_{0,k}^2 \right) \]

Second layer:

\[ f_k^1 = f_1 \left( \sum_{i=1}^{n} w_{i,k}^1 x_i^1 + x_{0,k}^1 \right) \]  \hspace{1cm} (2.2)

Where:

- \( y_j \) the \( j \)th network output
- \( y_j^1 \) the \( j \)th output of the first layer
- \( x_i \) the \( i \)th network input
- \( w_{i,k}^1 \) the weight between \( i \)th input and the \( k \)th hidden unit
- \( w_{k,j}^2 \) the weight between the \( k \)th hidden unit and the \( j \)th output
- \( f_1 \) the transfer function in the first layer
- \( f_2 \) the transfer function in the second layer
- \( p \) the number of network outputs
- \( r1 \) the number of network inputs and \( r2 \) is the number of hidden units
Figure 2.2 The structure of the artificial neural network

2.2-2 Structure

As mentioned in this section, the multi-layer perceptron is a global approximate. The net of its hidden units is a linear combination of the network-input units. Then, the net will be put into a transfer function. Last a linear combination of all the outputs of the hidden units is obtained as the network output.

The network is global in the sense that the hidden units will react with all the inputs from any part of the input space, rather than with some of them. In detail, each hidden unit divides the whole input space into two regions and assigns two different values to the inputs from the two different regions. The overall process of the network is summarized in equation 2.3 and is represented by figure 2.3.

\[
y_j = f_m(\ldots f_2(\sum_{k=1}^{n} w_{k,j}^2 y_k^1 + w_{0,k}^2 f_1(\sum_{l=1}^{n} w_{l,k}^1 + x_{0,k}) \ldots ))
\]  
(2.3)
Figure 2.3 The multi-layer the artificial neural network structure

Where:

- \( y_j \) the \( j^{th} \) network output
- \( x_i \) the \( i^{th} \) network input
- \( w_{m,k}^{i} \) the weight between the \( i^{th} \) input and the \( k^{th} \) unit in the \( m^{th} \) layer
- \( f_m \) the transfer function in the \( m^{th} \) layer
- \( q \) the number of layer
- \( p \) the number of network outputs
- \( r^m \) the number of inputs in the \( m^{th} \) layer
- \( w_{i,k}^{o} \) the weights associated with a constant input are called bias

In practice, nearly all applications of the multi-layer perceptron network have only one hidden layer. Moreover, according to a theorem in Tianping Chen and Hong Chen (1995), a multi-layer perceptron network can approximate nearly any function with only one hidden layer. This result means that a solution always exists from a single hidden layer network but it says nothing about how difficult it is to obtain that solution.

It is possible that the solution for a particular problem can be obtained in an easier way if we use more hidden layers. To sum up, we need to pay special
attention to the single layer network but need not reject other possible structures.

Besides the number of hidden layers, the number of hidden units in each layer needs to be determined. It depends on the complexity of the system being modeled. For a more complex system, a larger number of hidden units are needed. In practice, the optimal number of hidden units is found by trial and error. Let us see some examples in the real world.

A speculative paper by Schimdhuber deals with issues concerning the embedding of 'meta' -- levels in neural network architecture [58]. It is based on the idea that a network could examine binary values of 0 and 1, and then changes its own internal state by the use of appropriate feedback. No experimental results were given for this technique, however it is an interesting idea.

Brown, in the paper Gragy Layer Technology: Incorporation A Prior Knowledge into Feedforward Artificial Neural Networks [51] demonstrates the use of a technique that constrains the weights within a hidden 'grey' layer according to the prior knowledge that is known about the desired function approximation. Good results are demonstrated for a single example where the non-linear dynamics of a control system is to be approximated A method for constraining the weights on general problems is not suggested within the paper.

Weight sharing, where several network connections are forced to share the same weight value, has been successfully applied to the problem of building symmetry invariance into networks. Improvements in both the generalisation ability and learning times for problems requiring such invariance have been found [60].

It can be shown that the network structure and weights for approximating both linear and non-linear differential equations can be calculated by using a generalisation of the Taylor series [50]. This technique allows the direct integration of any a priori knowledge about the differential equation to be factored into the network design. It is however limited to application domains that use differential equations, such as the prediction of time series.

Probably the most generally applicable of the structural techniques is problem decomposition. Here a problem is broken down into one large network. The methods described in the 'weight techniques' can still be applied to these modular networks to yield even greater performance increase.

2.2-3 Weights

As described in later sections, the training of a multi-layer perceptron network
can be easily implemented by gradient methods. A derivative of the transfer function is needed for training. Thus, a differential transfer function is widely used in order to facilitate training. One commonly used transfer function for multi-layer perceptrons is the sigmoid function, given by equation 2.4

\[ f(x) = \frac{f(x)}{1-f(x)} \]  

(2.4)

Another commonly used transfer function is the hyperbolic tangent function given by equation 2.5.

\[ f(x) = 1 - f(x)^2 \]  

(2.5)

Probably the simplest type of transfer known is the literal transfer, where the source weights are copied directly for use as the initial weights for training a new problem. Sharkey [59] has examined this situation for some simple classification problems.

He found that both positive transfer, leading to decreased training times, and negative transfers, which increase training times, could occur over random initialization, under certain circumstances. Negative transfers tended to occur when the type of output classification was changed between tasks, and positive transfer was more likely in the case where only the input classification was changed.

All the networks considered in this thesis have either consistent outputs (e.g. for character recognition, a character will be classified with the same output, independently of the training set font). The consistent use of inputs and/or output is a necessary feature of training within the same problem environment.

**Discriminability** Based Transfer (DBT)[57] uses the relevance of input layer hyperplanes (as determined by the weight values into the neurons) on the new task to determine whether the weights for each input neuron layer should be allowed to change.

All other weights in higher layers are randomly initialized. The hyperplane relevance is determined by using an entropy measure or mutual information metric, which relies on determining class boundaries in the training data.

Enhanced training speed, relative to a small subset of randomly initialized networks, were shown across a range of real world classification tasks. DBT is constrained to systems in which a target class can be assigned for each of the inputs, and so cannot be used for problems such as surface approximation or regression.
Some other weight-based methods of utilizing prior knowledge revolve around the use of traditional AI methods to generate appropriate weights [112]. These systems rely on the knowledge of a human expert to initialize weight vectors. In the same way the similarity between fuzzy systems and radial basis functions also allows the direct incorporation of fuzzy rules into neural type systems [113]. The direct incorporation of these 'expert' rules into a neural network is presented in [114], so these techniques are not discussed further.

2.2-4 Learning Techniques

Learning is the process by which a neural network modifies its weights in response to external inputs.

In traditional programming, where highly structured programs such as FORTRAN are used, the programmer will be given the inputs, some type of processing requirements (what to do with the inputs), and the desired output. The programmer's job is to apply the necessary, minute, step-by-step instructions to develop the required relationship between the input and output.

Knowledge-based programming techniques (expert systems) use higher-level concepts to specify relationships between inputs and outputs. These higher-level concepts are referred to as heuristics, or more commonly, rules.

In contrast, neural networks do not require any instructions, rules, or processing requirements about how to process the input data. In fact, neural networks determine the relationship between input and output by looking at examples of many input-output pairs. This unique ability to determine how to process data is usually referred to as self-organization. The process of self-organizing is called adaptation, or learning.

Pairs of input and outputs are applied to the neural network. These pairs of data are used to teach or train the network, and as such are referred to as the training set. Knowing what output is expected from each of the inputs, the network learns by automatically adjusting or adapting the strengths of the connections between process elements. The method used for the adjusting process is called the learning rule.

How fast does learning occur? That depends on several things. There are trade-off in the rates of learning. Obviously, a lower rate means that a lot more time is spent in accomplishing the off-line learning to produce a trained system. With a faster rate however, the network may not be able to make the fine discriminations possible with a system that learns slower. Researchers are
working on giving us the best of both worlds.

Consider accuracy and speed with the following illustration. Once a system had learned a 500-matrix image consisting of 500 three-digit decimals ranging from .000 to 1.000 in less than three minutes, we purposely altered one digit of a three-digit decimal. Upon recall, the neural network detected this change every time. This is analogous to learning the image of a dollar bill in pixel form. Now, if we alter one pixel in George Washington’s eye, the neural network will detect it instantly.

Finally, the learning rule and the modification of the weights play a smaller but sometimes lengthy role in the training effort. Again, it depends on the particular problem for which the network was developed. We do find that, as before, imaging or pattern classification networks are several orders of magnitude simpler to train: little or no adjustment to the weights is required, and learning rules do not have to be changed.

Most learning equations have some provision for a learning rate, or learning constant. Usually this term is positive and between 0 and 1. If the learning rate is greater than 1, it is easy for the learning algorithm to 'overshoot in correcting the weights, and the network may oscillate.

Small values of learning rate will not correct the error as quickly, but if small steps are taken in correcting errors, there is a better chance of arriving at the minimum error and thus the optimum weight settings. The learning rate is, then a measure of the speed of convergence of the network.

A 'Meta' Neural Network (MNN) that learns to adjust the learning parameters by observing the changes in weights during training is presented in Meta-Neural Networks that learn by learning [115]. This technique is intended to allow the overall training speed on similar problems to be increased, by getting the MNN to choose an optimum step size and direction vector for a gradient descent learning algorithm. On simple four-bit parity and two-class problems this technique was shown to have significant speed improvement, however no follow-up work has been published to date.

Abu-Mostaf in the papers of Hints and the VC Dimension and Hints [116, 117] show how Hints can be integrated into the learning process by generating additional training examples from the prior knowledge. This can be an effective technique in environments where limited training data is available[118].

To train a network for a specific problem from within this environment, the new network is placed ‘on-top’ of the environment network such that its inputs are connected to the environment networks outputs. These outputs are intended to be
invariant under the similarity transforms applicable to a particular environment. The research is backed by some rigorous theoretical justification but is demonstrated only on toy problems.

2.2-5 Training (Finding the Weights) by way of Back Propagation (BP)

Most of the training methods for multi-layer perceptron networks can be classified as Gradient methods, because they all adjust the weights according to the gradient of the error with respect to the weights. These methods are widely used and perform very successfully in many applications. However, it is well known that these methods can only obtain locally optimal solutions.

If the data is very noisy or some special form of error function is used, then many locally optimal solutions may exist, and these methods will fail to find the global optimal. This kind of situation is very likely in financial forecasting. Thus, some modifications to the gradient methods or a different method altogether is needed. But here we will review the gradient training methods only. The simplest one is Back Propagation (BP).

Rumelhart, Hinton and Williams proposed the BP algorithm in 1986 for setting weights and hence for training multi-layer perceptrons. This opened the way for using multi-layer ANNs, observing that the hidden layers have no desired (hidden) outputs accessible.

Once the BP algorithm of Rumelhart et al. was published, it was very close to algorithms proposed earlier by Werbos in his Ph.D. dissertation in Harvard in 1974 and then in a report by D. B. Parker at Stanford in 1982, both unpublished and thus unavailable to the community at large.

It goes without saying that the availability of a rigorous method to set intermediate weights, namely to train hidden layers of ANNs, gave a major boost to the further development of ANN, opening the way to overcome the shortcomings of single-layer networks that had been pointed out by Minsky, and which nearly killed the ANNs.

A classical type of training method for the multi-layer perceptron network is the gradient method. It is the most widely used method. The simplest form is the BP. The idea of the method is very simple. It adjusts the weights \( w_j \) in a way such that the error, \( E \) (between simulation and data), is reduced. Specifically, the weights are adjusted repeatedly in a way proportional to the negative of its partial derivative on the error function, as expressed by equation 2.6.
\[ \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \]  

(2.6)

Where \( \eta \) is called the learning constant. There are two reasons for the adjustment. Firstly, weights are changed in a direction of reducing the error. The weight with more impact on the error will be adjusted more. Overall, the weights move in a direction that reduces most of the error. The training will be continued until a target error has been reached or a predetermined number of iterations have been performed.

Before the training starts, it is necessary to determine the learning constant: an explicit expression of the partial derivative is needed. The partial derivative can be expressed in terms of the outputs and the network error by the chain rule as given by equation 2.7

\[
\frac{\partial E}{\partial w_j^m} = \sum_{j=1}^{m} \frac{\partial E}{\partial y_j^m} \frac{\partial y_j^m}{\partial net_j^m} \frac{\partial net_j^m}{\partial w_j^m}
\]

(2.7)

where:

- \( y_j^m \) the \( j^{th} \) output in the \( m^{th} \) layer
- \( net_j^m \) the \( j^{th} \) net in the \( m^{th} \) layer
- \( w_j^m \) the \( i^{th} \) weight in the \( m^{th} \) layer

In particular, we can assume a mean squared error function and sigmoid transfer function.

For the weights in the output layer or the \( q^{th} \) layer, the derivative can be expressed by equation 2.8

\[
\frac{\partial E}{\partial w_j^m} = 2y_j^m(x_j^m)(1 - y_j^m)x_i^m
\]

(2.8)

For the weights in the hidden layers, the derivative can be expressed as in equation 2.9
\[ \frac{\partial E}{\partial w_j^m} = \sum_{j=1}^{n} \delta_j^m x_j y_j^m (1 - y_j^m) x w_j^q \]

\[ \text{(2.9)} \]

Where:

- \( y_j^m \) the output of the \( j \)th unit in the \( m \)th layer
- \( r_m \) the number of units in the \( m \)th layer
- \( \delta_j^m \) the partial derivative of the error with respected to \( y_j^m \)

The \( \delta_j^m \) can be found by equation 2.10

\[ \partial w_j^m = \sum_{j=1}^{n+1} \delta_j^{m+1} x y_j^{m+1} (1 - y_j^{m+1}) x w_{ij}^{m+1} \]

with \( \delta_j^q = 2y_j \)

\[ \text{(2.10)} \]

Where \( q \) is the number of layers. For a network with another error function or transfer function, the derivative can be calculated in a similar way.

To determine the learning constant (learning rage), a simple way is to use a fixed constant based on past experience.

Alternatively, the constant can be adjusted dynamically during the training. The training starts with a small learning constant. Then, if the error is decreasing, the learning constant is increased by a small in each successive iteration.

However, when the error starts to increase, the learning constant will be cut by a large amount to maintain the error level. In effect, the method maintains the highest possible learning constant at any time to ensure a fast convergence of the training.
Chapter 3. Ordinary Higher Order Neural Networks

In this chapter, I will introduce some ordinary higher order neural network models. This is due to they are basis of my research. During my studying, I used these models for comparing with my new model (MPHONNG), they are HONN, PHONN, THONN and PT-HONN.

3.1 Higher Order Neural Network (HONN)

The term “higher order” neural network means different things to different people-some authors use the term to refer to the neuron activation function, some to imply preprocessing of the neuron inputs, others to simply indicate connections to more than one layer. Still others use the term to refer to a neural network’s functionality, rather than its architecture (i.e. its ability to extract high-order correlation from the training data).

Following the lead of Giles & Maxwell (1987), we are able to formulate the output from first-order neurons as follows in the equation 3.1

\[
y_i(x) = f[\sum_j W(i, j)x(j)]
\]

(3.1)

Where:
\{x(j)\} = an n-element input vector,
\[W(i, j)\]=adaptable weights from all other neurons to neuron-\(i\), and
\(f\) = neuron threshold function (e.g. sigmoid).

Such units (neurons; nodes) are said to be linear, since they are only able to capture first-order correlations in the training data. Higher order correlations require more complex units, characterised by h Giles & maxwell in the following manner:

\[
y_i(x) = f[W_i(i) + \sum_j W_j(i, j)x(j) + \sum_{j, k} W_{jk}(i, j, k)x(j)x(k) + ...]
\]

(3.2)

Units which include terms up to and including degree-k are referred to as kth-order nodes. An alternative (simpler) formulaition is that of Lisboa & Perantonis (1991):
\[ y_i = f\left[ W_i^0 + \sum_l \sum_k \sum_j \ldots \sum_m W_{ijkl \ldots m} x_i x_k x_l \ldots x_m \right] \]

(3.3)

Where a single weight is applied to all n-tuples \( x_i \ldots x_m \) in order to generate the output \( y_i \) from that node. This formulation is reminiscent of the Sigma-Pi units of Rumelhart, Hinton & McClelland (1986):

\[ \sum W_{ij} a_{i1} a_{i2} \ldots a_{ik} \]

(3.4)

and for which they prove that the Generalised Delta Rule can be applied as readily as for simple additive units (\( \sum W_{ij} a_i \)).

In summary, HONNs include multiplicative terms in their activation function. Now it is possible to perform these data multiplications within a preprocessing stage; the major computational load then becomes a function of the large number of weights. It should also be pointed out that the output of a \( k^{th} \)-order single-layer HONN node is a nonlinear function of up to \( k^{th} \)-order polynomials. Moreover, since no hidden layers are involved, both Hebbian and Perceptron learning rules can be used (Shin & Ghosh, 1991).

### 3.2 Polynomial Higher Order Neural Networks (PHONN)

PHONN is that to use a combination of linear, power and multiplication neurons for simulation the data.

Its form of a definition is listed in the following:

\[ Z = \sum_{ij=0}^{n} a_{ij} x^i y^j \]

(3.5)

There are three major different PHONN models: Model-0, Model-1 Model-2. Model-0 is the general model. Model-1 and model-2 are the improved polynomial higher order neural network models.
3.3 Trigonometric Polynomial High Order Neural Network

(THONN)

The Trigonometric Polynomial Higher Order Neural Network Group (THONG) model is an artificial neural network set in which every element is a higher order trigonometric neural network ($ht$). The domain of the neural network $ht$ inputs is the n-dimensional real number $\mathbb{R}^n$. Likewise, the $ht$ outputs belong to the m-dimensional real number $\mathbb{R}^m$. The neural network function $f$ is a mapping from the inputs of $ht$ to its outputs. See the formula below:

$$ht \in \text{THONG} \quad \text{(where: } ht = f: \mathbb{R}^n \rightarrow \mathbb{R}^m)$$

$$Z = \sum_{i=0}^{n} a_i \sin^i(x) \cos^i(y)$$

(3.6)

Trigonometric Polynomial Higher Order Neural Network
3.4 Polynomial and Trigonometric Polynomial High Order Neural Network (PT-HONN)

The network architecture of PT-HONN combines both the characteristics of PHONN and THONN. It is a multi-layer that consists of an input layer with input-units, and output layer with output-units, and two hidden layers consisting of intermediate processing units. A specific definition of PT-HONN is presented in the following:

\[ Z = \sum_{i,j=0}^{n} a_{ij} \sin^i(x) \cos^j(y) + b_{ij} x^i y^j \]

(3.7)
Chapter 4. The Theoretical Basis of the Multi-Polynomial Higher Order Neural Network Group Models Program

This chapter presents the theoretical basis of the models that I have developed. The key idea in this chapter is to use group theory to study the structure of neural network group models. The reasons are given for using neural network group models. The definition of a neural network group is introduced. The neural network group models and features are then discussed in some detail, using the same format for all three models. As an illustrative example of how to apply neural network group models to real world problems, we have explored the topic of using neural network group model for data simulation and estimation.

4.1 The aim of introduction

Very little artificial neural network research has concentrated on the precursors of neural network group models. Examples of such work are the integrated neural network of Matsulka et al. [25], or Pentland and Turk’s holistic model [26]. Luner [27] proposed anew mechanism, selective attention among perceptual groups as part of his early vision on computational models. In his model, perceptual grouping is initially performed on ‘connectionist networks’ by dynamically binding the neural activities triggered in response to related image features.

Lie Groups were used in Tsao’s [28] group sets approach to the computer simulation of 3D rigid motion. More specifically, motion is expressed as an exponential mapping of the linear combination of six infinitesimal generators of the one-parameter Lie subgroup.

Hu [29] proposed a level-by-level learning scheme for artificial neural groups. This learning method closely resembles the process of knowledge growth observed in both human individuals and society. Further, leads to improved generalization and learning efficiency.

The neural network hierarchical model was devised by Willcox [30] consists of binary-state neurons grouped into clusters, can be analysed using a Renormalisation Group (RG) approach.

Unlike the research previously described, Yang [31] pays attention to the activities of neuron groups. His work, together with Naimark’s [32] earlier theory of group representations, is used as the basis for neural network group sets, which is developed in the following sections.

1) Neural network-based models developed so far are not yet sufficiently
powerful to characterize complex systems. Moreover, a gap exists in the research literature between complex systems and general systems. A step towards bridging this gap can be made using neural network group sets.

2) As mentioned earlier, neural networks can effectively simulate a function if it varies in a continuous and smooth fashion with respect to the input variables. However, in the real world data such variations can be discontinuous and non-smooth. Accordingly, if we use only simple neural networks to simulate these functions, then accuracy is a problem. Neural network groups are capable of performing much better. It is possible to simulate discontinuous functions to any degree of accuracy using neural network group sets, even at points of discontinuity.

3) Neural network are massively parallel architectures. Thus, by using parallel, ANN-based reasoning networks, we can compute all the rules, models, knowledge and facts stored in the different weights simultaneously.

However, real-world reasoning is invariably complex, nonlinear and discontinuous. Thus, simple neural network models may not always yield correct reasoning, whereas neural network groups, possibly may.

I provide an example in the following section to show how it is possible to reason more accurately, using neural network group models, even in complex situations.

The key idea in this chapter is to use group theory to study the structure of neural network groups. In this context, we need to concern ourselves with the following:

a) Rationale for using neural network groups.

b) Neural network group definitions.

c) Neural network group models and features.

d) An illustrative example of how to apply neural network group models to a real-world problem.

e) A brief discussion of neural network group models.
4.2 Artificial Neural Network Sets

A set [33] is defined as a collection of elements, which possess some properties. The symbol

\[ a \in A \]  

(4.1)

This means that A is a set, and a is an element of set A.

The Artificial Neural Network Set [1] is a set in which every element is an ANN. The symbol I have used is nns:

\[ \text{nns} \in \text{NNS} \text{ (where nns} = f : \mathbb{R}^n \rightarrow \mathbb{R}^m) \]  

(4.2)

This means that NNS is an ANN set. And nns is an element of the set NNS. The nns is also one kind of artificial neural network.

The domain of the neural network nns inputs is the n-dimensional real number \( \mathbb{R}^n \). Likewise the nns outputs belong to the m-dimensional real number \( \mathbb{R}^m \). The neural network function \( f \) is a mapping from the inputs of nns to its outputs.

The BPN set is a set whose elements are all BackPropagation Networks. The function of the BPN \( f - \) is defined as follows:

An input vector

\[ X_p = (X_{p1}, X_{p2}, X_{p3}, \ldots, X_{pn})^t \]  

(4.3)

is applied to the input layer of the BPN network. The input units (nodes) connect to the hidden-layer units via variable (adaptable) weights. The net input to the jth-hidden unit is

\[ net_{pj} = \sum_{i=1}^{n} w_{ji}^h x_{pi} + \theta_j^h \]  

(4.4)

where \( w_{ji}^h \) is the weight of the connection from the ith input unit, and \( \theta_j^h \) is the bias term. The "h" superscript denotes quantities in the hidden layer.

Assuming the activation of this node is equal to the net input, the output of this node is then:

\[ i_{pj} = f_j^h (net_{pj}^h) \]  

(4.5)
The equations for the output nodes are:

\[ \text{net}_{pk}^o = \sum_{i=1}^{l} w_{ij}^o i_{jl} + \theta_k^o \]  \hspace{1cm} (4.6)

\[ o_{pk} = f_k^o(\text{net}_{pk}^o) \quad k = 1, 2, 3, \ldots, m \]  \hspace{1cm} (4.7)

The set $\text{BPN}^{1,3,256}$ is one in which every element is a backpropagation network comprising one output neuron, one hidden layer (of 3 neurons), and 256 input neurons. Moreover, $\text{BPN}$ is a subset of the neural network set $\text{NNS}$:

\[ \text{BPN}^{1,3,256} \subset \text{BPN} \subset \text{NNS} \]  \hspace{1cm} (4.8)

Many other kinds of neural networks exist apart from the Backpropagation Network (BPN), such as:

- Additive Grossberg network (AG). [Grossberg (1986)]
- (binary) Adaptive Resonance Theory based (ART1). [Carpenter & Grossberg (1986), based upon the initial concept of adaptive resonance theory introduced by Grossberg (1976)]
- Continuous Hopfield (CH). [Cohen & Grossberg (1983); Hopfield (1984)]
- Bidirectional Associative Memory (BAM). [Scoffer (1986); Kosko (1986)]
- Self-Organising Map (SOM). [Kohonen (1984)]
- Learning Vector Quantisation (LVQ). [Kohonen (1988)]
- Cascade Correlation (CC). [Fahlman & Lebiere (1990)]

This list is not meant to be exhaustive, new ANN models continue to be reported in the literature on a regular basis.

We define the neural network set $\text{NNS}$ as the union of the subsets $\text{BPN}$, $\text{LVQ}$, and so on. See Figure 4.1. In formal notation, we write

\[ \text{NNS} = \text{BPN} \cup \text{AG} \cup \text{ART1} \cup \text{CH} \cup \text{BAM} \cup \text{SOM} \cup \text{LVQ} \cup \text{CC} \cup \ldots \]  \hspace{1cm} (4.9)
Figure 4.1. The artificial neural network set NNS and its subsets.
4.3 Generalised Artificial Neural Network Set

We proceed to define product generalised and additive generalised Artificial Neural Network sets (ANN sets). This is consistent with Group Theory generally, since if an element $n$ is a member of a group, then its inverse $1/n$ is also a member of the same group.

4.3-1 Generalised product sets of artificial neural network sets

A generalized product set of ANN set $- \text{NN}^*$ [1] is a set in which each element $nn_i^*$ is either an artificial neural network or a product of neural networks, for which the following conditions hold:

a) $nn_i^* \in \text{NNS}$:

- or if the product $nn_i^* nn_j^*$ is defined for every two elements $nn_i^*, nn_j^* \in \text{NNS}$, then $nn_i^*, nn_j^* \in \text{NN}^*, \forall nn_i^*, nn_j^* \in \text{NNS}$;

- or if the product $nn_i^* nn_j^*$ is defined for every two elements $nn_i^* \in \text{NNS}$ and $nn_j^* \in \text{NN}^*$, then $nn_i^*, nn_j^* \in \text{NN}^*, \forall nn_i^* \in \text{NNS}$ and $nn_j^* \in \text{NN}^*$;

- a product $nn_i^* nn_j^*$ is defined for every two elements $nn_i^*, nn_j^* \in \text{NN}^*$, then $nn_i^*, nn_j^* \in \text{NN}^*, \forall nn_i^*, nn_j^* \in \text{NN}^*$

b) $\exists$ an element $nn_{ie}^*$ in $\text{NN}^*$ such that

$$nn_{ie}^* nn_i^* = nn_i^* nn_{ie}^* = nn_i^* \forall nn_i^* \in \text{NN}^*$$

($nn_{ie}^*$ is called the identity element of the set $\text{NN}^*$);

c) For every element $nn_i^* \in \text{NN}^*$, $\exists$ a unique element, designated $nn_i^*^{-1}$, for which $nn_i^* nn_i^*^{-1} = nn_i^*^{-1} nn_i^* = nn_e^*$,

(the element $nn_i^*^{-1}$ is called the inverse of $nn_i^*$).
4.3-2 Additive Generalised Artificial Neural Network Sets

An additive generalised ANN set – \( \mathbb{NN}^+ \) is a set in which element \( nn_i^+ \) is either an artificial neural network or an additive generalised neural network, for which the following hold:

a) \( nn_i^+ \in \mathbb{NNS} \):
   - or if an addition \( nn_i^+ \) \( nn_j^+ \) is defined for every two elements \( nn_i^+ , nn_j^+ \in \mathbb{NNS} \), then \( nn_i^+ , nn_j^+ \in \mathbb{NN}^+ , \forall \ nn_i^+ , nn_j^+ \in \mathbb{NNS} \);
   - or if an addition \( nn_i^+ \) \( nn_j^+ \) is defined for every two elements \( nn_i^+ \in \mathbb{NNS} \) and \( nn_j^+ \in \mathbb{NN}^+ \), then \( nn_i^+ , nn_j^+ \in \mathbb{NN}^+ , \forall \ nn_i^+ \in \mathbb{NNS} \) and \( nn_j^+ \in \mathbb{NN}^+ \);
   - or if an addition \( nn_i^+ \) \( nn_j^+ \) is defined for every two elements \( nn_i^+ , nn_j^+ \in \mathbb{NN}^+ \), then \( nn_i^+ , nn_j^+ \in \mathbb{NN}^+ , \forall \ nn_i^+ , nn_j^+ \in \mathbb{NN}^+ \)

b) \( \exists \) an element \( nn_{io}^+ \) in \( \mathbb{NN}^+ \) such that
   \[ nn_{io}^+ + nn_i^+ = nn_i^+ + nn_{io}^+ = nn_i^+ , \forall \ nn_i^+ \in \mathbb{NN}^+ \]
   (\( nn_{io}^+ \) is called the identity element of the set \( \mathbb{NN}^+ \));

c) For every element \( nn_i^+ \in \mathbb{NN}^+ \), \( \exists \) a unique element, designated \( nn_i^- \), for which \( nn_i^+ + (-nn_i^-) = (-nn_i^-) + nn_i^+ = nn_{io} \)
   (the element \(-nn_i^-\) is called the inverse of \( nn_i^+ \)).

4.3-3 Generalised ANN Sets

The generalised artificial neural network set \([103] – \mathbb{NN} \) is the union of the product set \( \mathbb{NN}^* \) and the additive set \( \mathbb{NN}^+ \). Each element \( nn_i \) is either an artificial neural network of a generalised neural network. We write:

\[ \mathbb{NN} = \mathbb{NN}^* \cup \mathbb{NN}^+ \quad (4.10) \]

For example, the generalized BPN set is derived as follows:

\[ GBPN^{1,3,256} = BPN^{*,1,3,256} \cup BPN^{1,3,256} \quad (4.11) \]

For defining the unique \( bpn^{*,1,3,256}_i \) and \( bpn^{1,3,256}_i \) elements of the inverse of
bpm\textsuperscript{1,3,256}, we assign the same structures and weights to them, except that the inverse element has outputs of opposite sign.

Figure 4.2 shows the relationship between neural network set NNS and the generalised neural network set NN.

![Venn Diagram](image)

**Figure 4.2** Relationship between NNS and NN (=NNS* \( \cup \) NNS\textsuperscript{+})

The elements of set NNS are neural networks, the elements of set NN, by contrast, are neural network groups. The difference between a neural network and a neural network group was illustrated previously in Figure 4.2, in which we see that the neural network set NN is a more general form of neural network set NNS. Accordingly, neural network groups should hold more potential for characterizing complex systems. Moreover, the previously mentioned gap between complex systems and general systems could be bridged by neural network group sets.

Now, let us introduce the Multi-Polynomial Higher Order Neural Network Group models (MPHONNG) that I propose as new models.

I will start by developing the neural network group set, and then introduce and discuss the Multi-Polynomial Higher Order Neural Network Group models (MPHONNG).
4.4 Neural Network Group

4.4-1 Product notation of artificial neural network groups

A nonempty set \( \mathbb{N} \) is called a neural network group, if \( \mathbb{N} \subseteq \mathbb{NN} \) (the generalized neural network set), and the product \( n_i n_j \) is defined for every two elements \( n_i, n_j \in \mathbb{N} \), and for which the following conditions hold:

a) \( n_i n_j \in \mathbb{N}, \forall n_i, n_j \in \mathbb{N}; \)

b) \( (n_i n_j) n_k = n_i (n_j n_k), \forall n_i, n_j, n_k \in \mathbb{N}; \)

c) \( \exists \) a unique element \( n_e \) in \( \mathbb{N} \) such that \( n_e n = nn_e = n \forall n \in \mathbb{N}; \)

\( (n_e \) is called the identity element of the group \( \mathbb{N}; \)

d) for every element \( n \in \mathbb{N}, \\exists \) a unique element – designated \( n^{-1} \) – for which \( n n^{-1} = n^{-1} n = n_e \) [the element \( n^{-1} \) is called the inverse of \( n \); it is obvious that \( n \) is the inverse of \( n^{-1} \), so that \( (n^{-1})^{-1} = n \)].

Consider, for example, the generalised product backpropagation neural network set \( \mathbb{BPN}^* \). This is a neural network group, if ‘product’ is taken to mean ‘multiplication’ of the network outputs. Alternatively, the Boolean operator “logical AND” could be used for ‘product’. The generalised product backpropagation neural network set \( \mathbb{BPN}^* \) satisfies conditions (a), (c) and (d) already. Figure 4.3-A & -B show the product of elements in \( \mathbb{BPN}^* \). Now we prove that condition (b) also holds for \( \mathbb{BPN}^* \).

Proof:

\[ \forall bpn^{*1,256}_i, bpn^{*1,256}_j, bpn^{*1,256}_k \in \mathbb{BPN}^{*1,256}; \]

Let \( O_i, O_j \) and \( O_k \) be the outputs of \( bpn^{*1,256}_i, bpn^{*1,256}_j \) and \( bpn^{*1,256}_k \) respectively.

Because \( \mathbb{BPN}^{*1,256} \) has one output (which is a continuous real number function in \( \mathbb{R}^*_1 \)):

\[ O_i \in \mathbb{R}^*_1, \]
\( O_j \in R_0^1 \),

and \( O_k \in R_0^1 \),

because 'product' has been defined as multiplication of the neural network outputs,

\[
((bn^{1,3,256}_i) \times (bn^{1,3,256}_j)) \times (bn^{1,3,256}_k) = (O_i \times O_j \times O_k)
\]

and

\[
O_i \times (O_j \times O_k) = O_i \times (O_j \times O_k)
\]

In \( R^1 \),

\[
(O_i \times O_j \times O_k) = O_i \times (O_j \times O_k)
\]

therefore

\[
(O_i \times O_j \times O_k) = O_i \times (O_j \times O_k)
\]

and condition (b) holds.

Thus, since all four conditions hold, the product generalized backpropagation neural network set \( BPN^{1,3,256} \) is a neural network group.
Figure 4.3-A  Product Generalise BPN Group

Figure 4.3-B  Product Generalised BPN Group
4.4-2 Additive notation of artificial neural network groups

A nonempty set $\mathbb{N}$ is called a neural network group, if $\mathbb{N} \subseteq \mathbb{N}$ (a generalized neural network set), and an additon $n_i + n_j$ is defined for every two elements $n_i, n_j \in \mathbb{N}$, such that:

a) $n_i + n_j \in \mathbb{N}$, $\forall$ $n_i, n_j \in \mathbb{N}$;

b) $(n_i + n_j) + n_k = n_i + (n_j + n_k)$, $\forall$ $n_i, n_j, n_k \in \mathbb{N}$;

c) $\exists$ a unique element $n_0$ in $\mathbb{N}$ such that $n_0 + n = n + n_0 = n \forall n \in \mathbb{N}$; ($n_0$ is called the identity element of the group $\mathbb{N}$);

d) for every element $n \in \mathbb{N}$, $\exists$ a unique element, designated $-n$, for which $n + (-n) = (-n) + n = n_0$ [the element $-n$ is called the inverse of $n$; it is evident that $n$ is the inverse of $-n$, so that $(-n) = n$].

The Additive generalised backpropagation neural network set $\mathbf{BPN}^{+1,3,256}$ is a neural network group if “additon” is taken to mean “plus”. An alternative definition of “addition”, for example the Boolean operator logical – OR, could be given.

Based on the definition of generalised neural network, the generalised backpropagation $\mathbf{BPN}^{+1,3,256}$ satisfies conditions (a), (c) and (d). Figure 4.4-A & -B show the additon of elements in $\mathbf{BPN}^{+1,3,256}$. Now we prove $\mathbf{BPN}^{+1,3,256}$ also satisfies condition (b).

Proof:

$$\forall bpn^{+1,3,256}_i, bpn^{+1,3,256}_j, bpn^{+1,3,256}_k \in \mathbf{BPN}^{+1,3,256} :$$

Let $O_i$, $O_j$ and $O_k$ be the outputs of $bpn^{+1,3,256}_i$, $bpn^{+1,3,256}_j$ and $bpn^{+1,3,256}_k$ respectively.

Because $\mathbf{BPN}^{+1,3,256}$ has one output (which is a real number function in $\mathbb{R}^1$):

$$O_i \in \mathbb{R}^1,$$

$$O_j \in \mathbb{R}^1,$$

and $O_k \in \mathbb{R}^1,$
Because “addition” has been defined as “plus”,

\[(O_i + O_j) + O_k = O_i + (O_j + O_k),\]

in \(\mathbb{R}^1\) and condition (b) holds.

Thus, since all four conditions holds, the product generalized backpropagation neural network set \(\text{BPN}^{+1,3,256}\) is a neural network group.

---

**Figure 4.4-A**  Addition Generalise BPN Group
4.4-3 Neural Network Group Set (NNGS)

An artificial neural network group set is a set in which every element is an artificial neural network group. The symbol \( N \in \text{NNGS} \) means:

\[
\text{NNGS} \text{ is an artificial neural network group set and } N, \text{ which is one kind of artificial neural network group set can be written:}
\]

\[
\text{NNGS} = \{ \text{BPN}^{+1,3,256}, \text{BPN}^{+1,3,256}, \ldots \ldots \ldots \ldots \} \quad (4.9)
\]

In general, the neural network group set can be written:

\[
\text{NNGS} = \{ N_1, N_2, \ldots \ldots \ldots, N_i \} \quad (4.10)
\]

Where \( N_i \) is one kind of neural network group.

Figure 4.5 shows the relationships between the neural network set NNS, the generalized neural network set NN and the neural network group set NNGS.
Figure 4.5 Relationships between NNS, NN and NNGS
(N_i is one kind of neural network group)

We have \( \text{NNGS} \subseteq \text{NN} \) \hspace{2cm} (4.11)

We also have \( \text{NNGS} \preceq \text{NNS} \) \hspace{2cm} (4.12)
4.5 Neural Network Group Models

4.5-1 Neural Network Algebra Sum Groups

Neural Network Algebra Sum Groups (ASGs) are neural network groups in which each addition \( n_i + n_j \) is defined as an Algebraic Sum of every two elements \( n_i, n_j \in \mathbb{N} \).

Let \( O_i, O_j, O_k \) be the outputs of \( n_i, n_j, n_k \) respectively, \( n_i, n_j, n_k \in \mathbb{N} \).

Where:

\[
O_i = (O_{i1}, O_{i2}, O_{i3}, \ldots, O_{iq})
\]

\[
O_j = (O_{j1}, O_{j2}, O_{j3}, \ldots, O_{js})
\]

\[
O_k = (O_{k1}, O_{k2}, O_{k3}, \ldots, O_{ks})
\] (4.13)

The number of neural network outputs could either be the same or different. If \( q = s = t \) (= c), then every element of the neural network group \( n_i, n_j, n_k \) have \( c \) outputs.

If \( c = 1 \), every element of the neural network group \( n_i, n_j, n_k \) has one output.

\[
O_i = O_{i1}
\]

\[
O_j = O_{j1}
\]

\[
O_k = O_{k1}
\] (4.14)

If \( q \neq s \neq t \), the elements of the neural network group \( n_i, n_j, n_k \) have different numbers of outputs. Supposing \( q \leq s \leq t \), we will have:

\[
O_i = (O_{i1}, O_{i2}, O_{i3}, \ldots, O_{its}, \ldots, O_{iq})
\]

\[
O_j = (O_{j1}, O_{j2}, O_{j3}, \ldots, O_{jts}, 0, 0, \ldots, 0)
\]

\[
O_k = (O_{k1}, O_{k2}, O_{k3}, \ldots, O_{kts}, 0, 0, \ldots, 0)
\] (4.15)

Under the definitions of equation (4.15), if the numbers of neural network output differ, we can ensure the neural networks have the same number of outputs by inserting zeros, as indicated. Thus the networks still satisfy condition (section 4.4) and can be regarded as neural network group. The equations are showed in the below:
(± O_i) + (± O_j) + (± O_k)

= ((± O_i) + (± O_j)) + (± O_k)

= (± O_i) + ((± O_j) + (± O_k))

= ((± O_{i1}) + (± O_{j1}) + (± O_{k1})), ((± O_{i2}) + (± O_{j2}) + (± O_{k2})), ............

((± O_{i_l}) + (± O_{j_l}) + (± O_{k_l})), ((± O_{i_{l+1}}) + (± O_{j_{l+1}}) + (± O_{k_{l+1}})), ............

((± O_{i_{n1}}) + (± O_{j_{n1}}) + (± O_{k_{n1}})), ((± O_{i_{n2}}) + (± O_{j_{n2}}) + (± O_{k_{n2}}))

= (± O_{i_{n2}}) + (± O_{j_{n2}}) + (± O_{k_{n2}})

(4.16)

4.5-2 Piece-wise Function Groups (PFGs)

Piece-wise Function Groups (PFGs) can be defined in which each addition
n_i + n_j is defined as a Piece-wise Function for every two elements n_i, n_j ∈ N:

O_i + O_j = \begin{cases} O_i, A < I \leq B \\ O_j, B < I \leq C \end{cases}

(4.17)

Where I_i = I_j = I (inputs to the neural network)

(\sum (O_i) + (O_j) + (O_k) = (O_i) + (O_j) + (O_k))

= \begin{cases} O_i, A < I \leq B \\ O_j, B < I \leq C \\ O_k, A < I \leq D \end{cases}

(4.18)

Such a Piecewise Function can ensure neural networks satisfy the necessary
conditions to be regarded as a neural network group. Equation (4.18) means that
such Piecewise Function neural networks satisfy condition (b). We can
furthermore choose to build such neural networks in such a manner that they
become a generalised neural network set, which automatically satisfy conditions
(a), (c) and (d). Thus, Piecewise Functions can be used to ensure that neural networks satisfy all the necessary conditions of groups.

4.5-3 Neural Network Group Features

Hornik (1991) [35] proved the following general result:

"Whenever the activation function is continuous, bounded and nonconstant, then for an arbitrary compact subset $X \subseteq \mathbb{R}^n$, standard multilayer feedforward networks can approximate any continuous function on $X$ arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available]."

A more general result was proved by Leshno (1993):

"A standard multilayer feedforward network with a locally bounded piecewise continuous activation function can approximate any continuous function to any degree of accuracy if and only if the network’s activation function is not a polynomial]."

We proceed to use an inductive proof to show that a similar characteristic exists for artificial neural network groups:

Consider a neural network Piecewise Function Group,, in which each member is a standard multilayer feedforward neural network, and which has locally bounded, piecewise continuous (rather than polynomial) activation function and threshold. Each such group can approximate any kind of piecewise continuous function, and to any degree of accuracy.

We use the following definitions in our proof:

- K: compact set, $K \subseteq \mathbb{R}^n$;
- $N_w$: the network with $n$ input-units that is characterised by $w$;
- $C(\mathbb{R}^n)$: the family of "real world" functions one may wish to approximate with feedforward network architectures of the form $N_w$;
- $g$: every continuous function, $g \subseteq C(\mathbb{R}^n)$;
- $F$: the family of all functions implied by the network's architecture namely the family when $w$ runs over all its possible values;
• $f$: a good approximation to $g$ on $K$, $f \subset F$;

• $L^\infty_{loc}(K)$: essentially bounded on $K$ in $\mathbb{R}^n$ with respect to Lebesgue measurement;

• **PFG** (Neural Network Piecewise Function Group): the neural network group in which each addition $n_i + n_j$, is defined as the Piecewise Function for every two elements $n_i, n_j \in \mathbb{N}$:

$$O_i + O_j = \begin{cases} O_i, I_i = I \in K_1 \\ O_j, I_j = I \in K_2 \end{cases}$$

(4.19)

Where $I_i = I_j = I$ (inputs to the neural networks).

**Proof:**

Step1: $K_i = K$

Based on the Leshno (1993) [41],

$$\lim_{j \to \infty} \| g - f_j \|_{p(K)} = 0$$

(4.20)

Setp2: $K_1 \cup K_2 = K$

$$g = \begin{cases} g_1, & \text{if } i \text{ is a continuous function on } K_1 \\ g_2, & \text{if } g \text{ is a continuous function on } K_2 \end{cases}$$

(4.21)

In $K_1$, we have:

$$\lim_{j \to \infty} \| g_1 - f_j^{(1)} \|_{L^p(K_1)} = 0$$

(4.22)

There is a function $f^{(1)}$ which is a good approximation to $g_1$ on $K_1$.

In $K_2$, we have:
\[ \lim_{j \to \infty} \| g_2 - f^{(2)}_j \|_{L^e(K_2)} = 0 \]  \hspace{1cm} (4.23)

There is a function \( f^{(2)} \) which is a good approximation to \( g_2 \) on \( K_2 \).

Based on the definition of Neural Network Piecewise Function Groups,

\[
O_1 + O_2 = \begin{cases} 
O_1 = f^{(0)}_1, & I \in K_1 \\
O_2 = f^{(2)}_2, & I \in K_2
\end{cases}
\]  \hspace{1cm} (4.24)

Where \( I_1 = I_2 = 1 \) (inputs to the neural networks).

Setp3: \( K_1 \cup K_2 \cup \ldots \cup K_m = K \)

\( m \) is an any integer and \( m \to \infty \).

\[
g = \begin{cases} 
g_1, & g_1 \text{ is a continuous function on } K_1 \\
g_2, & g_2 \text{ is a continuous function on } K_2 \\
\ldots & \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\
g_m, & g_m \text{ is a continuous function on } K_m
\end{cases}
\]  \hspace{1cm} (4.25)

Based on the definition of Neural Network Piecewise Function Group, we have:

\[
O_1 + O_2 + \ldots + O_m = \begin{cases} 
O_1 = f^{(0)}_1, & I \in K_1 \\
O_2 = f^{(2)}_2, & I \in K_2 \\
\ldots & \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\
O_m = f^{(m)}_m, & I \in K_m
\end{cases}
\]  \hspace{1cm} (4.26)

Setp4: \( K_1 \cup K_2 \cup \ldots \cup K_m \cup K_{m+1} = K \)

\[
g = \begin{cases} 
g_1, & g_1 \text{ is a continuous function on } K_1 \\
g_2, & g_2 \text{ is a continuous function on } K_2 \\
\ldots & \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\
g_m, & g_m \text{ is a continuous function on } K_m \\
g_{m+1}, & g_{m+1} \text{ is a continuous function on } K_{m+1}
\end{cases}
\]  \hspace{1cm} (4.27)

In \( K_{m+1} \), based on the Leshno Theorem (1993) [25], we have:
\[
\lim_{j \to \infty} \left\| g_{m+1} - f_j^{(m+1)} \right\|_{L^2(K_{m+1})} = 0
\]

(4.28)

There is a function \( f^{(m+1)} \) which is a good approximation to \( g_{m+1} \) on \( K_{m-1} \).

Based on step 3 equation (4.27) and equation (4.28), we have:

\[
O_{1} + O_{2} + \ldots + O_{m} = \begin{cases} 
O_{1} = f^{(1)}, & I \in K_{1}, \\
O_{2} = f^{(2)}, & I \in K_{2}, \\
\ldots \ldots \ldots \\
O_{m} = f^{(m)}, & I \in K_{m}, \\
O_{m+1} = f^{(m+1)}, & I \in K_{m+1},
\end{cases}
\]

(4.29)
4.6 Using MPHONNG Models for Real Data Simulation and Estimation

4.6-1 The Formulation of MPHONNG

In order to handle real life cases of input training data, the Multi-Polynomial Higher Order Neural Network Group Model (MPHONNG) has been developed as follows:

MPHONNG is one kind of neural network group, in which each element is a Multi-Polynomial Higher Order Neural Network proposed in this thesis.

We have:

\[ \text{MPHONNG} \subset \mathbf{N} \]  \hspace{1cm} (4.30)

Where \( \text{MPHONNG} = \{ \text{model-0, model-1, model-2, \ldots} \} \).

Let us use the format that has been mentioned in the Chapter to express 4.30 as follows:

\[ \text{MPHONN} \in \text{MPHONNG} \]  \hspace{1cm} (4.31)

Where \( \text{MPHONN} = f : \mathbb{R}^n \rightarrow \mathbb{R}^m \)

In the formula (4.31), \( \text{MPHONNG} \) is a Multi-Polynomial Higher Order Neural Network Group Model, and MPHONN is an element of the MPHONNG set, which is a Multi-Polynomial Higher Order Neural Network Model.

The domain of the MPHONN inputs is the \( n \)-dimensional real number \( \mathbb{R}^n \). Likewise, the MPHONN outputs belong to the \( m \)-dimensional real number \( \mathbb{R}^m \). The neural network function \( f \) is a mapping from the inputs of MPHONN to its outputs.

4.6-2 The Structure of MPHONNG

The Backpropagation algorithm has been used in the Multi-Polynomial Higher Order Neural Network Model. There is no problem with the convergence. And based on the inference from Zhang and Fulcher (1996), such neural network group can approximate any kind of piece-wise continues function, and to any degree of accuracy. Hence, MPHONNG is able to simulate discontinuous data. We will prove these points in next Chapters.

In the following sections, we will introduce and discuss the three models of the MPHONNG program. These are the engines and the main part of the MPHONNG system. It also is my main contribution in this project. So, I will try giving about the
MPHONNG program as much detail as I think I am allowed to.

This work was supported by an USWM (University of Western Sydney Macarthur, Australia) Foundation Grant in 1999. This work also was performed while Dr. Ming Zhang held an USA National Research Council Senior Research Associateship at the National Oceanic and Atmospheric Administration (NOAA), National Environmental Satellite, Data, and Information Service (NESDIS), Office of Research and Application (ORA). As a consequence, all the software developed during the course of this thesis remains the property of these organisations and cannot be included in all details in this thesis for publication.

![Diagram of MPHONNG and MPHONNG-
1, MPHONNG-
2, ..., MPHONNG-
] Figure 4.6 Multi-Polynomial Higher Order Neural Network Group Model
Chapter 5. Multi-Polynomial Higher Order Neural Network Group Models (MPHONNG)

Multi-Polynomial Higher Order Neural Network Group models (MPHONNG) with higher accuracy has been developed. Using Sun workstation, C++, and Motif, a MPHONNG Simulator has been built as well. Real world data always cannot be simply simulated very well by single polynomial function. So the ordinary higher order neural networks could be failed to simulate such complicated real world data. But MPHONNG can simulate multi-polynomial functions, and it produces many good results through experiments, which can prove MPHONNG model can achieve more accuracy for real world data simulation, the Chapter7 and Chapter8 have been shown.

5.1 The MPHONNG Definition

The network architecture of MPHONNG consists of some MPHONN sets. A specify definition of MPHONNG is presented in the following:

\[
\text{let:} \\
\text{mphonn - sets : ms} \\
\text{MPHONNG : MS} \\
ms \in MS \\
Z = f(Z_i) = \sum_{i=1}^{n} Z_i = f_{pl}^k(x_{pl}, y_{pl})
\] (5.1)

![Diagram of MPHONNG structure](image)

**Figure 5.1** The structure of MPHONNG
5.2 The MPHONN Model

MPHONN is a multi-layer higher order neural network that consists of an input layer with input-units, and output layer with output-units, and two hidden layers consisting of intermediate processing units.

5.2.1 The MPHONN Definition

Based on derivatives of MPONNG Model, a Back Propagation leaning algorithm has been developed. It is combined the characteristics of PHONN, THONN, PT-HONN and PS-HONN.

\[ Z_i = \sum_{i,j=0}^{n} (a_{ij}(a1_{ij} x)^i(a2_{ij} y)^j) + b_{ij}sin'(b1_{ij} x)cos'(b2_{ij} y) \]

\[ + \frac{c_{ij}}{(1 + \exp(-c1_{ij} x))(1 + \exp(-c2_{ij} y))} \]  \hspace{1cm} (5.2)

while:

The coefficients: \(a1_{ij}, a2_{ij}, b1_{ij}, b2_{ij}, c1_{ij}\) and \(c2_{ij}\).

The derivatives of MPHONN model for polynomial, trigonometric polynomial neuron and sigmoid polynomial are:

(1) Polynomial function

\[ \sum_{i,j=0}^{n} (a_{ij}(a1_{ij} x)^i(a2_{ij} y)^j) \] \hspace{1cm} (5.3)

(2) Trigonometric polynomial function

\[ b_{ij}sin'(b1_{ij} x)cos'(b2_{ij} y) \] \hspace{1cm} (5.4)

(2) Sigmoid polynomial function

\[ \frac{c_{ij}}{(1 + \exp(-c1_{ij} x))(1 + \exp(-c2_{ij} y))} \] \hspace{1cm} (5.5)

Let:
net_{pi}^b: input of the neuron

x_{pi}: output of neuron

f_i^b: mapping for input to output

5.2.2 The Formula of MPHONN Model

The MPHONN Model has been set up, and it is a multilayer type of neural network model.

\[ Z = a_{o0} + a_{o1}y + a_{o2}y^2 + a_{01}x + a_{11}xy + a_{12}x^2 + a_{20}x^2 + a_{21}x^2y + a_{22}x^2y^2 + b_{00} + b_{01}\cos(y) + b_{02}\cos^2(y) + b_{10}\sin(x) + b_{11}\sin(x)\cos(y) + b_{12}\sin(x)\cos^2(y) + b_{20}\sin^2(x) + b_{21}\sin^2(x)\cos(y) + b_{22}\sin^2(x)\cos^2(y) \]

\[ + c_{00} + \frac{c_{01}}{1 + e^{-y}} + \frac{c_{02}}{(1 + e^{-y})^2} + \frac{c_{10}}{1 + e^{-x}} + \frac{c_{11}}{(1 + e^{-x})(1 + e^{-y})} + \frac{c_{12}}{(1 + e^{-x})(1 + e^{-y})^2} \]

\[ + \frac{c_{20}}{(1 + e^{-x})^2} + \frac{c_{21}}{(1 + e^{-x})^2(1 + e^{-y})} + \frac{c_{22}}{(1 + e^{-x})^2(1 + e^{-y})^2} \]

(5.6)

where \( a_y, b_y \) and \( c_y \) are the weights of the network. All the weights on the layer can be derived directly from the coefficients of the discrete analog form of the Polynomial, Trigonometric polynomial and Sigmoid polynomial.

In the following section, we show the structure of MPHONN model.

5.2.3 The Structure of MPHONN Model

The legend explanation of the below Figure 5.2

- Linear neuron : weight with value 1 (fixed)
- Polynomial neuron
- Trigonometric polynomial neuron
- Sigmoid polynomial neuron
Multiple neuron $a_{ij}$, $b_{ij}$, $c_{ij}$: trainable weight value

Figure 5.2 The structure of MPHONN
5.2.4 The Derivatives of MPHONN Model

(A) The weights on the output layer

The weights on the output layer of the MPHONN model are updated according to

$$w_{kj}(t+1) = w_{kj}(t) - \eta(\partial E / \partial w_{kj})$$  \hspace{1cm} (5.7)

Where: \(\eta\): learning-rate parameter – positive and usually less than 1

\(j\): input index \(j=1, \ldots, L\) means \(L\) input for output neuron

\(k\): the \(k\)th output neuron.

\(E\): Error

\(t\): training time

\(o\): output layer

\(p\): the \(p\)th training vector

\(w_{kj}\): weight

(B) The Output Nodes

The equations for the output nodes are:

$$net_{pk} = \sum_{j=1}^{f} w_{kj}i_{pj} + \theta_k$$  \hspace{1cm} (5.8)

$$O_{pk} = f_k(net_{pk})$$  \hspace{1cm} (5.9)

Where \(i_{pj}\): input to the output neuron (output of second hidden layer)

\(f_k\): activity function of output neuron

\(o_{pk}\): output of output neuron

\(\theta_k\): the bias term
(C) The Error at a Single Output unit

\[ \delta_{pk} = (y_{pk} - o_{pk}) \]  
(5.10)

where \( y_{pk} \): desired output value,
\( o_{pk} \): actual output from the \( k \)th unit

The total error \( E_p \) is the sum of the squares of the errors for all output units:

\[ E_p = \frac{1}{2} \sum_{k=1}^{m} \delta_{pk}^2 = \frac{1}{2} \sum_{k=1}^{m} (y_{pk} - o_{pk})^2 \]  
(5.11)

(D) The Derivatives of the MPHONN Model

The derivatives \( f_k'(\text{net}_{pk}) \) of the MPHONN model are:

** Linear function:

\[ f_k'(\text{net}_{pk}) = 1 \]  
(5.12)

** Sigmoid or logistic function:

\[ f_k'(\text{net}_{pk}) = f_k(1 - f_k) = o_{pk}(1 - o_{pk}) \]  
(5.13)

(E) The gradient of \( E_p \) -- \( \partial E_p / \partial w_{kj} \)

The gradient of \( E_p \) -- \( \partial E_p / \partial w_{kj} \) in the MPHONN model can be expressed to:

\[ \frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial o_{pk}} \frac{\partial o_{pk}}{\partial \text{net}_{pk}} \frac{\partial \text{net}_{pk}}{\partial w_{kj}} \]  
(5.14)

\[ \frac{\partial E_p}{\partial o_{pk}} = -(y_{pk} - o_{pk}) \]  
(5.15)
\[
\frac{\partial o_{p_k}}{\partial \text{net}_{p_k}} = \frac{\partial f_k}{\partial \text{net}_{p_k}} = f'_k(\text{net}_{p_k})
\]

(5.16)

\[
\frac{\partial \text{net}_{p_k}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left( \sum_{j=1}^{l} w_{ij} i_j + \theta_k \right) = i_{ij}
\]

(5.17)

Combining equations (5.14) (5.15) (5.16) and (5.17), we have for the negative gradient

\[
- \frac{\partial E_p}{\partial w_{ij}} = (y_{p_k} - o_{p_k}) f'_k(\text{net}_{p_k}) i_{ij}
\]

(5.18)

Combining equations (5.12) and (5.18), we have the negative gradient of a linear neuron as an output neuron:

\[
- \frac{\partial E_p}{\partial w_{ij}} = (y_{p_k} - o_{p_k}) i_{ij}
\]

(5.19)

Combining equations (5.13) and (5.18), we have the negative gradient of sigmoid neuron as an output neuron:

\[
- \frac{\partial E_p}{\partial w_{ij}} = (y_{p_k} - o_{p_k}) o_{p_k} (1 - o_{p_k}) i_{ij}
\]

(5.20)

(F) The equations of the weight update

*For the linear neuron as an output neuron, let:
\[
\partial_{pk}^t = (y_{pk} - o_{pk}) \tag{5.21}
\]

Combining equation (5.7), (5.19) and (5.21) we have:

\[
w_{kj}(t+1) = w_{kj}(t) + \eta \partial_{pk}^t i_{kj} \tag{5.22}
\]

*For a sigmoid neuron as an output neuron, let:

\[
\partial_{pk}^s = (y_{pk} - o_{pk}) o_{pk}(1 - o_{pk}) \tag{5.23}
\]

Combining equation (5.7), (5.20) and (5.23) we have:

\[
w_{kl}(t+1) = w_{kl}(t) + \eta \partial_{pk}^s i_{kj} \tag{5.24}
\]

The weight update in MPHONN Model (Output Layer) neuron Z:

\[
a_{00}(t+1) = a_{00}(t) + \eta (\partial_{pk}^L i_{ah00})
\]
\[
a_{01}(t+1) = a_{01}(t) + \eta (\partial_{pk}^L i_{ah01})
\]
\[
a_{02}(t+1) = a_{02}(t) + \eta (\partial_{pk}^L i_{ah02})
\]
\[
a_{10}(t+1) = a_{10}(t) + \eta (\partial_{pk}^L i_{ah10})
\]
\[
a_{11}(t+1) = a_{11}(t) + \eta (\partial_{pk}^L i_{ah11})
\]
\[
a_{12}(t+1) = a_{12}(t) + \eta (\partial_{pk}^L i_{ah12})
\]
\[
a_{20}(t+1) = a_{20}(t) + \eta (\partial_{pk}^L i_{ah20})
\]
\[
a_{21}(t+1) = a_{21}(t) + \eta (\partial_{pk}^L i_{ah21})
\]
\[
a_{22}(t+1) = a_{22}(t) + \eta (\partial_{pk}^L i_{ah22})
\]
\[
b_{00}(t+1) = b_{00}(t) + \eta (\partial_{pk}^L i_{bh00})
\]
\[
b_{01}(t+1) = b_{01}(t) + \eta (\partial_{pk}^L i_{bh01})
\]
\[ b_{02}(t + 1) = b_{02}(t) + \eta(\partial_L t_{0b02}) \]

\[ b_{10}(t + 1) = b_{10}(t) + \eta(\partial_L t_{1b10}) \]

\[ b_{11}(t + 1) = b_{11}(t) + \eta(\partial_L t_{1b11}) \]

\[ b_{12}(t + 1) = b_{12}(t) + \eta(\partial_L t_{1b12}) \]

\[ b_{20}(t + 1) = b_{20}(t) + \eta(\partial_L t_{2b20}) \]

\[ b_{21}(t + 1) = b_{21}(t) + \eta(\partial_L t_{2b21}) \]

\[ b_{22}(t + 1) = b_{22}(t) + \eta(\partial_L t_{2b22}) \]

\[ c_{00}(t + 1) = c_{00}(t) + \eta(\partial_L t_{c00}) \]

\[ c_{01}(t + 1) = c_{01}(t) + \eta(\partial_L t_{c01}) \]

\[ c_{02}(t + 1) = c_{02}(t) + \eta(\partial_L t_{c02}) \]

\[ c_{10}(t + 1) = c_{10}(t) + \eta(\partial_L t_{c10}) \]

\[ c_{11}(t + 1) = c_{11}(t) + \eta(\partial_L t_{c11}) \]

\[ c_{12}(t + 1) = c_{12}(t) + \eta(\partial_L t_{c12}) \]

\[ c_{20}(t + 1) = c_{20}(t) + \eta(\partial_L t_{c20}) \]

\[ c_{21}(t + 1) = c_{21}(t) + \eta(\partial_L t_{c21}) \]

\[ c_{22}(t + 1) = c_{22}(t) + \eta(\partial_L t_{c22}) \]

* \( i_{2hy} \): the output of second hidden layer neuron (the input to the output layer neuron)

* the output neuron is a linear neuron:
\[ \delta^l_{p_k} = (y_{pk} - o_{pk}) f'_k(n_{pk}) = (y_{pk} - o_{pk}) \]  

(5.25)

The experimental tests have shown that the MPHONN Model can get more better result than PHONN Model, THONN model and PT_HONN Model when simulated linear data and some high frequency, and high order non-linear and discontinuity data with this complex model.

(G) Artificial Neural Network Representation of derivatives

![Diagram of Artificial Neural Network](image)

**Figure 5.3 Artificial Neural network Representation of Derivatives**
Chapter 6. MPHONNG Simulator

In this chapter I will briefly introduce the interface of the MPHONNG, which includes procedures for using this program, and its package. I also will show some secessions of MPHONNG programming that I mortified.

6.1 Overview of the MPHONNG Program

In this section I will briefly introduce the MPHONNG program. The details of its operation are given in the near section where print out of the actual windows as shown.

Based on an open box approach, the MPHONNG models are developed and used within a transparency program. The MPHONNG program is written in computer C++ language, and runs under XWindows on a Sun workstation.

There are more than twelve windows and sub-windows in this visual program; therefore, it has a very high transparency. Both the system's mode and system's operation can be viewed in dynamic state, in real time, in the MPHONNG program.

The system mode is input/output data, neural network models, and coefficients/parameters, and so on.

The system operates as a general neural network system. The MPHONNG system's operation includes the following functions: loading data, generating a definition file, writing a definition file, loading neural network models, saving a report and saving coefficients.... etc.

The following Figure 6.1 shows all the windows of the MPHONNG program for data simulation & prediction.
Chapter 6. MPHONNG Simulator

In this chapter I will briefly introduce the interface of the MPHONNG, which includes procedures for using this program, and its package. I also will show some sections of MPHONNG programming that I mortified.

6.1 Overview of the MPHONNG Program

In this section I will briefly introduce the MPHONNG program. The details of its operation are given in the near section where print out of the actual windows as shown.

Based on an open box approach, the MPHONNG models are developed and used within a transparency program. The MPHONNG program is written in computer C++ language, and runs under XWindows on a Sun workstation.

There are more than twelve windows and sub-windows in this visual program; therefore, it has a very high transparency. Both the system's mode and system's operation can be viewed in dynamic state, in real time, in the MPHONNG program.

The system mode is input/output data, neural network models, and coefficients/parameters, and so on.

The system operates as a general neural network system. The MPHONNG system's operation includes the following functions: loading data, generating a definition file, writing a definition file, loading neural network models, saving a report and saving coefficients.... etc.

The following Figure 6.1 shows all the windows of the MPHONNG program for data simulation & prediction.
Figure 6.1 The windows of the MPHONNG simulation system
The 'MPHONNG Simulator' is the main window of the MPHONNG program, and is shown. Its main purpose is to display the errors graphically, and to give access to all the other windows and sub-windows, such as: operation windows, system mode windows and pop-up windows.

The operation windows display every step of the operation. They include: 'Load Data File', 'Load Neural Net', 'Generate a Definition File', 'Write Definition File', 'Save Report', and 'Save Coefficients' sub-windows. I will introduce and discuss all of these operation windows in the following section.

All the operation steps of the MPHONNG program are readily performed using a mouse. For instance, changing data, altering network models, and comparing results can be done easily and efficiently. That is because the MPHONNG program incorporates a user-friendly Graphical User Interface (GUI). It makes the MPHONNG program behave as any general 'Window' product.

The 'System mode' windows translate the 'open box' concept into fact. They let the operator view in real-time how the neural network model learns from the input training data (how the neural network model gets the values of the weights from the input training data).

When the MPHONNG program is running, the following system mode windows can be opened simultaneously from within the main window: 'Display Data (Training, Evolved, Superimposed, and Difference)', 'Show/Set Parameters', 'Network Model (with all weights)' and 'Coefficients'. Therefore, the system’s operation can be watched in every aspect.

What is enjoyable in the MPHONNG program is not only watching the mode, but also the possibility offered to modify the system's mode or change other parameters in real time.

For example, when the operator uses the 'Display Data' window to watch the input training data file, he can change the graph's format for the most suited type of display. For example, he can modify the graph's rotation, elevation, grids, smoothing, and influence, etc. all from within the 'Display Data' window.

During processing of the data, the 'Display Data' window offers four different modes to display the results, and changes them in real time along the processing: 'Training', 'Evolved', 'Superimposed', and 'Difference' with the same format as set for the input data.

If the operator is not satisfied with the results and wants a better outcome (a higher degree of accuracy of the model), she/he can stop the processing, set new values for the model's parameters, e.g. the learning rate. She/he can easily change the neural network model, too.

'Training' shows the data set that will be used for training the network.

'Evolved' shows the data set that the network produces. (It is unavailable if a network definition file has not been loaded.)

'Superimposed' shows both the training and the evolved data sets together. So you can directly compare them in the graph.

'Difference' shows the difference between the 'Training' data set and the 'Evolved'
data set.

The 'Rotation' command changes the angle of rotation at which the wire frame mesh is projected onto the screen. This allows the user to 'fly' around the wire frame surface. The default value is 30 degrees, adjustable from 0° to 355° in increments of 5° degrees, with wrap around at 360 to 0. The value can be simply adjusted with either the up/down buttons or by entering a number directly.

The 'Elevation' command changes the angle of elevation at which the wire frame mesh is projected onto the screen. This allows the user to 'fly' either above or below the wire-frame surface (usage similar to Rotation).

The 'Grids' command changes the number of wires in the wire frame meshes. It is adjustable from 6 to 30, using either the up/down buttons, or by entering a number directly. Low grid numbers allow fast display, but with a decrease in display resolution; higher numbers provide a more accurate rendition of the surface, but at the cost of increased display time.

If after the first changes, one is not yet satisfied with the results, further change can be made to the learning rate, momentum, error threshold, and random seed. One can change even the neural network model. It is also very simple to do these changes, as will be explained in the following section.

6.2 The Interface of MPHONNG simulator and procedures of operating

All the steps of the operation of the MPHONNG program are controlled from within the main window. Any step, data or calculation can be reviewed and modified from different sub-windows during the system’s operation. So, the system's operation starts with the main window.

As typical for neural network based software, the procedure of running the MPHONNG program is:

Step 1. Data pre-processing (Data encoding)

Step 2. Load data & view data

Step 3. Choose & load neural network model

Step 4. Show/Set the parameters of the network

Step 5. Run the program
Step 6. Check the results:
   If satisfactory then
   Go to step 7
   Else
   Go to Step 3

Step 7. Save & export the results.

Step 8. Data decoding.

In these seven steps, there are two basic requirements that must be satisfied before the MPHONNG program is able to start running. One is input training data, and the other one is input the network. The operator must have loaded some training data, and loaded a network.

I'd like to explain how this procedure is implemented. Figure 6.2 is a sketch map of the basic MPHONNG system’s operation.

Figure 6.2 Sketch map of the MPHONNG system’s operation

LD: Load Data-file

VD: View Data Graphically

LN: Load Network

VN: View Network’s Structure
P/S: Modify / Set Parameters

RUN: Run the System

Step 1. Data Pre-processing (Data encoding)

The neural network attempts to simulate the human brain. The human brain has front-end subsystems and can only handle preprocessed information. The front-end subsystem for humans are eyes, ears, ... . The same is also true for neural networks.

There are two types of data (languages) used by neural network systems: user application data (user data, or application data) and neural data. Neural data is used by neural networks directly. User data depends on the applications. Put it in another way: neural nets speak neural language, users speak user language.

The information processed by a neural network has to be pre-processed by a front-end subsystem. Data pre-processing is called 'data encoding', too. A neural network can not usually handle the user-application data directly.

Similarly, after neural computation, the result usually does not make sense to humans directly; the front-end system is responsible for converting the neural output data back into user-application data. This is called 'data decoding'. So, the complete neural computation process has three stages: Data Encoding, Neural Computation, and Data Decoding. Data encoding and data decoding are application-dependent, while the neural network is not application-dependent.

The following paragraphs give an example of the data pre-processing for the MPHONNG program. Data pre-processing (data decoding) transforms the existing data into another form adapted to the MPHONNG program. Typically, this is just a mathematical manipulation of the data.

Data sets are usually Pre-processed with Excel by the operator. For example, we have the data of total liabilities, extracted from the Reserve Bank of Australia Bulletin. From Jul 96 to May 97, the data shows as follows (Table 6.1):
<table>
<thead>
<tr>
<th></th>
<th>Labour Force Survey</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Labour force</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jun 96</td>
<td>9091</td>
<td>Jan 97</td>
</tr>
<tr>
<td></td>
<td>Jul</td>
<td>9109</td>
<td>Feb</td>
</tr>
<tr>
<td></td>
<td>Aug</td>
<td>9190</td>
<td>Mar</td>
</tr>
<tr>
<td></td>
<td>Sep</td>
<td>9136</td>
<td>Apr</td>
</tr>
<tr>
<td></td>
<td>Oct</td>
<td>9173</td>
<td>May</td>
</tr>
<tr>
<td></td>
<td>Nov</td>
<td>9141</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dec</td>
<td>9180</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.1** Original Data

First, we can see that all numbers are larger than 9000. In the MPHONNG function, the values of x or y are between 0 and 1. Therefore, using 10,000 to divide these data. The data set of Table 6.1, as pre-processed for the MPHONNG program, is shown in **Table 6.2** below:

<table>
<thead>
<tr>
<th></th>
<th>Labour Force Survey</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Labour force</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jun 96</td>
<td>0.9091</td>
<td>Jan 97</td>
</tr>
<tr>
<td></td>
<td>Jul</td>
<td>0.9109</td>
<td>Feb</td>
</tr>
<tr>
<td></td>
<td>Aug</td>
<td>0.9190</td>
<td>Mar</td>
</tr>
<tr>
<td></td>
<td>Sep</td>
<td>0.9136</td>
<td>Apr</td>
</tr>
<tr>
<td></td>
<td>Oct</td>
<td>0.9173</td>
<td>May</td>
</tr>
<tr>
<td></td>
<td>Nov</td>
<td>0.9141</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dec</td>
<td>0.9180</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.2** Pre-processed Data
A part of the pre-processed data will be used for neural network training. Rest of them will use for testing. The data in the Table 6.2 will be used to produce the input training data file. The input and output data will be formatted in the following form as a group of the input training data file:

<table>
<thead>
<tr>
<th>Input x</th>
<th>Input y</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9091</td>
<td>0.9109</td>
<td>0.9190</td>
</tr>
<tr>
<td>0.9109</td>
<td>0.9190</td>
<td>0.9136</td>
</tr>
<tr>
<td>0.9190</td>
<td>0.9136</td>
<td>0.9173</td>
</tr>
<tr>
<td>0.9136</td>
<td>0.9173</td>
<td>0.9141</td>
</tr>
<tr>
<td>0.9173</td>
<td>0.9141</td>
<td>0.9180</td>
</tr>
<tr>
<td>0.9141</td>
<td>0.9180</td>
<td>0.9197</td>
</tr>
<tr>
<td>0.9180</td>
<td>0.9197</td>
<td>0.9206</td>
</tr>
<tr>
<td>0.9197</td>
<td>0.9206</td>
<td>0.9186</td>
</tr>
<tr>
<td>0.9206</td>
<td>0.9186</td>
<td>0.9200</td>
</tr>
<tr>
<td>0.9186</td>
<td>0.9200</td>
<td>0.9182</td>
</tr>
</tbody>
</table>

The input training data file is now ready.

[Note]: This example has shown in detail the procedure used to prepare a data file. An original data set can be prepared into different input data sets by different operators or by the same operator at different times. For example, in this data file, you can subtract 9000, and then divided by 1000.
Step 2. Load Datafile

Open the main Windows that is shown in the below:

Figure 6.3 The MPHONNG Program Main Window
The main window is major to display the errors graphically and relevant data setup.

A Selects input data and displays data graphically.
B Creates DEF files for MPHONNG models or polynomial equations.
C Starts training the network.
D Loads DEF file into simulator, displays net or coefficients, generates a test report, changes network parameters, or sets error limit.

E Resets the network.
F Stops the network.
G Exits the simulator.
H Sets number of training iterations
I Displays number of iterations done.
J Displays current output error.
K Displays graph of error versus time.
L Scrollbar for error graph.
M Displays the number of iterations performed per screen update.
N Displays filename of the currently loaded DEF file.
O Sets the error graph to linear or logarithmic scale.
P Sets the number of iterations performed per screen update
Q Causes the error graph to automatically scale and scroll itself.
R Displays filename of the currently loaded input data file.

The main window contains three major buttons of the operation: 'Data', 'Translators' and 'Neural Network'. Each of these buttons is a pull-down menu, which allows the user to select from several options. Clicking an option creates another sub-window for further processing.

For loading a set of training data, click the 'Data' button. This is a pull-down menu. There are two options on this menu: to 'load data', or 'display data'. When pulling down this menu the first time, only the 'load data' option is highlighted. When the 'Load Data' is clicked, the standard X-view load data file dialog box - 'Load Data-file' will appear (A screen printout of this window is in Figure 3.1.3, as below). It allows the operator to move around the computer's directories easily
by double clicking on sub-directories, or by manually entering the directory path. Selecting a data file is done by clicking on its name, and then choosing the 'open' button. Data file is automatically loaded.

Figure 6.4 Load Data File Sub-Window
Otherwise, the operator can also view the loaded training data from the another option of the same drop-down menu, designated - 'Display Data' that is showed in Figure 6.5. This is a very helpful function. It will help the operator analyse the input data that was just loaded, and enable the operator to choose a suitable network model in the next step.

Figure 6.5 View Train Data from Graph Sub-Window
In this menu, the user can not only display the data graphically, but also can transform the graph (i.e. graphical rotation, elevation, grids, smoothing, influence, etc.).

**Step 3. Choose & load neural network model**

There are two ways to load a network model. One way is to load it from a network definition file (a network model should have been already stored in this network definition file). The other way is to define a new model, generate a network definition file, name it, save it, and then load it.

In the 'Translator' pull-down menu, there is a button call 'General Translator'. Click on it, the 'Generate a Definition' window appears (A figure is showed in Figure 3.1.7). Choose a neural network model, set up the parameters, and name it, save it. Then a new network definition file is generated.

![Figure 6.6 'Generate a Definition' Sub-Window](image)

Next, the operator will load it from the 'Neural Network' pull-down menu. Click
on the button of the 'Load Neural Network' in this menu and the 'Load Neural Net' window appeared (It is showed in Figure 3.1.8 in the below) The network definition files are listed in this window. Choose one and click the 'Open' button. This definition file will load in.

Figure 6.7 'Load Neural Net' Sub-Window
Step 4. Show/Sep the parameters of the network

When the operator has generated a definition file, the network’s parameters have also been set. The present step gives the operator a chance to double-check and modify these parameters. This action can be done in the ‘Show/Set Parameters’ window, which can be chosen from the ‘Neural Network’ pull-down menu.

Figure 6.8 ‘Show/Set Parameters’ Sub-Window
Step 5. Running the program

In the main window, the operator has set the 'Number of Epochs' field to 10,000. This means that the current model with the data will run for 10,000 epochs and stop.

We are now ready to run the network. Single click on the 'Run' button. The overall results are displayed directly on the main window, both as an 'error reached' figure and as a graph updated continually during processing.
Step 6. Check the results

The Final results of the relevant testing can be gathered from the ‘Save state of Neural Network’, ‘Display Network’, ‘Display Coefficients’ and ‘Generate Reprot’ windows.

The ‘Save State of Neural Network’ sub-window shows different aspects of the result such as: evolved, superimposed, and the difference between the original and the processed data sets.

![Image of the 'Save State of Neural Network' sub-window]

Figure 6.9 ‘Save State of Neural Network’ Sub-Window
* The 'Display Network' sub-window shows the values of the weights within the current neural network model, graphically.
[* The 'Display Coefficients' sub-window represents the coefficient associated with \( \sin(x) \), \( \cos(y) \), \( \sin^2(x) \), \( \cos^2(y) \), \( \frac{1}{(1+e^{-x})} \), \( \frac{1}{(1+e^{-x})^2} \), \( \frac{1}{(1+e^{-y})} \), \( \frac{1}{(1+e^{-y})^2} \).

\( x, y, x^2 \) and \( y^2 \) power indicated.]
Figure 6.11 'Display Coefficients' Sub-window
The ‘Generate Report’ sub-window gives all the details of the result in text format.

![Figure 6.12 'Generate Report' Sub-Window](image)

In briefly, all the operations above are readily performed using a mouse. Hence, changing data or network models and comparing results can be done easily and efficiently.
6.3 MPHONNG Software Package

The MPHONNG program requires XWindows 11 Release 5 or later, and Xview3.2 or later. A colour monitor is preferred, but not essential. All the various MPHONNG software modules were written in ANSI C, using standard libraries, under Solaris V2.4.

Its software package contains about thirty-seven of the program files that showed in the Figure 6.13. The eight of them are major sections. They are:

1. control.c
2. engine.c
3. graph.c
4. mesh.c
5. natneigh.c
6. netwindow.c
7. three_d.c
8. translator.c and translator2.c

![Files in the MPHONNG package](image-url)

*Figure 6.13 The files contained in MPHONNG program*
<table>
<thead>
<tr>
<th>The control.c</th>
<th>Call Xview function and creates all the windows/dialog box and initializes the structures.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The engine.c</td>
<td>MPHONNG function on the model of the simulation, the formula and the accurate of the error.</td>
</tr>
<tr>
<td>The graph.c</td>
<td>Call Xview function and graph windows such as the 'display data sub windows' and the 'display graphic of error versus time sub windows' and so on.</td>
</tr>
<tr>
<td>The mesh.c</td>
<td>Most of the code related to computing wireframe meshes is here, plus general maintenance stuff.</td>
</tr>
<tr>
<td>The netwindow.c</td>
<td>Call Xview function and creates the network model window.</td>
</tr>
<tr>
<td>The three_d.c</td>
<td>A handful of simple 3D transformations that operate on 4D vectors and 4 by 4 matrices.</td>
</tr>
<tr>
<td>The translator.c</td>
<td>Takes any polynomial, trigonometric and converts them into any one of the documented MPHONNG models. Translator will build a network specific to the polynomial and the trigonometric (exactly matching the polynomial and the trigonometric) with no redundant weight connections for polynomial and trigonometric coefficients, which are not present.</td>
</tr>
</tbody>
</table>

Page 81
Chapter 7 Experiment of MPHONNG

The experiments of the MPHONNG model will be presented in this chapter. The purpose of this experiment is to assess the accurate of the simulation results of the MPHONNG model with real-life data.

The standard of this experiment is based on the following parameters

- Model: 0
- Order: 2
- Learning Rate: 0.01
- Minimum Weight: -1
- Maximum Weight: 1
- Momentum: 0.01
- Number of Epochs: 10000
- X-axis: scale # epochs: 25
- Y-axis: Logarithmic

The resource of the real life data for training and testing has been gathered from a Reserve Bank of Australia Bulletin (RBAB), July 1999.

1) All Banks – Credit Card Lending on Total Number of Accounts 1996.6 – 1999.5 (Page S18)
2) Exchange Rate on Japanese (yen) 1996.6 – 1999.6 (Page S53)
3) Exchange Rate on United States (dollar) 1996.6 – 1999.6 (Page S53)
4) All Banks – Liabilities and Assets on Non Residents ($A) 1996.6 – 1999.5 (Page S3)
6) Share Price Indices – Australia on All Ordinaries 1996.6 – 1999.6 (Page S50)
7) Share Price Indices – United States on Dow Jones Industrial 1996.6 – 1999.6 (Page S50)

The MPHONNG models have been trained and tested with total of sixteen groups of data. All of these data were chosen from the seven major kinds of the real-life data that were extracted from the RBAB.

Each group of the real-life data has been tested. The sixteen groups of the results will be listed in the below with tables and charts.
7.1 Simulation-1

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.3436% error, see following Table 7.1 and Figure 7.1

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>Total Accounts (Real-life data)</th>
<th>MPHONNG</th>
<th>Average %</th>
<th>Error</th>
<th>Per Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/96</td>
<td>7220</td>
<td>0.47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sep/96</td>
<td>7275</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oct/96</td>
<td>7345</td>
<td>0.63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov/96</td>
<td>7396</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec/96</td>
<td>7448</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan/97</td>
<td>7445</td>
<td>0.32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb/97</td>
<td>7484</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar/97</td>
<td>7533</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apr/97</td>
<td>7548</td>
<td>0.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>May/97</td>
<td>7615</td>
<td>0.47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun/97</td>
<td>7664</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0.3436%

Table 7.1 All Banks - Credit Card Lending on Total Number of Accounts between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S18

Figure 7.1 The chart on %|Error| per pattern of the first group data
7.2 Simulation-2

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.6667% error, see following Table 7.2 and Figure 7.2.

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|---------------------------------|----------|----------------|-------------|------------|----------------|
| Aug/96     | 85.70                           | 0.88     |                |             | 0.6667%    |                |
| Sep/96     | 88.09                           | 0.57     |                |             |            |                |
| Oct/96     | 90.10                           | 1.39     |                |             |            |                |
| Nov/96     | 92.02                           | 0.25     |                |             |            |                |
| Dec/96     | 92.51                           | 0.64     |                |             |            |                |
| Jan/97     | 92.94                           | 0.50     |                |             |            |                |
| Feb/97     | 93.42                           | 1.56     |                |             |            |                |
| Mar/97     | 97.33                           | 1.12     |                |             |            |                |
| Apr/97     | 99.16                           | 0.45     |                |             |            |                |
| May/97     | 88.58                           | 0.21     |                |             |            |                |
| Jun/97     | 85.20                           | 0.16     |                |             |            |                |

Table 7.2 Exchange Rate on Japanese (yen) between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S53

Figure 7.2 The chart on %|Error| per pattern of the second group data
7.3 Simulation-3

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.4180% error, see following Table 7.3 and Figure 7.3

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|---------------------------------|---------------------------------|-----------------|-------------------------------|
| Aug/97     | 0.7344                          | 1.05                            |                 |                               |
| Sep/97     | 0.7198                          | 0.27                            |                 |                               |
| Oct/97     | 0.7036                          | 0.11                            |                 |                               |
| Nov/97     | 0.6803                          | 0.81                            |                 |                               |
| Dec/97     | 0.6527                          | 1.70                            |                 |                               |
| Jan/98     | 0.6693                          | 3.67                            |                 |                               |
| Feb/98     | 0.6745                          | 3.72                            |                 |                               |
| Mar/98     | 0.6634                          | 1.29                            |                 |                               |
| Apr/98     | 0.6499                          | 0.30                            |                 |                               |
| May/98     | 0.6236                          | 0.84                            |                 |                               |
| Jun/98     | 0.6135                          | 0.75                            |                 |                               |

Table 7.3 Exchange Rate on United States (dollar) between 1997.6 - 1998.6 – Reserve Bank of Australia Bulletin 1999.7 Page S53

Figure 7.3 The chart on %|Error| per pattern of the third group data
7.4 Simulation-4

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.9122% error, see following Table 7.4 and Figure 7.4.

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|---------------------------------|-----------------|-----------------|-----------------|-----------------|
| Aug/96     | 19178                           | 0.56             |                 |                 |                 |
| Sep/96     | 19496                           | 0.74             |                 |                 |                 |
| Oct/96     | 19467                           | 0.70             |                 |                 |                 |
| Nov/96     | 19285                           | 0.96             |                 |                 |                 |
| Dec/96     | 19617                           | 0.27             |                 |                 |                 |
| Jan/97     | 19550                           | 0.17             |                 |                 |                 |
| Feb/97     | 19738                           | 2.60             |                 |                 |                 |
| Mar/97     | 20232                           | 0.65             |                 |                 |                 |
| Apr/97     | 20730                           | 1.82             |                 |                 |                 |
| May/97     | 21151                           | 0.65             |                 |                 | 0.9122%         |

Table 7.4  All Banks - Liabilities and Assets on Non Residents of Total Liabilities between 1996.6 - 1997.5 – Reserve Bank of Australia Bulletin 1999.7 Page S3

Figure 7.4 The chart on %|Error| per pattern of the fourth group data
7.5 Simulation-5

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.8443% error, see following Table 7.5 and Figure 7.5

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|--------------------------------|---------------------|---------------------|-------------|---------------------|
| Aug/97     | 22337                          | 0.36                |                      |             | 0.8443%            |
| Sep/97     | 21959                          | 1.56                |                      |             |                      |
| Oct/97     | 22638                          | 0.98                |                      |             |                      |
| Nov/97     | 23477                          | 0.78                |                      |             |                      |
| Dec/97     | 22775                          | 0.37                |                      |             |                      |
| Jan/98     | 22907                          | 1.12                |                      |             |                      |
| Feb/98     | 23777                          | 0.84                |                      |             |                      |
| Mar/98     | 24219                          | 0.51                |                      |             |                      |
| Apr/98     | 24997                          | 0.45                |                      |             |                      |
| May/98     | 23675                          | 1.47                |                      |             |                      |

Table 7.5 All Banks - Liabilities and Assets on Non Residents of Total Liabilities between 1997.6 - 1998.5 – Reserve Bank of Australia Bulletin 1999.7 Page S3

Average 0.8443% |Error|

Figure 7.5 The chart on %|Error| per pattern of the fifth group data
7.6 Simulation-6

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.2070% error, see following Table 7.6 and Figure 7.6

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|--------------------------------|----------------|-----------------|----------------|
| Aug/97     | 7729                           | 0.14          |                 |               |
| Sep/97     | 7778                           | 0.27          |                 |               |
| Oct/97     | 7774                           | 0.38          |                 |               |
| Nov/97     | 7818                           | 0.26          |                 |               |
| Dec/97     | 7868                           | 0.30          |                 |               |
| Jan/98     | 7875                           | 0.04          |                 |               |
| Feb/98     | 7919                           | 0.31          |                 |               |
| Mar/98     | 7973                           | 0.12          |                 |               |
| Apr/98     | 7973                           | 0.07          |                 |               |
| May/98     | 8039                           | 0.09          |                 | 0.2070%       |

Table 7.6  All Banks – Credit Card Lending on total Number of Accounts between 1997.6 - 1998.5 – Reserve Bank of Australia Bulletin 1999.7 Page S18

Figure 7.6  The chart on %|Error| per pattern of the sixth group data
7.7 Simulation-7

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.7291\% error, see following Table 7.7 and Figure 7.7.

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|---------------------------------|---------|----------------|-------------|-----------|-------------|
| Aug/96     | 19178                           | 0.64    |                | 0.7291\%    |           |             |
| Sep/96     | 19496                           | 0.37    |                |             |           |             |
| Oct/96     | 19467                           | 1.23    |                |             |           |             |
| Nov/96     | 19285                           | 2.12    |                |             |           |             |
| Dec/96     | 19617                           | 0.57    |                |             |           |             |
| Jan/97     | 19550                           | 0.37    |                |             |           |             |
| Feb/97     | 19738                           | 0.94    |                |             |           |             |
| Mar/97     | 20232                           | 0.52    |                |             |           |             |
| Apr/97     | 20730                           | 0.79    |                |             |           |             |
| May/97     | 21151                           | 0.36    |                |             |           |             |
| Jun/97     | 21666                           | 0.11    |                |             |           |             |

Table 7.7 All Banks - Liabilities and Assets on Non Residents of Total Liabilities between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S3

Average 0.7291\% | Error |

Figure 7.7 The chart on %|Error| per pattern of the seventh group data
7.8 Simulation-8

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within **3.3512%** error, see following Table 7.8 and Figure 7.8

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|--------------------------------|----------------|----------------|-------------|-----------|----------------|
| Aug/96     | 9190                           | 0.60                       |               |             |           |                |
| Sep/96     | 9136                           | 1.38                       |               |             |           |                |
| Oct/96     | 9173                           | 2.26                       |               |             |           |                |
| Nov/96     | 9141                           | 3.08                       |               |             |           |                |
| Dec/96     | 9180                           | 3.43                       |               |             |           |                |
| Jan/97     | 9197                           | 0.67                       |               |             |           |                |
| Feb/97     | 9206                           | 8.80                       |               |             |           |                |
| Mar/97     | 9186                           | 1.82                       |               |             |           |                |
| Apr/97     | 9200                           | 9.55                       |               |             |           |                |
| May/97     | 9182                           | 1.92                       |               |             |           |                |
| Jun/97     | 9173                           | 4.56                       |               |             |           |                |


![Average 3.3512% | Error |](image)

**Figure 7.8** The chart on %Error per pattern of the eighth group data
7.9 Simulation-9

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 2.9532% error, see following Table 7.9 and Figure 7.9.

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|--------------------------------|-----------------|-----------------|-------------|-----------------|-----------------|
| Aug/97     | 9179                           | 5.27            |                 |             |                 |                 |
| Sep/97     | 9221                           | 0.88            |                 |             |                 |                 |
| Oct/97     | 9213                           | 4.84            |                 |             |                 |                 |
| Nov/97     | 9268                           | 1.42            |                 |             |                 |                 |
| Dec/97     | 9263                           | 0.95            |                 |             |                 |                 |
| Jan/98     | 9258                           | 5.62            |                 |             |                 |                 |
| Feb/98     | 9266                           | 1.65            |                 |             |                 |                 |
| Mar/98     | 9282                           | 1.84            |                 |             |                 |                 |
| Apr/98     | 9309                           | 4.70            |                 |             |                 |                 |
| May/98     | 9312                           | 4.29            |                 |             |                 |                 |
| Jun/98     | 9360                           | 1.30            |                 |             |                 |                 |


![Average 2.9532% Error](image)

Figure 7.9 The chart on %|Error| per pattern of the ninth group data
7.10 Simulation-10

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 2.5291% error, see following Table 7.10 and Figure 7.10.

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average %|Error| Per Pattern |
|------------|--------------------------------|------------------|------------------|------------------|
| Aug/97     | 9179                           | 2.17             |                  |
| Sep/97     | 9221                           | 0.58             |                  |
| Oct/97     | 9213                           | 0.17             |                  |
| Nov/97     | 9268                           | 5.23             |                  |
| Dec/97     | 9263                           | 2.48             |                  |
| Jan/98     | 9258                           | 6.34             |                  |
| Feb/98     | 9266                           | 3.78             |                  |
| Mar/98     | 9282                           | 0.04             |                  |
| Apr/98     | 9309                           | 2.89             |                  |
| May/98     | 9312                           | 2.16             |                  |
| Jun/98     | 9360                           | 2.01             |                  |


![Average 2.5291% Error](image)

Figure 7.10 The chart on %|Error| per pattern of the tenth group data
7.11 Simulation-11

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.9127% error, see following Table 7.11 and Figure 7.11

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|---------------------------------|---------|-----------|-------------|-----------|-----------|
| Aug/96     | 293.3                           | 0.54    |           |             | 0.9127%   |           |
| Sep/96     | 296.1                           | 0.32    |           |             |           |           |
| Oct/96     | 303.9                           | 0.81    |           |             |           |           |
| Nov/96     | 308.8                           | 0.41    |           |             |           |           |
| Dec/96     | 313.5                           | 0.04    |           |             |           |           |
| Jan/97     | 313.4                           | 0.94    |           |             |           |           |
| Feb/97     | 316.8                           | 0.75    |           |             |           |           |
| Mar/97     | 313.2                           | 3.11    |           |             |           |           |
| Apr/97     | 321.7                           | 0.31    |           |             |           |           |
| May/97     | 337.5                           | 1.99    |           |             |           |           |
| Jun/97     | 352.5                           | 0.82    |           |             |           |           |

Table 7.11 Share Price Indices – Australia on All Ordinaries between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Average 0.9127% | Error |

Figure 7.11 The chart on %|Error| per pattern of the eleventh group data
7.12 Simulation-12

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 0.8071% error, see following Table 7.12 and Figure 7.12.

| Month/Year | Total Accounts (Real-life data) | PT_HONN %|Error| Per Pattern | Average %|Error| Per Pattern |
|------------|--------------------------------|------------|-----------|--------------|-----------|------------|
| Aug/96     | 293.3                          | 1.78       |           |              |           |            |
| Sep/96     | 296.1                          | 0.22       |           |              |           |            |
| Oct/96     | 303.9                          | 0.64       |           |              |           |            |
| Nov/96     | 308.8                          | 0.25       |           |              |           |            |
| Dec/96     | 313.5                          | 0.63       |           |              |           |            |
| Jan/97     | 313.4                          | 2.16       |           |              |           |            |
| Feb/97     | 316.8                          | 1.39       |           |              |           |            |
| Mar/97     | 313.2                          | 3.32       |           |              |           |            |
| Apr/97     | 321.7                          | 0.08       |           |              |           |            |
| May/97     | 337.5                          | 2.28       |           |              |           |            |
| Jun/97     | 352.5                          | 2.03       |           |              |           |            |

Table 7.12 Share Price Indices – Australia on All Ordinaries between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Figure 7.12 The chart on |Error| per pattern of the twelfth group data
7.13 Simulation-13

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 1.6964% error, see following Table 7.13 and Figure 7.13

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|---------------------------------|----------|-------------|-------------|
| Aug/97     | 335.3                           | 5.48     |             |             |
| Sep/97     | 357.8                           | 2.12     |             |             |
| Oct/97     | 318.7                           | 0.54     |             |             |
| Nov/97     | 318.7                           | 0.78     |             |             |
| Dec/97     | 338.3                           | 0.47     |             |             |
| Jan/98     | 343.5                           | 1.34     |             |             |
| Feb/98     | 348.8                           | 0.26     |             |             |
| Mar/98     | 354.8                           | 0.97     |             |             |
| Apr/98     | 357.1                           | 4.11     |             |             |
| May/98     | 351.1                           | 2.45     |             |             |
| Jun/98     | 345.0                           | 0.14     |             |             |

1.6964%

Table 7.13 Share Price Indices – Australia on All Ordinaries between 1997.6 – 1998.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Average 1.6964% | Error |

Figure 7.13 The chart on %|Error| per pattern of the thirteenth group data
7.14 Simulation-14

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 1.4618\% error, see following Table 7.14 and Figure 7.14

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>Total Accounts (Real-life data)</th>
<th>MPHONNG</th>
<th>Average %</th>
<th>Error Per Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/96</td>
<td>436.5</td>
<td>3.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sep/96</td>
<td>457.1</td>
<td>0.61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oct/96</td>
<td>468.6</td>
<td>4.74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov/96</td>
<td>506.8</td>
<td>0.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec/96</td>
<td>501.1</td>
<td>0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan/97</td>
<td>529.5</td>
<td>0.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb/97</td>
<td>534.5</td>
<td>3.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar/97</td>
<td>511.6</td>
<td>2.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apr/97</td>
<td>544.7</td>
<td>1.43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>May/97</td>
<td>569.7</td>
<td>0.42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun/97</td>
<td>596.3</td>
<td>0.87</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Average 1.4618% | Error |

Figure 7.14 The chart on %Error per pattern of the fourteenth group data
7.15 Simulation-15

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within 1.7973% error, see following Table 7.15 and Figure 7.15.

| Month/Year | Total Accounts (Real-life data) | MPHONNG %|Error| Per Pattern | Average % |Error| Per Pattern |
|------------|-------------------------------|------------|-------------|-------------|------------|-------------|
| Aug/97     | 592.4                         | 7.61       |             |             |            |             |
| Sep/97     | 617.5                         | 0.17       |             |             |            |             |
| Oct/97     | 578.4                         | 0.43       |             |             |            |             |
| Nov/97     | 608.0                         | 0.31       |             |             |            |             |
| Dec/97     | 614.6                         | 0.75       |             |             |            |             |
| Jan/98     | 614.4                         | 0.33       |             |             |            |             |
| Feb/98     | 664.1                         | 0.36       |             |             |            |             |
| Mar/98     | 683.9                         | 2.74       |             |             |            |             |
| Apr/98     | 704.4                         | 5.13       |             |             |            |             |
| May/98     | 691.7                         | 1.47       |             |             |            |             |
| Jun/98     | 695.7                         | 0.47       |             |             |            |             |

Table 7.15 Share Price Indices – United States on Dow Jones Industrial between 1997.6 - 1998.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Figure 7.15 The chart on %|Error| per pattern of the fifteenth group data
7.16 Simulation-16

The result of the simulation has shown that the MPHONNG model converged without any difficulty of within $2.1036\%$ error, see following Table 7.16 and Figure 7.16

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>Total Accounts (Real-life data)</th>
<th>MPHONNG %</th>
<th>Error</th>
<th>Per Pattern</th>
<th>Average %</th>
<th>Error</th>
<th>Per Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/98</td>
<td>585.9</td>
<td>1.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sep/98</td>
<td>609.5</td>
<td>4.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oct/98</td>
<td>667.7</td>
<td>0.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov/98</td>
<td>708.5</td>
<td>3.47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec/98</td>
<td>713.5</td>
<td>2.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan/99</td>
<td>727.3</td>
<td>2.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb/99</td>
<td>723.3</td>
<td>0.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar/99</td>
<td>760.5</td>
<td>3.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apr/99</td>
<td>838.5</td>
<td>0.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>May/99</td>
<td>820.6</td>
<td>1.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun/99</td>
<td>852.6</td>
<td>1.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.16  *Share Price Indices – United States on Dow Jones Industrial between 1998.6 - 1999.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50*

Average 2.1036\% | Error |

Figure 7.16  The chart on %|Error| per pattern of the sixteenth group data
7.17 Analysis and Summary

The purpose of developing MPHONNG model is for increasing the ability of the MPHONN models on processing the real life data. That includes to be able to process more variety kinds of real life data and ability to get the high accurate results.

From above the experimental results, we can say that our goal has been attained. The seven kinds of the data have been chosen and the sixteen groups of experiments have been done. Through the results of these experiments, we have verified that MPHONNG models have very good characteristic. The advantages of MPHONNG will be showed in the below:

- The ability of processing variety kinds of data has been increased (improved), because MPHONNG can cover all of seven kinds of the data while it can get very good accuracy.

- The average error of MPHONNG is from \(0.2070\%\) up to \(3.3512\%\) among the sixteen group of the testing. The following chart show the average percent error of the sixteen testing:

![Chart showing the average percent error of the sixteen testing](image)

Figure 7.17 The average percent error of the sixteen testing
Chapter 8 Comparing MPHONNG with PHONN, THONN and PT-HONN

The results of comparative analysis experiment among, PHONN, THONN, PT_HONN and MPHONNG will be presented in this chapter.

This comparison will be pay particular attention on the degree of the accuracy of the error. The aim of this comparative analysis is to identify how the feature of MPHONNG is when it is compared with PHONN, THONN and PT_HONN. Because the development of MPHONNG program is based on combined the features of PHONN, THONN and PT_HONN. Therefore it is necessary to make understand what different among the new program and others.

Rules of the comparison experiments - ‘Keep compared programs in the same environment’. Therefore the comparison will be based on the following environment:

1. Same groups of testing data.
2. Same parameters of experimental environment such as learning rates and number of hidden layers and so on.

8.1 Comparison MPHONNG program with PHONN, THONN and PT_HONN

In this section, we will compare PHONN, THONN, PT_HONN and MPHONNG with the error per pattern and the average percent error per pattern. All of the comparative data come from the Chapter 7.

The specialty of PHONN, THONN, PT-HONN and MPHONNG program is to simulate the kinds of high polynomial linear and trigonometric polynomial data. The bandwidth of the kinds of data of processing focuses on those kinds of discontinued, linear and high frequency data. And MPHONNG program was developed based on the PHONN, THONN and PT_HONN. So if PHONN, THONN and PT_HONN can get good result of the simulation with some kinds of data, MPHONNG should get better. How are the results of the experiment? They will be shown in the below:
8.1-1 Comparison –1

The result of the compare has shown that the average %|Error| of the PHONN is 1.8927%, THONN is 0.9717, the PT_HONN is 0.5208% and the MPHONNG is 0.3436%, see following Table 8.1 and Figure 8.1.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|--------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Aug/96     | 7220                          | 4.28                                   | 1.77            | 1.05            | 0.47            |
| Sep/96     | 7275                          | 3.71                                   | 1.65            | 1.00            | 0.52            |
| Oct/96     | 7345                          | 3.13                                   | 1.01            | 1.00            | 0.63            |
| Nov/96     | 7396                          | 1.93                                   | 0.86            | 0.57            | 0.27            |
| Dec/96     | 7448                          | 1.04                                   | 0.20            | 0.34            | 0.58            |
| Jan/97     | 7445                          | 0.48                                   | 0.50            | 0.55            | 0.32            |
| Feb/97     | 7484                          | 0.23                                   | 0.64            | 0.19            | 0.13            |
| Mar/97     | 7533                          | 0.46                                   | 0.62            | 0.03            | 0.15            |
| Apr/97     | 7548                          | 1.64                                   | 0.46            | 0.59            | 0.21            |
| May/97     | 7615                          | 1.44                                   | 1.62            | 0.08            | 0.47            |
| Jun/97     | 7664                          | 2.46                                   | 1.37            | 0.33            | 0.04            |
| Average    |                               | 1.8927%                               | 0.9717%         | 0.5208%         | 0.3436%         |

Table 8.1 All Banks - Credit Card Lending on Total Number of Accounts between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S18

Figure 8.1 The chart on the comparison-1 average %|Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-2 Comparison – 2

The result of the compare has shown that the average \(|\text{Error}|\) of the PHONN is 2.3452%, THONN is 2.2780%, the PT_HONN is 0.9130% and the MPHONNG is 0.6667%, see following Table 8.2 and Figure 8.2.

| Month/Year | Total Accounts (Real-life data) | PHONN \(|\text{Error}|\) Per Pattern | THONN \(|\text{Error}|\) Per Pattern | PT_HONN \(|\text{Error}|\) Per Pattern | MPHONNG \(|\text{Error}|\) Per Pattern |
|------------|--------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Aug/96     | 85.70                          | 0.27                                | 1.23                                | 1.78                                | 0.88                                |
| Sep/96     | 88.09                          | 3.72                                | 3.67                                | 0.64                                | 0.57                                |
| Oct/96     | 90.10                          | 5.39                                | 5.38                                | 2.16                                | 1.39                                |
| Nov/96     | 92.02                          | 2.78                                | 1.31                                | 0.35                                | 0.25                                |
| Dec/96     | 92.51                          | 0.23                                | 0.82                                | 1.18                                | 0.64                                |
| Jan/97     | 92.94                          | 1.33                                | 0.17                                | 1.20                                | 0.50                                |
| Feb/97     | 93.42                          | 3.61                                | 3.69                                | 0.08                                | 1.56                                |
| Mar/97     | 97.33                          | 4.10                                | 4.29                                | 2.11                                | 1.12                                |
| Apr/97     | 99.16                          | 3.08                                | 1.88                                | 0.32                                | 0.45                                |
| May/97     | 88.58                          | 1.03                                | 2.06                                | 0.18                                | 0.21                                |
| Jun/97     | 85.20                          | 0.24                                | 0.55                                | 0.04                                | 0.16                                |
| Average \(|\text{Error}|\) Per Pattern | 2.3452%                          | 2.2780%                            | 0.9130%                            | 0.6667%                            |

Table 8.2 Exchange Rate on Japanese (yen) between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page 553

![Average Error Chart](image)

**Figure 8.2** The chart on the comparison-2 average \(|\text{Error}|\) of PHONN, THONN, PT_HONN and MPHONNG

Page 102
8.1-3 Comparison –3

The result of the compare has shown that the average $\%|\text{Error}|$ of the PHONN is 1.6057%, THONN is 1.6969%, PT_HONN is 1.3283% and the MPHONNG is 0.4180% see following Table 8.3 and Figure 8.3.

| Month/Year | Total Accounts (Real-life data) | PHONN $\%|\text{Error}|$ Per Pattern | THONN $\%|\text{Error}|$ Per Pattern | PT_HONN $\%|\text{Error}|$ Per Pattern | MPHONNG $\%|\text{Error}|$ Per Pattern |
|------------|--------------------------------|------------------------------------------|--------------------------------------|------------------------------------------|------------------------------------------|
| Aug/97     | 0.7344                         | 0.97                                     | 1.80                                 | 0.69                                     | 1.05                                     |
| Sep/97     | 0.7198                         | 0.08                                     | 0.42                                 | 0.03                                     | 0.27                                     |
| Oct/97     | 0.7036                         | 0.27                                     | 0.34                                 | 0.03                                     | 0.11                                     |
| Nov/97     | 0.6803                         | 1.19                                     | 1.69                                 | 0.90                                     | 0.81                                     |
| Dec/97     | 0.6527                         | 2.33                                     | 3.04                                 | 1.77                                     | 1.70                                     |
| Jan/98     | 0.6693                         | 2.84                                     | 2.48                                 | 3.57                                     | 3.67                                     |
| Feb/98     | 0.6745                         | 4.05                                     | 3.86                                 | 3.19                                     | 3.72                                     |
| Mar/98     | 0.6634                         | 1.39                                     | 0.95                                 | 0.92                                     | 1.29                                     |
| Apr/98     | 0.6499                         | 0.12                                     | 0.50                                 | 0.01                                     | 0.30                                     |
| May/98     | 0.6236                         | 2.45                                     | 2.46                                 | 2.22                                     | 0.84                                     |
| Jun/98     | 0.6135                         | 1.98                                     | 1.14                                 | 1.29                                     | 0.75                                     |
| **Average $\%|\text{Error}|$ Per Pattern** | **1.6057%** | **1.6969%** | **1.3283%** | **0.4180%** |

Table 8.3 Exchange Rate on United States (dollar) between 1997.6 - 1998.6 – Reserve Bank of Australia Bulletin 1999.7 Page S53

![Average Error Chart](image)

Figure 8.3 The chart on the comparison-3 average $\%|\text{Error}|$ of PHONN, THONN, PT_HONN and MPHONNG
8.1-4 Comparison –4

The result of the compare has shown that the average %|Error| of the PHONN is 1.9895%, THONN is 2.0976%, the PT_HONN is 1.9375% and the MPHONNG is 0.9122%, see following Table 8.4 and Figure 8.4.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>Total Accounts (Real-life data)</th>
<th>PHONN</th>
<th>THONN</th>
<th>PT_HONN</th>
<th>MPHONNG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>%</td>
<td>Error</td>
<td>%</td>
<td>Error</td>
</tr>
<tr>
<td>Aug/96</td>
<td>19178</td>
<td>1.78</td>
<td>1.62</td>
<td>1.95</td>
<td>0.56</td>
</tr>
<tr>
<td>Sep/96</td>
<td>19496</td>
<td>4.65</td>
<td>4.69</td>
<td>5.21</td>
<td>0.74</td>
</tr>
<tr>
<td>Oct/96</td>
<td>19467</td>
<td>0.99</td>
<td>1.60</td>
<td>1.03</td>
<td>0.70</td>
</tr>
<tr>
<td>Nov/96</td>
<td>19285</td>
<td>0.69</td>
<td>0.17</td>
<td>0.58</td>
<td>0.96</td>
</tr>
<tr>
<td>Dec/96</td>
<td>19617</td>
<td>1.55</td>
<td>1.76</td>
<td>2.34</td>
<td>0.27</td>
</tr>
<tr>
<td>Jan/97</td>
<td>19550</td>
<td>1.42</td>
<td>1.25</td>
<td>0.98</td>
<td>0.17</td>
</tr>
<tr>
<td>Feb/97</td>
<td>19738</td>
<td>3.64</td>
<td>3.90</td>
<td>2.58</td>
<td>2.60</td>
</tr>
<tr>
<td>Mar/97</td>
<td>20232</td>
<td>0.71</td>
<td>1.61</td>
<td>0.03</td>
<td>0.65</td>
</tr>
<tr>
<td>Apr/97</td>
<td>20730</td>
<td>2.35</td>
<td>1.34</td>
<td>2.24</td>
<td>1.82</td>
</tr>
<tr>
<td>May/97</td>
<td>21151</td>
<td>2.12</td>
<td>3.02</td>
<td>2.42</td>
<td>0.65</td>
</tr>
<tr>
<td>Average %</td>
<td>Error</td>
<td>Per Pattern</td>
<td>1.9895%</td>
<td>2.0976%</td>
<td>1.9375%</td>
</tr>
</tbody>
</table>

Table 8.4 All Banks - Liabilities and Assets on Non Residents of Total Liabilities between 1996.6 - 1997.5 – Reserve Bank of Australia Bulletin1999.7 Page S3

![Average Error Chart](image)

**Figure 8.4** The chart on the comparison-4 average %|Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-5 Comparison – 5

The result of the compare has shown that the average \(|\text{Error}|\) of the PHONN is 2.9776\%, THONN is 2.1269\%, the PT_HONN is 1.8814\% and the MPHONNG is 0.8443\%, see following Table 8.5 and Figure 8.5.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|---------------------------------|--------|-------------|-------------|--------|-------------|--------|-------------|--------|-------------|--------|-------------|
| Aug/97     | 22337                           | 0.21   | 0.22        | 0.93        | 0.36   |
| Sep/97     | 21959                           | 2.89   | 4.47        | 4.42        | 1.56   |
| Oct/97     | 22638                           | 4.02   | 2.03        | 0.73        | 0.98   |
| Nov/97     | 23477                           | 4.32   | 0.71        | 0.05        | 0.78   |
| Dec/97     | 22775                           | 0.54   | 2.00        | 1.62        | 0.37   |
| Jan/98     | 22907                           | 3.92   | 2.31        | 2.02        | 1.12   |
| Feb/98     | 23777                           | 3.34   | 1.43        | 2.65        | 0.84   |
| Mar/98     | 24219                           | 3.47   | 1.99        | 0.89        | 0.51   |
| Apr/98     | 24997                           | 6.87   | 1.71        | 2.85        | 0.45   |
| May/98     | 23675                           | 0.21   | 4.38        | 2.66        | 1.47   |
| Average %|Error| Per Pattern | 2.9776\% | 2.1269\% | 1.8814\% | 0.8443\% |

Table 8.5 All Banks - Liabilities and Assets on Non Residents of Total Liabilities between 1997.6 - 1998.5 – Reserve Bank of Australia Bulletin 1999.7 Page S3

![Average Error Chart](image)

**Figure 8.5** The chart on the comparison-5 average % \(|\text{Error}|\) of PHONN, THONN, PT_HONN and MPHONNG
8.1-6 Comparison –6

The result of the compare has shown that the average |Error| of the PHONN is 0.5314%, THONN is 1.3422%, the PT_HONN is 0.3428 % and the MPHONNG is 0.2070%, see following Table 8.6 and Figure 8.6.

| Month/Year | Total Accounts (Real-life date) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|--------|-------------|-------------|--------|-------------|-------------|--------|-------------|-------------|--------|-------------|
| Aug/97     | 7729                           | 0.43   | 2.31        | 0.23        | 0.14   |
| Sep/97     | 7778                           | 0.62   | 1.67        | 0.95        | 0.27   |
| Oct/97     | 7774                           | 0.22   | 1.65        | 0.03        | 0.38   |
| Nov/97     | 7818                           | 1.17   | 1.06        | 0.58        | 0.26   |
| Dec/97     | 7868                           | 0.24   | 0.43        | 0.44        | 0.30   |
| Jan/98     | 7875                           | 0.19   | 0.28        | 0.16        | 0.04   |
| Feb/98     | 7919                           | 0.99   | 0.32        | 0.74        | 0.31   |
| Mar/98     | 7973                           | 0.15   | 1.02        | 0.05        | 0.12   |
| Apr/98     | 7973                           | 0.53   | 1.91        | 0.18        | 0.07   |
| May/98     | 8039                           | 0.77   | 2.77        | 0.07        | 0.09   |

Average |Error| Per Pattern: 0.5314% 1.3422% 0.3428% 0.2070%

Table 8.6 All Banks – Credit Card Lending on total Number of Accounts between 1997.6 - 1998.5 – Reserve Bank of Australia Bulletin 1999.7 Page S18

Figure 8.6 The chart on the comparison-6 average |Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-7 Comparison –7

The result of the compare has shown that the average % Error of the PHONN is 1.3285%, THONN is 1.4534%, the PT_HONN is 1.1774% and the MPHONNG is 0.7291%, see following Table 8.7 and Figure 8.7.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|---------------------------------|--------|---------|-------------|--------|---------|-------------|--------|---------|---------|--------|---------|
| Aug/96     | 19178                           | 2.33   | 2.20    | 1.84        | 0.64   |        |             |         |         |         |         |         |
| Sep/96     | 19496                           | 0.99   | 0.68    | 0.75        | 0.37   |        |             |         |         |         |         |         |
| Oct/96     | 19467                           | 0.93   | 1.27    | 1.10        | 1.23   |        |             |         |         |         |         |         |
| Nov/96     | 19285                           | 3.00   | 3.03    | 2.87        | 2.12   |        |             |         |         |         |         |         |
| Dec/96     | 19617                           | 1.16   | 1.08    | 0.90        | 0.57   |        |             |         |         |         |         |         |
| Jan/97     | 19550                           | 1.08   | 1.43    | 1.26        | 0.37   |        |             |         |         |         |         |         |
| Feb/97     | 19738                           | 1.48   | 1.46    | 1.32        | 0.94   |        |             |         |         |         |         |         |
| Mar/97     | 20232                           | 0.87   | 0.69    | 0.81        | 0.52   |        |             |         |         |         |         |         |
| Apr/97     | 20730                           | 1.81   | 1.56    | 1.48        | 0.79   |        |             |         |         |         |         |         |
| May/97     | 21151                           | 0.85   | 1.18    | 0.53        | 0.36   |        |             |         |         |         |         |         |
| Jun/97     | 21666                           | 0.13   | 1.41    | 0.09        | 0.11   |        |             |         |         |         |         |         |
| Average %|Error| Per Pattern | 1.3285%  | 1.4534%  | 1.1774%  | 0.7291% |        |             |         |         |         |         |         |

Table 8.7 All Banks - Liabilities and Assets on Non Residents of Total Liabilities between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S3

Figure 8.7 The chart on the comparison-7 average % |Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-8 Comparison –8

The result of the compare has shown that the average |Error| of the PHONN is 5.8787%, THONN is 6.7638%, the PT_HONN is 5.7999% and the MPHONNG is 3.3512%, see following Table 8.8 and Figure 8.8.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|-----------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| Aug/96     | 9190                           | 1.20            | 5.44        | 0.52        | 0.60        |
| Sep/96     | 9136                           | 5.07            | 7.88        | 5.40        | 1.38        |
| Oct/96     | 9173                           | 1.26            | 6.64        | 1.89        | 2.26        |
| Nov/96     | 9141                           | 12.73           | 12.58       | 12.36       | 3.08        |
| Dec/96     | 9180                           | 0.94            | 1.54        | 0.87        | 3.43        |
| Jan/97     | 9197                           | 14.82           | 14.52       | 15.04       | 8.67        |
| Feb/97     | 9206                           | 11.69           | 12.28       | 12.13       | 8.80        |
| Mar/97     | 9186                           | 3.84            | 0.93        | 2.88        | 1.82        |
| Apr/97     | 9200                           | 4.16            | 4.22        | 5.13        | 9.55        |
| May/97     | 9182                           | 1.77            | 0.56        | 1.20        | 1.92        |
| Jun/97     | 9173                           | 7.19            | 7.81        | 6.38        | 4.56        |

Average %|Error| Per Pattern | 5.8787% | 6.7638% | 5.7999% | 3.3512%


Figure 8.8  The chart on the comparison-8 average %|Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-9 Comparison –9

The result of the compare has shown that the average %\(|\text{Error}|\) of the PHONN is 4.1013% THONN is 5.3549%, the PT_HONN is 4.0732% and the MPHONNG is 2.9532%, see following Table 8.9 and Figure 8.9.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| Aug/97     | 9179                           | 8.02                           | 10.62                          | 7.58                           | 5.27                           |
| Sep/97     | 9221                           | 1.76                           | 5.84                           | 1.93                           | 0.88                           |
| Oct/97     | 9213                           | 0.74                           | 4.21                           | 0.36                           | 4.84                           |
| Nov/97     | 9268                           | 8.97                           | 12.07                          | 8.84                           | 1.42                           |
| Dec/97     | 9263                           | 4.16                           | 2.99                           | 3.93                           | 0.95                           |
| Jan/98     | 9258                           | 7.54                           | 7.65                           | 8.15                           | 5.62                           |
| Feb/98     | 9266                           | 3.63                           | 3.42                           | 4.23                           | 1.65                           |
| Mar/98     | 9282                           | 0.71                           | 0.74                           | 0.14                           | 1.84                           |
| Apr/98     | 9309                           | 4.28                           | 3.71                           | 3.75                           | 4.70                           |
| May/98     | 9312                           | 3.10                           | 3.67                           | 3.28                           | 4.29                           |
| Jun/98     | 9360                           | 2.22                           | 3.98                           | 2.60                           | 1.30                           |
| Average %\(|\text{Error}|\) Per Pattern | 4.1013% | 5.3549% | 4.0732% | 2.9532% |

Table 5.9  

Figure 8.9  
The chart on the comparison-9 average %\(|\text{Error}|\) of PHONN, THONN, PT_HONN and MPHONNG

Page 109
8.1-10 Comparison –10

The result of the compare has shown that the average $|\text{Error}|$ of the PHONN is 4.0252%, THONN is 5.3549%, the PT_HONN is 3.9827% and the MPHONNG is 2.5291%, see following Table 8.10 and Figure 8.10.

| Month/Year | Total Accounts (Real-life data) | PHONN $|\text{Error}|$ Per Pattern | THONN $|\text{Error}|$ Per Pattern | PT_HONN $|\text{Error}|$ Per Pattern | MPHONNG $|\text{Error}|$ Per Pattern |
|------------|---------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|
| Aug/97     | 9179                            | 6.22                              | 10.62                             | 6.80                              | 2.17                              |
| Sep/97     | 9221                            | 2.30                              | 5.84                              | 1.71                              | 0.58                              |
| Oct/97     | 9213                            | 0.23                              | 4.21                              | 0.10                              | 0.17                              |
| Nov/97     | 9268                            | 8.31                              | 12.07                             | 8.54                              | 5.23                              |
| Dec/97     | 9263                            | 3.32                              | 2.99                              | 3.73                              | 2.48                              |
| Jan/98     | 9258                            | 8.67                              | 7.65                              | 8.33                              | 6.34                              |
| Feb/98     | 9266                            | 4.85                              | 3.42                              | 4.46                              | 3.78                              |
| Mar/98     | 9282                            | 0.30                              | 0.74                              | 0.04                              | 0.04                              |
| Apr/98     | 9309                            | 3.63                              | 3.71                              | 3.75                              | 2.89                              |
| May/98     | 9312                            | 2.70                              | 3.67                              | 3.10                              | 2.16                              |
| Jun/98     | 9360                            | 3.75                              | 3.98                              | 3.25                              | 2.01                              |
| **Average $|\text{Error}|$ Per Pattern** | **4.0252%** | **5.3549%** | **3.9827%** | **2.5291%** |


**Figure 8.10**  The chart on the comparison-10 average $|\text{Error}|$ of PHONN, THONN, PT_HONN and MPHONNG

Page 110
8.1-11 Comparison – 11

The result of the compare has shown that the average %|Error| of the PHONN is 1.4000%, THONN is 1.3734%, the PT_HONN is 1.2689% and the MPHONNG is 0.9127%, see following Table 8.11 and Figure 8.11.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Aug/96     | 293.3                          | 0.21            | 1.14            | 0.79            | 0.54            |
| Sep/96     | 296.1                          | 0.90            | 0.18            | 0.43            | 0.32            |
| Oct/96     | 303.9                          | 0.95            | 0.63            | 0.94            | 0.81            |
| Nov/96     | 308.8                          | 0.83            | 0.09            | 0.54            | 0.41            |
| Dec/96     | 313.5                          | 0.01            | 0.58            | 0.16            | 0.04            |
| Jan/97     | 313.4                          | 1.77            | 2.09            | 1.92            | 0.94            |
| Feb/97     | 316.8                          | 1.76            | 1.45            | 1.64            | 0.75            |
| Mar/97     | 313.2                          | 3.36            | 3.26            | 3.45            | 3.11            |
| Apr/97     | 321.7                          | 1.03            | 0.23            | 0.71            | 0.31            |
| May/97     | 337.5                          | 2.71            | 2.52            | 2.33            | 1.99            |
| Jun/97     | 352.5                          | 1.86            | 2.67            | 1.06            | 0.82            |
| Average    | 1.4000%                       | 1.3734%         | 1.2689%         | 0.9127%         |

Table 8.11  Share Price Indices – Australia on All Ordinaries between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Figure 8.11  The chart on the comparison-11 average % |Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-12 Comparison –12

The result of the compare-3 has shown that the average %|Error| of the PHONN is 1.6091%, THONN is 1.5242%, the PT_HONN is 1.3448% and the MPHONNG is 0.8073%, see following Table 8.12 and Figure 8.12.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|----------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| Aug/96     | 293.3                          | 1.48      | 3.24       | 1.78       | 0.67       |
| Sep/96     | 296.1                          | 1.21      | 2.27       | 0.22       | 0.15       |
| Oct/96     | 303.9                          | 1.08      | 0.73       | 0.64       | 0.37       |
| Nov/96     | 308.8                          | 0.47      | 0.69       | 0.25       | 0.09       |
| Dec/96     | 313.5                          | 0.40      | 0.38       | 0.63       | 0.46       |
| Jan/97     | 313.4                          | 2.13      | 1.80       | 2.16       | 0.99       |
| Feb/97     | 316.8                          | 1.88      | 0.45       | 1.39       | 0.78       |
| Mar/97     | 313.2                          | 3.57      | 2.80       | 3.32       | 2.78       |
| Apr/97     | 321.7                          | 0.96      | 1.19       | 0.08       | 0.16       |
| May/97     | 337.5                          | 2.37      | 2.20       | 2.28       | 1.64       |
| Jun/97     | 352.5                          | 2.16      | 1.00       | 2.03       | 0.79       |
| Average    |                                |           |            |            | 1.6091%    | 1.5242%    | 1.3448%    | 0.8073%    |

Table 8.12 Share Price Indices – Australia on All Ordinaries between 1996.6 - 1997.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

![Figure 8.12](image)

Figure 8.12 The chart on the comparison-12 average %|Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-13 Comparison –13

The result of the compare has shown that the average \% Error of the PHONN is 2.7315\%, THONN is 2.8255\%, the PT_HONN is 2.6404\% and the MPHONNG is 1.6963\%, see following Table 8.13 and Figure 8.13.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|------------------|------------------|------------------|------------------|------------------|
| Aug/97     | 335.3                          | 8.65             | 8.15             | 9.44             | 5.48             |
| Sep/97     | 357.8                          | 2.61             | 3.16             | 3.63             | 2.12             |
| Oct/97     | 318.7                          | 1.12             | 1.12             | 0.61             | 0.54             |
| Nov/97     | 318.7                          | 2.74             | 2.39             | 1.52             | 0.78             |
| Dec/97     | 338.3                          | 1.42             | 0.16             | 0.63             | 0.47             |
| Jan/98     | 343.5                          | 1.79             | 2.26             | 1.63             | 1.34             |
| Feb/98     | 348.8                          | 0.18             | 0.53             | 0.05             | 0.26             |
| Mar/98     | 354.8                          | 1.82             | 2.58             | 2.32             | 0.97             |
| Apr/98     | 357.1                          | 5.26             | 4.44             | 5.79             | 4.11             |
| May/98     | 351.1                          | 3.69             | 6.02             | 2.62             | 2.45             |
| Jun/98     | 345.0                          | 0.75             | 0.27             | 0.80             | 0.14             |
| Average %|Error| Per Pattern | 2.7315\%          | 2.8255\%          | 2.6404\%          | 1.6963\%          |

Table 8.13 Share Price Indices – Australia on All Ordinaries between 1997.6 - 1998.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

![Average Error Chart](chart.png)

Figure 8.13 The chart on the comparison-13 average % |Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-14 Comparison –14

The result of the compare has shown that the average %|Error| of the PHONN is 2.8190%, THONN is 3.1090%, the PT_HONN is 2.7167% and the MPHONNG is 1.4618%, see following Table 8.14 and Figure 8.14.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|---------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Aug/96     | 436.5                           | 5.47            | 6.55            | 5.51            | 3.45            |
| Sep/96     | 457.1                           | 1.61            | 1.37            | 0.76            | 0.61            |
| Oct/96     | 468.6                           | 5.64            | 6.15            | 5.54            | 4.74            |
| Nov/96     | 506.8                           | 0.41            | 2.36            | 0.21            | 0.31            |
| Dec/96     | 501.1                           | 1.21            | 0.04            | 1.21            | 0.08            |
| Jan/97     | 529.5                           | 1.43            | 0.78            | 1.14            | 0.31            |
| Feb/97     | 534.5                           | 5.17            | 6.19            | 5.54            | 3.75            |
| Mar/97     | 511.6                           | 4.53            | 3.66            | 4.37            | 2.11            |
| Apr/97     | 544.7                           | 2.90            | 2.76            | 2.87            | 1.43            |
| May/97     | 569.7                           | 1.38            | 2.24            | 1.67            | 0.42            |
| Jun/97     | 596.3                           | 1.27            | 2.10            | 1.06            | 0.87            |
| Average % |Error| Per Pattern | 2.8190%         | 3.1090%         | 2.7167%         | 1.4618%         |


![Average Error Chart](image)

**Figure 8.14** The chart on the comparison-14 average %|Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-15 Comparison—15

The result of the compare has shown that the average %Error of the PHONN is 3.2577%, THONN is 4.0028%, the PT_HONN is 3.1933% and the MPHONNG is 1.7973%, see following Table 8.15 and Figure 8.15.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>Total Accounts (Real-life data)</th>
<th>PHONN %Error Per Pattern</th>
<th>THONN %Error Per Pattern</th>
<th>PT_HONN %Error Per Pattern</th>
<th>MPHONNG %Error Per Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/97</td>
<td>592.4</td>
<td>13.37</td>
<td>15.04</td>
<td>13.77</td>
<td>7.61</td>
</tr>
<tr>
<td>Sep/97</td>
<td>617.5</td>
<td>0.12</td>
<td>0.41</td>
<td>0.41</td>
<td>0.17</td>
</tr>
<tr>
<td>Oct/97</td>
<td>578.4</td>
<td>0.62</td>
<td>5.15</td>
<td>0.49</td>
<td>0.43</td>
</tr>
<tr>
<td>Nov/97</td>
<td>608.0</td>
<td>1.24</td>
<td>2.49</td>
<td>0.65</td>
<td>0.31</td>
</tr>
<tr>
<td>Dec/97</td>
<td>614.6</td>
<td>0.88</td>
<td>1.97</td>
<td>1.21</td>
<td>0.75</td>
</tr>
<tr>
<td>Jan/98</td>
<td>614.4</td>
<td>0.88</td>
<td>1.00</td>
<td>0.54</td>
<td>0.33</td>
</tr>
<tr>
<td>Feb/98</td>
<td>664.1</td>
<td>1.12</td>
<td>2.94</td>
<td>1.42</td>
<td>0.36</td>
</tr>
<tr>
<td>Mar/98</td>
<td>683.9</td>
<td>3.91</td>
<td>1.83</td>
<td>3.59</td>
<td>2.74</td>
</tr>
<tr>
<td>Apr/98</td>
<td>704.4</td>
<td>8.47</td>
<td>7.16</td>
<td>8.29</td>
<td>5.13</td>
</tr>
<tr>
<td>May/98</td>
<td>691.7</td>
<td>4.32</td>
<td>2.09</td>
<td>4.14</td>
<td>1.47</td>
</tr>
<tr>
<td>Jun/98</td>
<td>695.7</td>
<td>0.89</td>
<td>3.96</td>
<td>0.61</td>
<td>0.47</td>
</tr>
<tr>
<td>Average % Error Per Pattern</td>
<td>3.2577%</td>
<td>4.0028%</td>
<td>3.1933%</td>
<td>1.7973%</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.15 Share Price Indices – United States on Dow Jones Industrial between 1997.6 - 1998.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Figure 8.15 The chart on the comparison-1 average %|Error| of PHONN, THONN, PT_HONN and MPHONNG
8.1-16 Comparison –16

The result of the compare has shown that the average %|Error| of the PHONN is 3.5730%, THONN is 3.5755%, the PT_HONN is 3.4853% and the MPHONNG is 2.1036%, see following Table 8.16 and Figure 8.16.

| Month/Year | Total Accounts (Real-life data) | PHONN %|Error| Per Pattern | THONN %|Error| Per Pattern | PT_HONN %|Error| Per Pattern | MPHONNG %|Error| Per Pattern |
|------------|--------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Aug/98     | 585.9                          | 0.48            | 0.25            | 0.09            | 1.02            |
| Sep/98     | 609.5                          | 7.68            | 7.22            | 7.93            | 4.34            |
| Oct/98     | 667.7                          | 1.58            | 0.93            | 0.89            | 0.78            |
| Nov/98     | 708.5                          | 4.27            | 4.09            | 4.68            | 3.47            |
| Dec/98     | 713.5                          | 5.19            | 4.90            | 4.67            | 2.31            |
| Jan/99     | 727.3                          | 4.71            | 5.97            | 4.65            | 2.42            |
| Feb/99     | 723.3                          | 1.59            | 2.07            | 1.95            | 0.97            |
| Mar/99     | 760.5                          | 5.89            | 5.57            | 6.30            | 3.98            |
| Apr/99     | 838.5                          | 1.76            | 2.10            | 1.20            | 0.64            |
| May/99     | 820.6                          | 2.79            | 3.08            | 2.50            | 1.45            |
| Jun/99     | 852.6                          | 3.37            | 3.07            | 3.47            | 1.77            |
| Average %|Error| Per Pattern | 3.5730% | 3.5755% | 3.4853% | 2.1036% |

Table 8.16  Share Price Indices – United States on Dow Jones Industrial between 1998.6 - 1999.6 – Reserve Bank of Australia Bulletin 1999.7 Page S50

Average %|Error|

PHONN
THONN
PT_HONN
MPHONNG

Figure 8.16  The chart on the comparison-16 average % |Error| of PHONN, THONN, PT_HONN and MPHONNG
8.2 Summary

All of the average % Error per pattern of PHONN, THONN, PT_HONN and MPHONNG in each the compare will be gathered and listed in the Table 8.17 and the Figure 8.17-1 and Figure 8.17-2:

| Compare  | PHONN (Ave %|Error) | THONN (Ave %|Error) | PT_HONN (Ave %|Error) | MPHONNG (Ave %|Error) |
|----------|-------------|-------------|-------------|-------------|-------------|
| Compare-1| 1.8927%     | 0.9717%     | 0.5208%     | 0.3436%     |
| Compare-2| 2.3452%     | 2.2780%     | 0.9130%     | 0.6667%     |
| Compare-3| 1.6057%     | 1.6969%     | 1.3283%     | 0.4180%     |
| Compare-4| 1.9895%     | 2.0976%     | 1.9375%     | 0.9122%     |
| Compare-5| 2.9776%     | 2.1269%     | 1.8814%     | 0.8443%     |
| Compare-6| 0.5314%     | 1.3422%     | 0.3428%     | 0.2070%     |
| Compare-7| 1.3285%     | 1.4534%     | 1.1774%     | 0.7291%     |
| Compare-8| 5.8787%     | 6.7638%     | 5.7999%     | 3.3512%     |
| Compare-9| 4.1013%     | 5.3549%     | 4.0732%     | 2.9532%     |
| Compare-10| 4.0252%    | 5.3549%     | 3.9827%     | 2.5291%     |
| Compare-11| 1.4000%    | 1.3734%     | 1.2689%     | 0.9127%     |
| Compare-12| 1.6091%    | 1.5242%     | 1.3448%     | 0.8073%     |
| Compare-13| 2.7315%    | 2.8255%     | 2.6404%     | 1.6963%     |
| Compare-14| 2.8190%    | 3.1090%     | 2.7167%     | 1.4618%     |
| Compare-15| 3.2577%    | 4.0028%     | 3.1933%     | 1.7973%     |
| Compare-16| 3.5730%    | 3.5755%     | 3.4853%     | 2.1036%     |

Table 5.17 The Sum of Comparative Analysis experiment on the average % Error
Figure 8.17-1  The comparison average % Error of PHONN, THONN, PT_HONN and MPHONNG
Figure 8.17-2  The comparison average % Error of PHONN, THONN, PT_HONN and MPHONNG

Look at the Table 8.17 and the Figure 8.17-1, Figure 8.17-2 the results show MPHONNG program is better than PHONN, THONN and PT_HONN.
Chapter 9 Using MPHONNG Model for estimating heavy rainfall from satellite data

9.1 Introduction

Currently, satellite derived precipitation estimates [1] and 3 hour precipitation outlooks for convective systems, extra-tropical cyclones, and tropical cyclones are computed on the NOAA/NESDIS Interactive Flash Flood Analyser (IFFA) system and transmitted to National Weather Service Forecast Offices, and River Forecast Centres. However, this system permits the computation of rainfall estimates for only one convective system and is very time consuming. This is due to the considerable time needed for image processing, interpretation, and the computation involved in the estimation of rainfall. If there were several storms occurring, an automatic estimation technique would be useful in providing rainfall estimates for the entire country. But the classic methods cannot give good results so far. Artificial Intelligent (AI) is another possible way to solve this problem. Some artificial intelligence system for weather forecasting are designed to be objective and automated, some are designed to augment human skill. In Knowledge Augmented Severe Storms Predictor (KASSPr), knowledge was elicited in a series of interviews and exchanges of documentation between the developer and an expert in severe weather forecasting. But so far no AI system can solve these problems very well.

Artificial Neural Network (ANN) computing is an area that is receiving increased research interest. Since Richard Lippmann's [121] tutorial article "An Introduction to Computing with Neural Networks", Lippmann's article becomes one of the most widely referenced papers in the neural network literature. In Lippmann's [121] article, Multi-Layer Perceptron (MLP) neural network has been introduced. So far, the MLP is still the most widely used neural network in the world. But no one had very good results for rainfall estimation only using simple neural network models.

Artificial neuron network-based models are not yet sufficiently powerful to characterise complex systems. Moreover, a gap exists in the research literature between complex systems and general systems. To characterise complex systems, neuron-based and neural network-based group models are studied. Lumer [130] proposed a new mechanism of selective attention among perceptual groups as part of his computational model of early vision. Lie Groups were used in Tsao's [122] group theory approach to the computer simulation of 3D rigid motion.

So far there have been limited studies with emphasis on setting a few free parameters in the activation function. Vecchi et al studied the properties of a
Feedforward Neural Network (FNN) which is able to adapt its activation function. In Chen and Chang’s paper, real variables \( a \) (gain) and \( b \) (slope) in the generalised sigmoid activation function are adjusted during the learning process. In Campolucci, a neuron-adaptive activation function built as a piecewise approximation with suitable cubic splines can have arbitrary shape and this enables reduction of the overall size of the neural networks, trading connection complexity with activation function complexity. Neural networks with a neuron-adaptive activation function seem to provide better fitting properties than classical architectures with fixed activation function neurones.

Assessment of global climate change is a very important research area for the future of humans and their environment. Rainfall estimation is a key parameter in this research. During the past 20 years, there has been a great increase in our understanding of how satellite data can be used to estimate rainfall. But, even with the use of interactive computer systems, the time needed to prepare estimates of rainfall is very time consuming (about a half-hour). Verification results show that the average error for an event is about 30% [1.4. 119] An automatic estimation technique with faster speed and more accuracy are the issues the experts want to solve.

### 9.2 ANSER- plus system

The architecture of the ANSER-plus system consists of three components: (1) USER System (which comprises several IBM PC computers); (2) TRAINING System (which uses a NCCF HDS 9000 mainframe computer); (3) CENTER System (which uses a SUN Ultra 10 workstation). The USER, TRAINING and CENTER Systems communicate with each other over an Ethernet LAN.

The output functions of ANSER can be either: (1) fixed-point rainfall estimates; (2) maximum isohyet area rainfall estimates; and/or (3) total aerial rainfall estimates. All three outputs can be saved to files, as well as being displayed on a colour screen.

A graphic interface has been designed for the ANSER-plus software using Motif. The components of Graphic interface are as given in Figure 9.1.

### 9.3 Estimating with MPHONNG and comparison results

Using the ANSER-plus system with the MPHONNG model has tested the heavy convective rainfall estimating. The Table 9.1 has been shown the results from X/S (Xie/Scofield’s study), MLP (Multi Layer Perceptron as basic reasoning neural network), NANN (Neuron-Adaptive Neural Networks) and MPHONNG.
Figure 9.1  Graphic User Interface of ANSER-Plus Expert System
<table>
<thead>
<tr>
<th>No.</th>
<th>Date</th>
<th>Location</th>
<th>Observation (Inch)</th>
<th>X/S</th>
<th>MLP</th>
<th>NANN</th>
<th>MPHONNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 #</td>
<td>07/19/85</td>
<td>NY</td>
<td>6.0</td>
<td>0.83</td>
<td>14.0</td>
<td>5.84</td>
<td>2.21</td>
</tr>
<tr>
<td>2 *</td>
<td>05/16/86</td>
<td>IL, MO</td>
<td>7.0</td>
<td>10.0</td>
<td>5.52</td>
<td>4.28</td>
<td>1.03</td>
</tr>
<tr>
<td>3 *</td>
<td>08/04/82</td>
<td>WI</td>
<td>5.3</td>
<td>62.08</td>
<td>8.30</td>
<td>3.27</td>
<td>0.86</td>
</tr>
<tr>
<td>4 *</td>
<td>05/03/87</td>
<td>MO</td>
<td>7.0</td>
<td>47.86</td>
<td>10.39</td>
<td>5.26</td>
<td>3.67</td>
</tr>
<tr>
<td>5 *</td>
<td>05/26/87</td>
<td>IA</td>
<td>5.8</td>
<td>18.62</td>
<td>7.08</td>
<td>2.89</td>
<td>1.47</td>
</tr>
<tr>
<td>6 #</td>
<td>08/13/87</td>
<td>KS</td>
<td>12.2</td>
<td>16.62</td>
<td>15.25</td>
<td>7.28</td>
<td>5.84</td>
</tr>
<tr>
<td>7 #</td>
<td>08/12/87</td>
<td>KS</td>
<td>8.7</td>
<td>30.57</td>
<td>21.0</td>
<td>16.32</td>
<td>9.16</td>
</tr>
<tr>
<td>8 #</td>
<td>07/19/85</td>
<td>IA</td>
<td>9.5</td>
<td>96.21</td>
<td>18.2</td>
<td>17.28</td>
<td>9.48</td>
</tr>
<tr>
<td>9 #</td>
<td>08/22/85</td>
<td>KS</td>
<td>4.2</td>
<td>22.86</td>
<td>13.3</td>
<td>10.81</td>
<td>3.79</td>
</tr>
<tr>
<td>10 #</td>
<td>07/02/83</td>
<td>IL</td>
<td>5.2</td>
<td>13.65</td>
<td>18.1</td>
<td>12.24</td>
<td>6.32</td>
</tr>
<tr>
<td>11 *</td>
<td>05/01/83</td>
<td>MO</td>
<td>6.0</td>
<td>31.17</td>
<td>4.0</td>
<td>2.62</td>
<td>0.52</td>
</tr>
<tr>
<td>12 #</td>
<td>05/27/85</td>
<td>MO</td>
<td>4.2</td>
<td>2.62</td>
<td>18.5</td>
<td>12.17</td>
<td>5.37</td>
</tr>
<tr>
<td>13 *</td>
<td>09/05/85</td>
<td>KS</td>
<td>6.2</td>
<td>5.16</td>
<td>1.90</td>
<td>0.78</td>
<td>0.39</td>
</tr>
<tr>
<td>14 *</td>
<td>07/15/84</td>
<td>IA</td>
<td>5.0</td>
<td>78.4</td>
<td>8.80</td>
<td>6.20</td>
<td>4.79</td>
</tr>
<tr>
<td>15 *</td>
<td>07/16/84</td>
<td>TN</td>
<td>4.2</td>
<td>19.52</td>
<td>6.43</td>
<td>3.24</td>
<td>2.17</td>
</tr>
</tbody>
</table>

**Table 9.1** The results from X/S, MLP, NANN and MPHONNG

X/S: Xie/Scofield's study;
MLP: Multi Layer Perceptron as basic reasoning neural network
NANN: Neuron-Adaptive Neural Networks used in ANSER system
MPHONNG: Multi-Polynomial Higher Order Neural Network Group Models used in ANSER system
*: Training cases for training neural network(s);
#: Test cases.

The **Figure 9.2** has been results of the comparison average % |Error| of X/S, MLP, NANN and MPHONNG.
Figure 9.2 The comparison average % |Error| of X/S, MLP, NANN and M-CONNG
Chapter 10 Conclusion

I have introduced the Multi-Polynomial Higher Order Neural Network Models (MPHONNG) in this thesis, together with their applications in the financial data and rainfall estimation and prediction. Within the group the theory of neural network, I have developed the MPHONNG model. The MPHONNG is combined with characteristics of Polynomial function \[ \sum_{i,j=0}^{n} (a_{y}^{1}(a_{1_{y}}x)^{i}(a_{2_{y}}y)^{j}) \], Trigonometric polynomial function \[ b_{y}Sin^{i}(b_{1_{y}}x)Cos^{j}(b_{2_{y}}y) \] and Sigmoid polynomial function \[ \frac{c_{y}}{(1 + \exp(-c_{1_{y}}x))^{i}(1 + \exp(-c_{2_{y}}y))^{j}} \]. The models are constructed with three layers Multi-Polynomial Higher Order Neural Network. The weights of the MPHONNG models are derived directly from the coefficients of the Polynomial form, Trigonometric polynomial form and Sigmoid polynomial form.

The MPHONNG program has a user-friendly Graphical User Interface (GUI), as in most windows 98 programs, the operation of the MPHONNG program can be controlled easily with a mouse. The program enables the user to monitor on the screen, in real time its operation and results, as they progress. If desired, the program can be interrupted at any stage, and both input data and network model can be modified.

The program is developed as a visual system and it has very high transparency. One can view the input data file in both numeral and graphical form. One can also view the neural network structure with all its parameters and weights, as they evolve, in real time. All the operation steps are visible too.

The results of the experiments using real-life data show that the simulation and prediction accuracy of error is satisfactory. A comparative analysis of data processing by MPHONNG with PHONN, THONN, PT_HONN and others programs has proven that the MPHONNG program more accurately.
10.1 Original Contribution

The original contributions of the thesis in the field of the neural networks for financial data simulation and rainfall estimation are:

1. On neural networks:
   
a) Using Multi-Polynomial Higher Order Neural Network Group Models, which have been developed and theoretically analysed.

   b) The thesis is the first attempt to use Multi-Polynomial Higher Neural Network Group Models for finance data simulation and prediction, and rainfall estimation and prediction.

   c) Use of group theory in the Multi-Polynomial Higher Order Neural Network Group Models.

   d) Combining MPHONNG with ANSER-PLUS EXPERT SYSTEM for rainfall estimation and prediction, making results visible directly.

2. On system architecture:

   a) A complete new computer program has been developed. It is called MPHONNG, for applying Multi-Polynomial Higher Order Neural Network Group Models for financial data simulation and prediction, and rainfall estimation.

   b) The program has been developed in a user-friendly GUI system, easily controlled with drop-down menus and the mouse. Both system operation and system mode can be viewed during the process of data simulation and estimation.

3. On testing results:

   a) In the financial data area, MPHONNG program has been tested by using real life data extracted from Reserve Bank of Australia Bulletin. The average error of the simulation is from 0.2070% up to 3.3512%.

   b) In the rainfall data area, MPHONNG program Combining with ANSER-PLUS EXPERT SYSTEM for rainfall has been tested by using satellite picture data from NOAA/NESDIS/SAL-USA. The average error of the simulation is only about 1.86% * and 6.02% #.

Page 126
10.2 Future Research

Predication and Control of Power Factor of an Electric Power system by Using Neural Network Models

With the growing worldwide demand for electric power and the rising concern about global warming, the efficient utilisation of electric energy becomes a topic that attracts increasing interest. Power Factor - the ratio of active delivered electric power to the total voltage-ampere capacity - is one of the most important quality parameters for an electric power system. Inductance and capacitance of the power-transmission lines and the electric loads (eg induction motor drives for various industrial applications) have the effect of varying the Power Factor on power system as the current varies. When the inductance and capacitance is not well balanced, the Power Factor of the power system will be low. For a lower Power Factor, the amount of electric power delivered for a given voltage and current is smaller. To bring the Power Factor as nearly as possible to 1 to obtain an efficient power transmission and utilisation, several kinds of devices are used on the conditions of power system operation. The devices include capacitors and three-phase synchronous motors (called synchronous condensers), both of which vary the effective amount of inductance and capacitance of the power system. The implementation of inserting or removing these regulation devices involves fertile experience on the power system and complex human operations.

Neural networks have been applied to many tasks that are easy for humans to accomplish, but difficult for traditional computers. This prompts us a measure to adjust the power factor of the electric power system because of the similar non-linear operation characteristics. The objective of this project is to develop a methodology to predict and control Power Factor of an electric power system by using neural network modelling. My tasks include understanding modern control technologies, development of an appropriate artificial neural network model, test of the model and application on the power factor control.

I realised that the advanced modelling methodology is based on input-output data, exploits a polynomial artificial neural network architecture, and then control the objective via a hierarchical fuzzy logic based approach. Neural networks are superior in recognising patterns in extremely large data sets. Because neural networks have the ability to learn from a set of examples and generalise this knowledge to new situations, they are excellent for work requiring adaptive control systems. This is the feature required for the control of electric power system.
The next task is to apply the developed model for the prediction and control of Power Factor of an electric power system. This will involve collection of input data including Power Factors varying with time, electric loads, transmission lines and the power range of the system, development of control strategies, extensive computer programs and experiments etc. I may extend this work to pursue my future work.
References


22. Ming Zhang, and John Fulcher (1996), Neural Network Group Models For Financial Data Simulation, World Congress on Neural Networks, San Diego, California, USA.
23. Marwan Jabri, Ed Tinker and Laurens Leerink (1995), MUME - A Multi-
Modules Multi-Algorithms Neural Networks Simulation Environment.

24. Smieja F. J. and H. Muhlenbein (1992), Reflective Modular Neural Networks
Systems, Technical Report, German National Research Center for Computer
Science (GMD).

Neural Networks, Proc. Int. Joint Conf. Neural Networks, Washington, DC, June,
1989, pp.251-258.

Synthetic, Proceedings NIPS’92.

27. C.C. Ching, H.C. Fu (1992), A fast Learning Multilayer Neural Network Model
and Its Array Processor Implementation, Journal of Information Science &

28. G. Carpenter, S. Grossberg, Absolutely Stable Learning of Recognition Codes
by A Self-Organizing Neural Network, in: J. Denkert(Ed.),AIP Conf. Proc.151:
Neural Network for Computing, American Institute of Physics, New York, 1986,
pp. 77-85.

29. M. Cohen, S. Grossberg, Absolute Stabilitu of Global Pattern and Parallel
Memory Storage by Competitive Neural Networks IEEE Trans. Systems Man

30. S.E. Fahlman, C Lebiere, The Cascade Correlation Learning Architecture,
Carnegie Mellon University, School Computer Science Report CMU-CS-90-100,
1990.

31. S. Grossberg, Adaptive Pattern Classification and Universal Recording: I.
Parallel Development and coding of Neural Detectors, Biol. Cybernet. 23 (1976)
121-134.

32. S. Grossberg, Some Nonlinear Networks Capable of Learning A Spatial Pattern

33. R. Hecht-Nielsen, Neural Networks Nearest Matched-Fitter Classification of

34. J. Hopfield, Neurons with graded response have collective computational
3088-3092.


54. L. Bochereau, P. Bourgine, Rule \textit{extraction and Validity Domain on a Multilayer Neural Network, IJCNN, pp. 97-100, 1990.}


69. G. Mani, An Analysis of Neural Network Forecasts from a Large Scale, Real-World Stock Selection System, Proceedings of the IEEE/IAFE'95 Computational Intelligence for Financial Engineering (CIFEr), pp. 72-78, April 9-11, 1995, New York City, USA.

70. B. Lehman, A Multinomial Characterization of Feedforward Neural Networks, Proceedings of the IEEE/IAFE'95 Computational Intelligence for Financial Engineering (CIFEr), pp. 79-86, April 9-11, 1995, New York City, USA.


84. R. Salomon, Accelerating Backpropagation through Dynamic Self-Adaptation, Neural Networks, Vol. 9, No. 4, pp. 589-602, 1996.


94. J. Sima, Backpropagation is not Efficient, Neural Networks, Vol. 9, Mo. 6, pp. 1017-1024, 1996.


0-33-61832-7.


121. D. R. Lippmann, An introduction to computing with neural nets, IEEE


128. Ming Zhang, and John Fulcher, Neural Network Group Models For Financial Data Simulation, World Congress On Neural Networks, San Diego, California, USA, 1996, 910-913.


Appendix A:

Papers Published and Accepted in International Conferences


Rainfall Estimation Using M-PHONN Model*

Hui Qi
University of Western Sydney, Campbelltown, NSW 2560, Australia
Email: h.qi@uws.edu.au

Ming Zhang
Christopher Newport University, Newport News, VA 23606, USA
Email: mzhang@pcs.cnu.edu

Roderick A. Scofield
NOAA/NESDIS/SAL, 5200 Auth Road, Room 601, Washington DC, USA
Email: Roderick.Scofield@noaa.gov

* This work was supported by an USWM (University of Western Sydney Macarthur, Australia) Foundation Grant in 2000. This work was performed while Dr. Ming Zhang held a USA National Research Council Senior Research Associateship at the National Oceanic and Atmospheric Administration (NOAA), National Environmental Satellite, Data, and Information Service (NESDIS), Office of Research and applications (ORA).

1 INTRODUCTION

Currently, satellite derived precipitation estimates [1] and 3 hour precipitation outlooks for convective systems, extratropical cyclones, and tropical cyclones are computed on the NOAA/NESDIS Interactive Flash Flood Analyser (IFFA) system and transmitted to National Weather Service Forecast Offices, and River Forecast Centres. However, this system permits the computation of rainfall estimates for only one convective system and is very time consuming. This is due to the considerable time needed for image processing, interpretation, and the computation involved in the estimation of rainfall. If there were several storms occurring, an estimation technique would be useful in providing rainfall estimates for the entire country. Automatic algorithms such as the Auto-Estimators (Vicente, Scofield, and Menzel, 1998 [2]) have been accepted for operational implementation (for convection only) into National Weather Service flash flood operations. However, initial research into Artificial Intelligence

ABSTRACT

Multi-Polynomial High Order Neural Network (M-PHONN) model has been developed in this paper. The M-PHONN model for estimating heavy convective rainfall from satellite data has been tested as well. The M-PHONN model has 5% to 15% more accuracy than PT-HONN (Polynomial and Trigonometric Polynomial model) and PHONN (Polynomial Higher Order Neural Network) Models. Using ANSER-plus expert system, the average rainfall estimate errors for the total precipitation event could be reduced to less than 20%.
(AI) (Zhang and Scofield, 1994, [8]) have shown promise as yet another way to help solve this all important flash rainfall estimation problem. Some artificial intelligence system for weather forecasting are designed to be objective and automated, some are designed to augment human skill. In the Knowledge Augmented Severe Storms Predictor (KASSPr), “knowledge” was elicited in a series of interviews and exchanges of documentation between the developer and an expert in severe weather forecasting. But so far no AI system can solve these problems very well.

Artificial Neural Network (ANN) computing is an area that is receiving increased research interest. Since Richard Lippmann's [3] tutorial article "An Introduction to Computing with Neural Networks", Lippmann's article becomes one of the most widely referenced papers in the neural network literature. In Lippmann's [3] article, Multi-Layer Perceptron (MLP) neural network has been introduced. So far, the MLP is still the most widely used neural network in the world. But so results for rainfall estimation have not been very good using simple neural network models.

Artificial neuron network-based models are not yet sufficiently powerful to characterise complex systems. Moreover, a gap exists in the research literature between complex systems and general systems. To characterise complex systems, neuron-based and neural network-based group models are studied. Lie Groups were used in Tsao's [4] group theory approach to the computer simulation of 3D rigid motion. So far there have been limited studies with emphasis on setting a few free parameters in the activation function. Vecchi et al [5] studied the properties of a Feedforward Neural Network (FNN) which is able to adapt its activation function. In Chen and Chang's [6] paper, real variables a (gain) and b (slope) in the generalized sigmoid activation function are adjusted during the learning process. In Campolucci et al [7], a neuron-adaptive activation function built as a piecewise approximation with suitable cubic splines can have arbitrary shape. And this enables reduction of the overall size of the neural networks, trading connection complexity with activation function complexity. Neural networks with a neuron-adaptive activation function seem to provide better fitting properties than classical architectures with fixed activation function neurons.

Assessment of global climate change is a very important research area for the future of humans and their environment. Rainfall estimation is a key parameter in this research. During the past 20 years, there has been a great increase in our understanding of how satellite data can be used to estimate rainfall. But, even with the use of interactive computer systems, the time needed to prepare estimates of rainfall is very time consuming (about a half-hour). Verification results show that the average error for an event is about 30% ([8], [9], and [10]). As mentioned previously, an automatic estimation algorithm (Auto Estimator) has been developed for convective systems but there is still a need to increase the speed of computation and improve the accuracy of the satellite derived rainfall estimates.

Zhang, Murugesan, and Sadeghi (1995) developed a Polynomial Higher Order
Neural Network (PHONN) [13] for data simulation. Lu and Zhang [14] developed a model, called Polynomial and Trigonometric polynomial Higher Order Neural Network (PT-HONN). The special feature of PT-HONN model would be it could simulate polynomial and/or trigonometric polynomial functions. But in real world, data are complicated. Real world data always can not be simply simulated very well by single polynomial function, or even combination of polynomial and trigonometric polynomial functions. The ordinary higher order neural networks only can simulate such polynomial and/or trigonometric polynomial function. It is one reason why ordinary higher order neural networks could be failed to simulate complicated real world data.

M-PHONN model not only can simulate polynomial and/or trigonometric polynomial functions, but also can simulate multi-polynomial functions, such as combination of polynomial, trigonometric polynomial, and sigmoid polynomial functions. So M-PHONN model can achieve more accuracy for simulation of real world data. But this model have no adaptively adjustable coefficients. This paper will study M-PHONN model, a M-PHONN model with adaptively adjustable coefficients. For testing M-PHONN model, an M-PHONN simulator has been built and the comparison experimental results between M-PHONN and ordinary higher order neural network also will be presented here.

2 M-PHONN Model

The network architecture of M-PHONN is developed based on the characteristics of PHONN, THONN, and PT-HONN. The different part of M-PHONN is, M-PHONN includes a sigmoid polynomial function part. It is a multi-layer that consists of an input layer with input-units, and output layer with output-units, and two hidden layers consisting of intermediate processing units. A specify definition of M-PHONN is presented in the following:

\[ Z = \sum_{i,j=0}^{n} (a_{ij}(x)^{i}(y)^{j}) + b_{y} \sin^{i}(x) \cos^{j}(y) + c_{y} \frac{1}{(1 + \exp(-x))(1 + \exp(-y))^{j}} \]

The derivatives of M-PHONN Model for polynomial, trigonometric polynomial neuron and sigmoid polynomial are:

Let:

net\(_{pi}^{h}\) : input of the neuron

x\(_{pi}\) : output of neuron

f\(_{i}\) : mapping for input to output

If f\(_{i}\) is polynomial function:

\[ f_{i}^{\text{h}}(\text{net}_{pi}^{h}) = \partial x_{pi} / \partial (\text{net}_{pi}^{h}) \]

\[ = n(\text{net}_{pi}^{h})^{n-1} \]

If f\(_{i}\) is trigonometric polynomial function:

\[ f_{i}^{\text{h}}(\text{net}_{pi}^{h}) = \partial x_{pi} / \partial (\text{net}_{pi}^{h}) \]

\[ = \partial (\sin^{n}(\text{net}_{pi}^{h})) / \partial (\text{net}_{pi}^{h}) \]

\[ = n \sin^{n-1}(\text{net}_{pi}^{h}) \cos(\text{net}_{pi}^{h}) \]

or

\[ f_{i}^{\text{h}}(\text{net}_{pi}^{h}) = \partial x_{pi} / \partial (\text{net}_{pi}^{h}) \]

\[ = \partial (\cos^{n}(\text{net}_{pi}^{h})) / \partial (\text{net}_{pi}^{h}) \]

\[ = n \cos^{n-1}(\text{net}_{pi}^{h}) \sin(\text{net}_{pi}^{h}) \]
If $f_i$ is sigmoid polynomial function:

$$x_{pi} = f_i(n_{\text{net}}_{pi})$$

$$f_i'(n_{\text{net}}_{pi}) = \frac{\partial x_{pi}}{\partial (n_{\text{net}}_{pi})}$$

$$= \frac{\partial \left(1/(1 + \exp(-n_{\text{net}}_{pi}))\right)^n}{\partial (n_{\text{net}}_{pi})}$$

$$= n(x_{pi})^n(1 - x_{pi})$$

Based on derivatives of $M$-PHONN Model, a leaning algorithm has been developed.

3 The M-PHONN Simulator

The M-PHONN simulator has been written in the C computing language, running under X window on Sun workstation, based on the previous work did by Zhang and Fulcher [11]. A user-friendly GUI (Graphical User Interface) system has also been incorporated. When you run the system, any step, data or calculation can be reviewed and modified from different windows during processing. Hence, changing data, network models and comparing results can be done very easily and efficiently.

4 ANSER-PLUS SYSTEM

The architecture of the ANSER-plus system consists of three components: (1) USER System (which comprises several IBM PC computers); (2) TRAINING System (which uses a NCCF HDS 9000 mainframe computer); (3) CENTER System (which uses a SUN Ultra 10 workstation). The USER, TRAINING and CENTER Systems communicate with each other over an Ethernet LAN.

The output functions of ANSER can be either: (1) fixed-point rainfall estimates; (2) maximum isohyet area rainfall estimates; and/or (3) total aerial rainfall estimates. All three outputs can be saved to files, as well as being displayed on a colour screen.

5 Comparing M-PHONN with PT-HONN, PHONN Models

In this section, we will compare $M$-PHON with PHONN, and PT-HONN with the error per pattern and the average percent error. The results of comparative analysis experiment among $M$-PHONN and PT-HONN and PHONN are presented.

This comparison will be pay particular attention on the degree of the accuracy of the error. The aim of this comparative analysis is to identify how the feature of $M$-PHONN is when it is compared with PT-HONN and PHONN.

Rules of the comparison experiments - ‘Keep compared programs in the same environment’. Therefore the comparison will be based on the following conditions: same groups of testing data and same parameters of experimental environment such as learning rates and number of hidden layers and so on. Some experimental result are shown in the below:

For testing rainfall estimation data simulation, our expert knowledge of rainfall estimation based on the satellite observed cloud top temperature and cloud growth was used [9]. For example, when the cloud top temperature is between -58°C and -60°C, and the cloud growth is more than 2/3 latitude, the half hour rainfall estimate
is 0.94 inch according to the technique developed by Scofield and Oliver [10]. Details of this expert knowledge are listed in the following Table 5.1 [12].

<table>
<thead>
<tr>
<th>Cloud Temperature</th>
<th>Top Growth</th>
<th>Cloud Growth</th>
<th>Half Hour Rainfall Inches</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; -32° C</td>
<td>2/3</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>-36° C</td>
<td>2/3</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>-46° C</td>
<td>2/3</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>-55° C</td>
<td>2/3</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td>-60° C</td>
<td>2/3</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>&gt; -32° C</td>
<td>1/3</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>-36° C</td>
<td>1/3</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>-46° C</td>
<td>1/3</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>-55° C</td>
<td>1/3</td>
<td>0.43</td>
<td></td>
</tr>
<tr>
<td>-60° C</td>
<td>1/3</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td>-70° C</td>
<td>1/3</td>
<td>0.85</td>
<td></td>
</tr>
<tr>
<td>&lt; -80° C</td>
<td>1/3</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>&gt; -32° C</td>
<td>0</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>-36° C</td>
<td>0</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>-46° C</td>
<td>0</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>-55° C</td>
<td>0</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>-60° C</td>
<td>0</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>-70° C</td>
<td>0</td>
<td>0.49</td>
<td></td>
</tr>
<tr>
<td>&lt; -80° C</td>
<td>0</td>
<td>0.55</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1 Cloud Top Temperature, Cloud growth, and Rainfall inches

Table 5.2 presents the rainfall data estimation results using PHONN model and PT-HONN model. The average error of M-PHON is 5.42%. The average error of PT-HONN is 6.57%. The average error of PHONN is 5.58%. This means the M-PHON model is about 4.5% better than PT-HONN model when using the rainfall estimate experimental data base in Table 5.1. And This means the M-PHON model is about 15% better than PHONN model when using the rainfall estimate experimental data base.

<table>
<thead>
<tr>
<th>Cloud Top Temperature</th>
<th>Cloud Growth</th>
<th>PHONN</th>
<th>PT-HONN</th>
<th>M-PHON</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; -32° C</td>
<td>2/3</td>
<td>3.22</td>
<td>8.80</td>
<td>8.78</td>
</tr>
<tr>
<td>-36° C</td>
<td>2/3</td>
<td>9.92</td>
<td>6.48</td>
<td>6.52</td>
</tr>
<tr>
<td>-46° C</td>
<td>2/3</td>
<td>25.19</td>
<td>23.18</td>
<td>22.58</td>
</tr>
<tr>
<td>-55° C</td>
<td>2/3</td>
<td>14.63</td>
<td>14.22</td>
<td>14.21</td>
</tr>
<tr>
<td>-60° C</td>
<td>2/3</td>
<td>3.66</td>
<td>0.09</td>
<td>0.12</td>
</tr>
<tr>
<td>&gt; -32° C</td>
<td>1/3</td>
<td>10.47</td>
<td>10.11</td>
<td>8.03</td>
</tr>
<tr>
<td>-36° C</td>
<td>1/3</td>
<td>3.50</td>
<td>4.25</td>
<td>4.18</td>
</tr>
<tr>
<td>-46° C</td>
<td>1/3</td>
<td>3.52</td>
<td>4.63</td>
<td>4.24</td>
</tr>
<tr>
<td>-55° C</td>
<td>1/3</td>
<td>0.22</td>
<td>2.04</td>
<td>1.89</td>
</tr>
<tr>
<td>-60° C</td>
<td>1/3</td>
<td>3.21</td>
<td>0.30</td>
<td>0.65</td>
</tr>
<tr>
<td>-70° C</td>
<td>1/3</td>
<td>9.01</td>
<td>5.08</td>
<td>5.12</td>
</tr>
<tr>
<td>&lt; -80° C</td>
<td>1/3</td>
<td>3.89</td>
<td>1.21</td>
<td>1.32</td>
</tr>
<tr>
<td>&gt; -32° C</td>
<td>0</td>
<td>9.81</td>
<td>7.86</td>
<td>7.24</td>
</tr>
<tr>
<td>-36° C</td>
<td>0</td>
<td>2.98</td>
<td>3.62</td>
<td>3.25</td>
</tr>
<tr>
<td>-46° C</td>
<td>0</td>
<td>5.69</td>
<td>5.68</td>
<td>5.67</td>
</tr>
<tr>
<td>-55° C</td>
<td>0</td>
<td>5.28</td>
<td>4.35</td>
<td>4.03</td>
</tr>
<tr>
<td>-60° C</td>
<td>0</td>
<td>3.32</td>
<td>1.33</td>
<td>1.43</td>
</tr>
<tr>
<td>-70° C</td>
<td>0</td>
<td>0.77</td>
<td>3.57</td>
<td>2.78</td>
</tr>
<tr>
<td>&lt; -80° C</td>
<td>0</td>
<td>2.50</td>
<td>1.10</td>
<td>1.12</td>
</tr>
</tbody>
</table>

Table 5.2 Rainfall data simulation using PHONN, PT-HONN, and M-PHON model

The estimation of heavy convective rainfall has been tested using the ANSER-plus system with the M-PHON models. Since rainfall estimation is a complex, nonlinear, discontinuous task, M-PHON models are able to perform this well; simple neural network models are inaccurate at points of discontinuity. It has been demonstrated that by using artificial M-PHON models,
rainfall estimation can be performed ten times faster (compared with decision tree techniques). Moreover, the average rainfall estimate errors for the total precipitation event can be reduced to less than 20%.

6 CONCLUSION

M-PHONN models are studied in this paper. Using M-PHONN models and an automatic estimation technique, rainfall estimation expert system with faster speed and more accuracy could be developed. M-PHONN models can be used to complement the already established Auto-Estimator algorithm by increasing the speed, accuracy, and the robustness of the rainfall estimates.

REFERENCE


From inns@cos.com
To: h.qi@uws.edu.au; m.zhang@pcs.cn.edu; Roderick.Scofield@noaa.gov
Sent: Friday, March 30, 2001 1:24PM
Subject INNS Abstract Submission Acceptance Letter

Dear Hui Qi, Ming ZHANG, and Roderick SCOFIELD:

RE: Abstract Title: Rainfall Estimation Using M-PHONN Model
Hui Qi, Ming Zhang, Roderick Scofield

Tracking ID:
29839

We are pleased to inform you that your paper has been accepted for the International Joint INNS-IEEE Conference on Neural Networks to be held in Washington DC, July 14-19, 2001.

Please use the following guidelines to submit your paper:

Beginning April 5, 2001, all papers must be submitted electronically using the web site link at www.geocities.com/ijcnn. Once you have linked on the site, click on Instructions for Authors and an author kit will be provided to guide you through the process.

The submission system can only accept postscript or pdf files, with a camera-ready version of your paper no longer than six (6) pages long, in a 2-column format (IEEE Conference style).

Papers will be accepted from April 5, 2001 through May 15, 2001.

Authors that submit their papers must fax their Copyright form (see information in the author's kit) AND register for the conference by May 15, 2001, or the paper will not be included in the final printed proceedings and the CDROM, (fax #856-423-3420). "Orphans" (papers in which at least one author or co-author has not registered for the conference) will NOT be published.

Reviewers may have critiqued your submitted abstract. To assist in preparation of your paper, below please find Reviewer comments. If nothing is displayed, there were no comments.

Reviewer 3: The authors deal with the problem of rainfall estimation. They have improved their previous work in this area by using a more powerful NN model. They report improved results (more accurate predictions).

Please note, you will be notified of the type of presentation assigned (oral or poster) in a separate message.

Please make sure to refer to your Abstract Tracking ID number in all correspondence with the Program Committee.
Thank you very much for your contribution! We look forward to seeing you in DC at the IJCNN '01!

Sincerely,

Danil Prokhorov
IJCNN '01 TPC
USING PT-HONN MODELS
FOR MULTI-POLYNOMIAL FUNCTION SIMULATION*

BO LU, HUI QI, MING ZHANG
University of Western Sydney Macarthur, Campbelltown, NSW 2560, Australia

RODERICK A. SCOFIELD
NOAA/NESDIS/ORA, 5200 Auth Road, Room 601, Washington DC, USA

ABSTRACT
A new model, the Polynomial and Trigonometric polynomial Higher Order Neural Network (PT-HONN), has been developed. Using Sun workstation, C and Motif, the PT-HONN Simulator has been built as well. Global real time data can not always be simply simulated by single polynomial function. So the ordinary higher order neural networks could fail to simulate these kinds of global data. However, the PT-HONN model can simulate multi-polynomial functions; the PT-HONN model can achieve more accuracy for global real time data simulation. The financial data and rainfall data simulation comparison experiments between PT-HONN and ordinary higher order neural network also shows that PT-HONN always gives 2-3% more accuracy than ordinary higher order neural networks.

KEYWORDS
Multi-Polynomial, Higher Order Neural Network Model, Data Simulation, Financial Data, Rainfall Data

1 INTRODUCTION
In the last few years, more and more people [1, 2] have been studying neural networks for simulations and predictions. Zhang, Murugesan, and Sadeghi (1995) developed a Polynomial Higher Order Neural Network (PHONN) [3] for economic data simulation. Zhang and Pulcher (1996) also developed neural network group models [4] for financial data simulation. As a one step for that research, Trigonometric polynomial Higher Order Neural Network (THONG) [5, 6, 7, 8] has been developed and studied; THONN not only handle discontinuities but also can simulate higher frequency and higher order nonlinear data.

Real time data can be complicated. Global data can not be always simply simulated by single polynomial functions. The ordinary higher order neural networks can only simulate single polynomial functions (polynomial or trigonometric polynomial functions). This is why ordinary higher order neural networks can fail to simulate complicated real time global data.

A new model, the Polynomial and Trigonometric polynomial Higher Order Neural Network (PT-HONN), has been developed in this paper. The special feature of PT-HONN model would be that it can simulate multi-polynomial functions. Therefore, it is expected that the PT-HONN model can achieve more accuracy for global real time data simulation. For testing PT-HONN models, a PT-HONN simulator has been built. A comparison of experimental results has also been presented between PT-HONN and ordinary higher order neural network.
2 PT-HONN MODEL

The network architecture of PT-HONN has combined both the characteristics of PHONN and THONN. It is a multi-layer network that consists of an input layer with input-units, and output layer with output-units, and two hidden layers consisting of intermediate processing units. A specific definition of PT-HONN is

\[ Z = \sum_{i=0}^{n} (a_i S_i(x) \cos^i(y) + b_i x^i y^i + c_i \cos^i(x) \sin^i(y)) \]

presented in the following:

The derivatives of PT-HONN Model for polynomial and trigonometric polynomial neuron are:

Let:

set \( h \), \( p_i \) : input of the neuron

\( s_{pi} \) : output of neuron

\( f_{pi}^h \) : mapping for input to output

\[ x_{pi} = f_{pi}^h(\text{net}^h_{pi}) \]

\[ = (\text{net}^h_{pi})^n + \sin^n(\text{net}^h_{pi}) + \cos^n(\text{net}^h_{pi}) \]

\[ f_{pi}^h(\text{net}^h_{pi}) = \partial x_{pi}/\partial (\text{net}^h_{pi}) \]

\[ = \partial((\text{net}^h_{pi})^n + \sin^n(\text{net}^h_{pi}) + \cos^n(\text{net}^h_{pi}))/\partial (\text{net}^h_{pi}) \]

\[ = n(\text{net}^h_{pi})^{n-1} + n \sin^{n-1}(\text{net}^h_{pi}) \cos(\text{net}^h_{pi}) + n \cos^{n-1}(\text{net}^h_{pi}) \sin(\text{net}^h_{pi}) \]

Based on derivatives of PT-HONN Model, a leaning algorithm has been developed.

3 PT-HONN SIMULATOR

The PT-HONN simulator has been written in the C computing language, running under X window on Sun workstation. This simulator is based on the previous work done by Zhang and Fulcher [4]. A user-friendly GUI (Graphical User Interface) system has also been incorporated. When the system is run, any step, data or calculation can be reviewed and modified from different windows during the processing. Hence, changing data, network models and comparing results can be done very easily and efficiently.

4. FINANCIAL DATA SIMULATION

After the PT-HONN simulator was developed, real time testing was done to prove it worked with real time global data. The PT-HONN simulator was also compared with other artificial neural network.

The financial data simulation results of comparative analysis experiments among PT-HONN, PHONN (Polynomial Higher Order Neural Network) and THONN (Trigonometric polynomial Higher Order Neural Network) are presented.

This comparison will be accomplished with particular attention on the degree of the accuracy of the error. The purpose of this comparative analysis is to identify how the features identified by PT-HONN. Therefore it is necessary to make clear what the difference among the new method and others.

In this section, we will compare PHONN, THONN and PT-HONN with the error per pattern and the average percent error. The specialty of PHONN and THONN program is to simulate the kinds of high order polynomial or trigonometric polynomial data. The PT-HONN program was developed based on the PHONN and THONN. Thus, if PHONN and THONN can obtain good results of the simulation with some kind of data results should improve with PT-HONN. Some experimental results will be shown in the following sections

4.1 Liabilities and Assets on NonResidents

All Banks - Liabilities and Assets on NonResidents of Total Liabilities between 1996.8 and 1997.5 has been tested. Results of the comparison have shown that the average |Error| % of the PHONN is 1.9895%, THONN is 2.0976% and the PT-HONN is 1.9375%, see following Table 4.1. This means PT-HONN model is about 2.6% better than PHONN and 7.5% better than THONN.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN</th>
<th>THONN</th>
<th>PT-HONN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov/96</td>
<td>1.78</td>
<td>1.62</td>
<td>1.95</td>
</tr>
<tr>
<td>Oct/96</td>
<td>4.65</td>
<td>4.69</td>
<td>5.21</td>
</tr>
<tr>
<td>Sep/96</td>
<td>0.99</td>
<td>1.60</td>
<td>1.03</td>
</tr>
<tr>
<td>Aug/96</td>
<td>0.69</td>
<td>0.17</td>
<td>0.58</td>
</tr>
<tr>
<td>Jul/96</td>
<td>1.55</td>
<td>1.76</td>
<td>2.34</td>
</tr>
<tr>
<td>Jun/96</td>
<td>1.42</td>
<td>1.25</td>
<td>0.98</td>
</tr>
<tr>
<td>May/96</td>
<td>3.64</td>
<td>3.90</td>
<td>2.58</td>
</tr>
</tbody>
</table>
Table 4.1 All Banks - Liabilities and Assets on NonResidents of Total Liabilities between 96.8 - 97.5 (Reserve Bank of Australia Bulletin 1999.7 Page S3)

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN Error %</th>
<th>THONN Error%</th>
<th>PT-HONN Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar/97</td>
<td>0.71</td>
<td>1.61</td>
<td>0.03</td>
</tr>
<tr>
<td>Apr/97</td>
<td>2.35</td>
<td>1.34</td>
<td>2.24</td>
</tr>
<tr>
<td>May/97</td>
<td>2.12</td>
<td>3.02</td>
<td>2.42</td>
</tr>
<tr>
<td>Average</td>
<td>1.9895%</td>
<td>2.0976%</td>
<td>1.9375%</td>
</tr>
</tbody>
</table>

4.2 Credit Card Lending

All Banks - Credit Card Lending on Total Number of Accounts between 1996.8 and 1997.6 has been tested. The result of this comparison has shown that the average Error % of the PHONN is 1.8927%, THONN is 0.9717% and the PT-HONN is 0.5208%, see following Table 4.2. This means PT-HONN model is about 72.4% better than PHONN and 46.4% better than THONN.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN Error %</th>
<th>THONN Error%</th>
<th>PT-HONN Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/96</td>
<td>4.28</td>
<td>1.77</td>
<td>1.05</td>
</tr>
<tr>
<td>Sep/96</td>
<td>3.71</td>
<td>1.65</td>
<td>1.00</td>
</tr>
<tr>
<td>Oct/96</td>
<td>3.13</td>
<td>1.01</td>
<td>1.00</td>
</tr>
<tr>
<td>Nov/96</td>
<td>1.93</td>
<td>0.86</td>
<td>0.57</td>
</tr>
<tr>
<td>Dec/96</td>
<td>1.04</td>
<td>0.20</td>
<td>0.34</td>
</tr>
<tr>
<td>Jan/97</td>
<td>0.48</td>
<td>0.50</td>
<td>0.55</td>
</tr>
<tr>
<td>Feb/97</td>
<td>0.23</td>
<td>0.64</td>
<td>0.19</td>
</tr>
<tr>
<td>Mar/97</td>
<td>0.46</td>
<td>0.62</td>
<td>0.03</td>
</tr>
<tr>
<td>Apr/97</td>
<td>1.64</td>
<td>0.46</td>
<td>0.59</td>
</tr>
<tr>
<td>May/97</td>
<td>1.44</td>
<td>1.62</td>
<td>0.08</td>
</tr>
<tr>
<td>June/97</td>
<td>2.46</td>
<td>2.46</td>
<td>0.33</td>
</tr>
<tr>
<td>Average</td>
<td>1.8927%</td>
<td>0.9717%</td>
<td>0.5208%</td>
</tr>
</tbody>
</table>

Table 4.2 All Banks - Credit Card Lending on Total Number of Accounts between 96.8 - 97.6 (Reserve Bank of Australia Bulletin 1999.7 P. S18)

4.3 Exchange Rate on Japanese

Exchange Rate on the Japanese (Yen) between 1996.8 and 1997.6 has been tested. The results of this comparison has shown that the average Error % of the PHONN is 2.3452%, THONN is 2.2780% and the PT-HONN is 0.9130%, see following Table 4.3. This means PT-HONN model is about 61.0% better than PHONN and 59.9% better than THONN.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN Error %</th>
<th>THONN Error%</th>
<th>PT-HONN Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/96</td>
<td>0.27</td>
<td>1.23</td>
<td>1.78</td>
</tr>
<tr>
<td>Sep/96</td>
<td>3.72</td>
<td>3.67</td>
<td>0.64</td>
</tr>
<tr>
<td>Oct/96</td>
<td>5.39</td>
<td>5.38</td>
<td>2.16</td>
</tr>
<tr>
<td>Nov/96</td>
<td>2.78</td>
<td>1.31</td>
<td>0.35</td>
</tr>
<tr>
<td>Dec/96</td>
<td>0.23</td>
<td>0.82</td>
<td>1.18</td>
</tr>
<tr>
<td>Jan/97</td>
<td>1.33</td>
<td>0.17</td>
<td>1.20</td>
</tr>
<tr>
<td>Feb/97</td>
<td>3.61</td>
<td>3.69</td>
<td>0.08</td>
</tr>
<tr>
<td>Mar/97</td>
<td>4.10</td>
<td>4.29</td>
<td>2.11</td>
</tr>
<tr>
<td>Apr/97</td>
<td>3.08</td>
<td>1.88</td>
<td>0.32</td>
</tr>
<tr>
<td>May/97</td>
<td>1.03</td>
<td>2.06</td>
<td>0.18</td>
</tr>
<tr>
<td>Jun/97</td>
<td>0.24</td>
<td>0.55</td>
<td>0.04</td>
</tr>
<tr>
<td>Average</td>
<td>2.3452%</td>
<td>2.2780%</td>
<td>0.9130%</td>
</tr>
</tbody>
</table>

Table 4.3 Exchange Rate on Japanese (yen) between 96.8 - 97.6 (Reserve Bank of Australia Bulletin 1999.7 Page S53)

4.4 Exchange Rate on United States

Exchange Rate on the United States (dollar) between 1996.8 and 1997.6 has been tested. The result of this comparison has shown that the average Error % of the PHONN is 1.6057%, THONN is 1.6969% and the PT-HONN is 1.3283%, see following Table 4.4. This means PT-HONN model is about 17.2% better than PHONN and 21.7% better than THONN.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN Error %</th>
<th>THONN Error%</th>
<th>PT-HONN Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/96</td>
<td>0.97</td>
<td>1.80</td>
<td>0.69</td>
</tr>
<tr>
<td>Sep/96</td>
<td>0.08</td>
<td>0.42</td>
<td>0.03</td>
</tr>
<tr>
<td>Oct/96</td>
<td>0.27</td>
<td>0.34</td>
<td>0.03</td>
</tr>
<tr>
<td>Nov/96</td>
<td>1.19</td>
<td>1.69</td>
<td>0.90</td>
</tr>
<tr>
<td>Dec/96</td>
<td>2.33</td>
<td>3.04</td>
<td>1.77</td>
</tr>
<tr>
<td>Jan/97</td>
<td>2.84</td>
<td>2.48</td>
<td>3.57</td>
</tr>
<tr>
<td>Feb/97</td>
<td>4.05</td>
<td>3.86</td>
<td>3.19</td>
</tr>
<tr>
<td>Mar/97</td>
<td>1.39</td>
<td>0.95</td>
<td>0.92</td>
</tr>
<tr>
<td>Apr/97</td>
<td>0.12</td>
<td>0.50</td>
<td>0.01</td>
</tr>
<tr>
<td>May/97</td>
<td>2.45</td>
<td>2.46</td>
<td>2.22</td>
</tr>
<tr>
<td>June/97</td>
<td>1.98</td>
<td>1.14</td>
<td>1.29</td>
</tr>
<tr>
<td>Average</td>
<td>1.6057%</td>
<td>1.6969%</td>
<td>1.3283%</td>
</tr>
</tbody>
</table>

Table 4.4 Exchange Rate on United States (dollar) between 96.8 - 97.6 (Reserve Bank of Australia Bulletin 1999.7 Page S53)
4.5 Labor Force on Labor

Labor Force on Labor between 1997.8 and 1998.6 has been tested. The results of this comparison has shown that the average error % of the PHONN is 4.0252%, THONN is 5.3549% and the PT-HONN is 3.9827%, see following Table 4.5. This means PT-HONN model is about 17.1% better than PHONN and 25.6% better than THONN.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN % Error</th>
<th>THONN % Error</th>
<th>PT-HONN % Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/97</td>
<td>6.22</td>
<td>10.62</td>
<td>6.80</td>
</tr>
<tr>
<td>Sep/97</td>
<td>2.30</td>
<td>5.84</td>
<td>1.71</td>
</tr>
<tr>
<td>Oct/97</td>
<td>0.23</td>
<td>4.21</td>
<td>0.10</td>
</tr>
<tr>
<td>Nov/97</td>
<td>8.31</td>
<td>12.07</td>
<td>8.54</td>
</tr>
<tr>
<td>Dec/97</td>
<td>3.32</td>
<td>2.99</td>
<td>3.73</td>
</tr>
<tr>
<td>Jan/98</td>
<td>8.67</td>
<td>7.65</td>
<td>8.33</td>
</tr>
<tr>
<td>Feb/98</td>
<td>4.85</td>
<td>3.42</td>
<td>4.46</td>
</tr>
<tr>
<td>Mar/98</td>
<td>0.30</td>
<td>0.74</td>
<td>0.04</td>
</tr>
<tr>
<td>Apr/98</td>
<td>3.63</td>
<td>3.71</td>
<td>3.75</td>
</tr>
<tr>
<td>May/98</td>
<td>2.70</td>
<td>3.67</td>
<td>3.10</td>
</tr>
<tr>
<td></td>
<td>3.75</td>
<td>3.98</td>
<td>3.25</td>
</tr>
<tr>
<td>Average</td>
<td>4.0252%</td>
<td>5.3549%</td>
<td>3.9827%</td>
</tr>
</tbody>
</table>

Table 4.5 Labour Force on Labour between 97.8 – 98.6 (Reserve Bank of Australia Bulletin 1999.7 Page 563)

4.6 Share Price Indices

Share Price Indices - the Dow Jones Industrial (a measure of 30 industrial stocks in the US) between 1998.6 and 1999.6 has been tested. The result of this comparison has shown that the average error % of the PHONN is 3.5730%, THONN is 3.5755% and the PT-HONN is 3.4853%, see following Table 4.6. This means PT-HONN model is about 2.45% better than PHONN and 2.52% better than THONN.

<table>
<thead>
<tr>
<th>Month/Year</th>
<th>PHONN % Error</th>
<th>THONN % Error</th>
<th>PT-HONN % Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug/98</td>
<td>0.48</td>
<td>0.25</td>
<td>0.09</td>
</tr>
<tr>
<td>Sep/98</td>
<td>7.68</td>
<td>7.22</td>
<td>7.93</td>
</tr>
<tr>
<td>Oct/98</td>
<td>1.58</td>
<td>0.93</td>
<td>0.89</td>
</tr>
<tr>
<td>Nov/98</td>
<td>4.27</td>
<td>4.09</td>
<td>4.68</td>
</tr>
<tr>
<td>Dec/98</td>
<td>5.19</td>
<td>4.90</td>
<td>4.67</td>
</tr>
<tr>
<td>Jan/99</td>
<td>4.71</td>
<td>5.97</td>
<td>4.65</td>
</tr>
<tr>
<td>Feb/99</td>
<td>1.59</td>
<td>2.07</td>
<td>1.95</td>
</tr>
<tr>
<td>Mar/99</td>
<td>5.89</td>
<td>5.57</td>
<td>6.30</td>
</tr>
<tr>
<td>Apr/99</td>
<td>1.76</td>
<td>2.19</td>
<td>1.20</td>
</tr>
<tr>
<td>May/99</td>
<td>2.79</td>
<td>3.08</td>
<td>2.50</td>
</tr>
<tr>
<td>Jun/99</td>
<td>3.37</td>
<td>3.07</td>
<td>3.47</td>
</tr>
<tr>
<td>Average</td>
<td>3.5730%</td>
<td>3.5755%</td>
<td>3.4853%</td>
</tr>
</tbody>
</table>

Table 4.6 Share Price Indices – United States on Dow Jones industrial between 1998.8-99.6 (Reserve Bank of Australia Bulletin 1999.7 Page 550)

5 RAINFALL ESTIMATION DATA SIMULATION

For testing rainfall estimation data simulation, our expert knowledge of rainfall estimation based on the satellite observed cloud top temperature and cloud growth was used [9]. For example, when the cloud top temperature is between -58°C and -60°C, and the cloud growth is more than 2/3 latitude, the half hour rainfall estimate is 0.94 inch according to the technique developed by Schofield and Oliver [10]. Details of this expert knowledge are listed in the following Table 5.1 [11].

<table>
<thead>
<tr>
<th>Cloud Top Temperature</th>
<th>Cloud Growth Latitude Degree</th>
<th>Half Hour Rainfall Inches</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;-32°C</td>
<td>2/3</td>
<td>0.05</td>
</tr>
<tr>
<td>-36°C</td>
<td>2/3</td>
<td>0.20</td>
</tr>
<tr>
<td>-46°C</td>
<td>2/3</td>
<td>0.48</td>
</tr>
<tr>
<td>-55°C</td>
<td>2/3</td>
<td>0.79</td>
</tr>
<tr>
<td>-60°C</td>
<td>2/3</td>
<td>0.94</td>
</tr>
<tr>
<td>&gt;-32°C</td>
<td>1/3</td>
<td>0.05</td>
</tr>
<tr>
<td>-36°C</td>
<td>1/3</td>
<td>0.13</td>
</tr>
<tr>
<td>-46°C</td>
<td>1/3</td>
<td>0.24</td>
</tr>
<tr>
<td>-55°C</td>
<td>1/3</td>
<td>0.43</td>
</tr>
<tr>
<td>-60°C</td>
<td>1/3</td>
<td>0.65</td>
</tr>
<tr>
<td>-70°C</td>
<td>1/3</td>
<td>0.85</td>
</tr>
<tr>
<td>&lt; -80°C</td>
<td>1/3</td>
<td>0.95</td>
</tr>
<tr>
<td>&gt;-32°C</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>-36°C</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>-46°C</td>
<td>0</td>
<td>0.11</td>
</tr>
<tr>
<td>-55°C</td>
<td>0</td>
<td>0.22</td>
</tr>
<tr>
<td>-60°C</td>
<td>0</td>
<td>0.36</td>
</tr>
<tr>
<td>-70°C</td>
<td>0</td>
<td>0.49</td>
</tr>
</tbody>
</table>

Table 5.1 Cloud Top Temperature, Cloud growth, and Rainfall inches

Table 5.2 presents the rainfall data estimation results using PHONN model and PT-HONN model. The average error of PT-HONN is 5.68%. The average error of PHONN is 6.36%. There is a 0.68% difference. This means the PT-HONN model is about 10% better than PHONN model when using the rainfall estimate experimental data base in Table 5.1.
<table>
<thead>
<tr>
<th>Temperature</th>
<th>Growth Latitude Degree</th>
<th>Error %</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;-32°C</td>
<td>2/3</td>
<td>3.22</td>
<td>8.80</td>
</tr>
<tr>
<td>-36°C</td>
<td>2/3</td>
<td>9.92</td>
<td>6.48</td>
</tr>
<tr>
<td>-46°C</td>
<td>2/3</td>
<td>25.19</td>
<td>23.18</td>
</tr>
<tr>
<td>-55°C</td>
<td>2/3</td>
<td>14.63</td>
<td>14.22</td>
</tr>
<tr>
<td>-60°C</td>
<td>2/3</td>
<td>3.66</td>
<td>0.09</td>
</tr>
<tr>
<td>&gt;-32°C</td>
<td>1/3</td>
<td>10.47</td>
<td>10.11</td>
</tr>
<tr>
<td>-36°C</td>
<td>1/3</td>
<td>3.50</td>
<td>4.25</td>
</tr>
<tr>
<td>-46°C</td>
<td>1/3</td>
<td>3.52</td>
<td>4.63</td>
</tr>
<tr>
<td>-55°C</td>
<td>1/3</td>
<td>0.22</td>
<td>2.04</td>
</tr>
<tr>
<td>-60°C</td>
<td>1/3</td>
<td>3.21</td>
<td>0.30</td>
</tr>
<tr>
<td>-70°C</td>
<td>1/3</td>
<td>9.01</td>
<td>5.08</td>
</tr>
<tr>
<td>&lt;-80°C</td>
<td>1/3</td>
<td>3.89</td>
<td>1.21</td>
</tr>
<tr>
<td>&gt;-32°C</td>
<td>0</td>
<td>9.81</td>
<td>7.86</td>
</tr>
<tr>
<td>-36°C</td>
<td>0</td>
<td>2.98</td>
<td>3.62</td>
</tr>
<tr>
<td>-46°C</td>
<td>0</td>
<td>5.69</td>
<td>5.68</td>
</tr>
<tr>
<td>-55°C</td>
<td>0</td>
<td>5.28</td>
<td>4.35</td>
</tr>
<tr>
<td>-60°C</td>
<td>0</td>
<td>3.32</td>
<td>1.33</td>
</tr>
<tr>
<td>-70°C</td>
<td>0</td>
<td>0.77</td>
<td>3.57</td>
</tr>
<tr>
<td>&lt;-80°C</td>
<td>0</td>
<td>2.50</td>
<td>1.10</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>6.36</td>
<td>5.68</td>
</tr>
</tbody>
</table>

Table 5.2 Rainfall data simulation using PHONN and PT-HONG

6 CONCLUSION

The results of the comparative experiments show that the PT-HONG model is able to simulate higher frequency and higher order non-linear data, as well as being able to simulate a multi-polynomial function. Also, the PT-HONG model is always 2-50% more accurate than ordinary higher order neural networks, if the data is a polynomial and trigonometric polynomial mixed function. The PT-HONG model can not only be used for financial data, but also for satellite data rainfall estimation.

As the next step of this research, a new model called Multi-Polynomial Higher Order Neural Network will be studied for simulating complicated and mixed multi-polynomial functions and real time global data (financial and satellite data).

REFERENCES


NEURON-ADAPTIVE NEURAL NETWORK MODEL FOR ESTIMATING HEAVY RAINFALL FROM SATELLITE DATA *

MING ZHANG, SHUXIANG XU, HUI QI
Department of Computing and Information Systems
University of Western Sydney Macarthur, PO BOX 555, Campbelltown, NSW, Australia

RODERIC A. SCOFIELD
NOAA/NESDIS/SAL, 5200 Auth Road, Room 601, Washington DC, USA

ABSTRACT

Neuron-Adaptive Feedforward Neural Network (NANN) models with a neuron-adaptive activation function are studied in this paper. The NANN models for estimating heavy convective rainfall from satellite data has been tested as well. Using ANSER-plus expert system, the average rainfall estimate errors for the total precipitation event could be reduced to less than 20%.

KEYWORDS


1 INTRODUCTION

Currently, satellite derived precipitation estimates [1] and 3 hour precipitation outlooks for convective systems, extra-tropical cyclones, and tropical cyclones are computed on the NOAA/NESDIS Interactive Flash Flood Analyser (IFFA) system and transmitted to National Weather Service Forecast Offices, and River Forecast Centres. However, this system permits the computation of rainfall estimates for only one convective system and is very time consuming. This is due to the considerable time needed for image processing, interpretation, and the computation involved in the estimation of rainfall. If there were several storms occurring, an automatic estimation technique would be useful in providing rainfall estimates for the entire country. But the classic methods cannot give good results so far. Artificial Intelligent (AI) is another possible way to solve this problem. Some artificial intelligence system for weather forecasting are designed to be objective and automated, some are designed to augment human skill. In Knowledge Augmented Severe Storms Predictor (KASSP), knowledge was elicited in a series of interviews and exchanges of documentation between the developer and an expert in severe weather forecasting. But so far no AI system can solve these problems very well.

Artificial Neural Network (ANN) computing is an area that is receiving increased research interest. Since Richard Lippmann's [2] tutorial article "An Introduction to Computing with Neural Networks", Lippmann's article becomes one of the most widely referenced papers in the neural network literature. In Lippmann's [2] article, Multi-Layer Perceptron (MLP) neural network has been introduced. So far, the MLP is still the most widely used neural network in the world. But no one had very good results for rainfall estimation only using simple neural network models.

Artificial neuron network-based models are not yet sufficiently powerful to characterise complex systems. Moreover, a gap exists in the research literature between complex systems and general systems. To characterise complex systems, neuron-based and neural network-based group models are studied. Lumer [3] proposed a new mechanism of selective attention among perceptual groups as part of his computational model of early vision. Lie Groups were used in Tsao's [4] group theory approach to the computer simulation of 3D rigid motion.

So far there have been limited studies with emphasis on setting a few free parameters in the activation function. Vecci et al [5] studied the properties of a Feedforward Neural Network (FNN) which is able to adapt its activation function. In Chen and Chang's [6] paper, real variables a (gain) and b (slope) in the generalised sigmoid activation function are adjusted during the learning process. In Campolucci et al [7], a neuron-adaptive activation function built as a piecewise approximation with suitable cubic splines can have arbitrary shape and this enables reduction of the overall size of the neural networks, trading connection complexity with activation function complexity. Neural networks with a neuron-adaptive activation function seem to provide better

* This work was supported by an University of Western Sydney Macarthur Foundation Grant in 1999 and was performed while Dr. Ming Zhang held a Senior USA National Research Council - NOAA/NESDIS/SAL Research Associateship.
fitting properties than classical architectures with fixed activation function neurones.

Assessment of global climate change is a very important research area for the future of humans and their environment. Rainfall estimation is a key parameter in this research. During the past 20 years, there has been a great increase in our understanding of how satellite data can be used to estimate rainfall. But, even with the use of interactive computer systems, the time needed to prepare estimates of rainfall is very time consuming (about a half-hour). Verification results show that the average error for an event is about 30% ([8], [9], and [10]). An automatic estimation technique with faster speed and more accuracy are the issues the experts want to solve.

2 ANSER-PLUS EXPERT SYSTEM

The architecture of the ANSER-plus system consists of three components: (1) USER System (which comprises several IBM PC computers); (2) TRAINING System (which uses a NCCF HDS 9000 mainframe computer); (3) CENTER System (which uses a SUN Ultra 10 workstation). The USER, TRAINING and CENTER Systems communicate with each other over an Ethernet LAN.

The output functions of ANSER can be either: (1) fixed-point rainfall estimates; (2) maximum isohyet area rainfall estimates; and/or (3) total annual rainfall estimates. All three outputs can be saved to files, as well as being displayed on a colour screen.

3. DEFINITION OF NANN

The network architecture of NANN is a multilayer feed-forward network that consists of an input layer with input-units, an output layer with output-units, and one hidden layer consisting of intermediate processing units. While there is no activation function in the input layer and the output neurons are summing units (linear activation), our activation function for the hidden layer processing units is a Neuron-Adaptive Activation Function (NAAF) defined as

\[ \sigma(x) = a_1 \cdot \sin(b_1 \cdot x) + a_2 \cdot e^{-a_3 x} + \frac{a_3}{1 + e^{-a_3 x}} \]  \hspace{1cm} (1)

where

\[ a_1, b_1, a_2, b_2, a_3, b_3 \]

are real variables which will be adjusted (as well as weights) during training. One typical NANN architecture is depicted in Figure 1.

As Leshno et al [11] established that multilayer feedforward networks with a nonpolynomial activation function can approximate any continuous function. We conclude that NANN with a NAAF is a universal approximator to continuous functions.

A NANN with more than one hidden layer can be defined as:

\[ F_i(X) = \sum_{k} w_{ik} \sigma \left( \sum_{l} v_{lk} \sigma \left( \cdots \sigma \left( \sum_j u_{lj} X \right) \right) \right) \]  \hspace{1cm} (2)

The learning algorithm is based on steepest descent rule (Rumelhart et al [12]). Essentially, it is similar to traditional back propagation algorithm, however, as parameters in (1) can be adjusted, it provides more flexibility and better approximation ability for NANN.

We use the following notations:

\[ I_{ik}, \quad \text{the input or internal state of the } i\text{th neuron in the } k\text{th layer} \]

\[ w_{ij,k}, \quad \text{the weight that connects the } j\text{th neuron in layer } k-1 \text{ and the } i\text{th neuron in layer } k \]

\[ O_{ik}, \quad \text{the value of output from the } i\text{th neuron in layer } k \]

\[ a_1, a_2, b_2, a_3, b_3 \quad \text{adjustable variables in activation function} \]

\[ \theta_{ik}, \quad \text{the threshold value of the } i\text{th neuron in the } k\text{th layer} \]

\[ d_j, \quad \text{the } j\text{th desired output value} \]

\[ r, \quad \text{the iteration number} \]

\[ \eta, \quad \text{momentum} \]

\[ \beta, \quad \text{learning rate} \]

\[ m, \quad \text{total number of output layer neurons} \]

\[ l, \quad \text{total number of network layers} \]

The input-output relation of the \( i\)th neuron in the \( k\)th layer can be described by:
\[ I_{j,k} = \sum_{l,j,l,k} w_{j,k} O_{j,k} \cdot \theta_{j,k} \]

where \( j \) is the number of neurons in layer \( k-1 \), and

\[ O_{j,k} = \Psi(I_{j,k}) = A1_{j,k} \cdot \sin(b1_{j,k} \cdot I_{j,k}) + A2_{j,k} \cdot e^{-b2_{j,k} \cdot I_{j,k}} + \frac{A3_{j,k}}{1 + e^{-b3_{j,k} \cdot I_{j,k}}} \]

The energy function is

\[ E = \frac{1}{2} \sum_{j,k} (d_{j,k} - O_{j,k})^2 \]

The aim of learning is to minimize the energy function by adjusting the weights associated with various interconnections, and the variables in the activation function:

\[ w_{i,j,k}^{(t)} = \eta w_{i,j,k}^{(t-1)} + \beta \frac{\partial E}{\partial w_{i,j,k}} \]

\[ \theta_{i,k}^{(t)} = \eta \theta_{i,k}^{(t-1)} + \beta \frac{\partial E}{\partial \theta_{i,k}} \]

\[ a_{i,j,k}^{(t)} = \eta a_{i,j,k}^{(t-1)} + \beta \frac{\partial E}{\partial a_{i,j,k}} \]

Other variables in the activation function (\( b1, a2, a3, \) and \( b3 \)) are adjusted in the same way as (8).

Now, if we define

\[ \frac{\partial E}{\partial I_{j,k}} = \zeta_{j,k} \]

\[ \frac{\partial E}{\partial O_{j,k}} = \xi_{j,k} \]

Then the partial derivatives of \( E \) with respect to adjustable parameters are as follows:

\[ \frac{\partial E}{\partial w_{i,j,k}} = \frac{\partial E}{\partial I_{j,k}} \frac{\partial I_{j,k}}{\partial w_{i,j,k}} = \zeta_{j,k} \cdot O_{j,k} \]

\[ \frac{\partial E}{\partial \theta_{i,k}} = \frac{\partial E}{\partial I_{j,k}} \frac{\partial I_{j,k}}{\partial \theta_{i,k}} = -\zeta_{i,k} \]

\[ \frac{\partial E}{\partial a_{i,k}} = \frac{\partial E}{\partial O_{j,k}} \frac{\partial O_{j,k}}{\partial a_{i,k}} = \xi_{i,k} \cdot \sin(b1_{i,k} \cdot I_{i,k}) \]

\[ \frac{\partial E}{\partial b1_{i,k}} = \frac{\partial E}{\partial O_{j,k}} \frac{\partial O_{j,k}}{\partial b1_{i,k}} = \xi_{i,k} \cdot a1_{i,k} \cdot I_{i,k} \cdot \cos(b1_{i,k} \cdot I_{i,k}) \]

\[ \frac{\partial E}{\partial b2_{i,k}} = \frac{\partial E}{\partial O_{j,k}} \frac{\partial O_{j,k}}{\partial b2_{i,k}} = -\xi_{i,k} \cdot a2_{i,k} \cdot I_{i,k} \cdot e^{-b2_{i,k} \cdot I_{i,k}} \]

\[ \frac{\partial E}{\partial b3_{i,k}} = \frac{\partial E}{\partial O_{j,k}} \frac{\partial O_{j,k}}{\partial b3_{i,k}} = \xi_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot I_{i,k}}} \]

\[ \frac{\partial E}{\partial b4_{i,k}} = \frac{\partial E}{\partial O_{j,k}} \frac{\partial O_{j,k}}{\partial b4_{i,k}} = \xi_{i,k} \cdot \frac{a2_{i,k} \cdot I_{i,k} \cdot e^{-b2_{i,k} \cdot I_{i,k}}}{1 + e^{-b3_{i,k} \cdot I_{i,k}}} \]

And for (9) and (10) the following equations can be computed:

\[ \zeta_{i,k} = \frac{\partial E}{\partial I_{j,k}} = \frac{\partial E}{\partial O_{j,k}} \frac{\partial O_{j,k}}{\partial I_{j,k}} = \xi_{i,k} \cdot \frac{\partial O_{j,k}}{\partial I_{j,k}} \]

while

\[ \frac{\partial O_{j,k}}{\partial I_{j,k}} = a1_{i,k} \cdot b1_{i,k} \cdot \cos(b1_{i,k} \cdot I_{i,k}) \]

\[ -a2_{i,k} \cdot b2_{i,k} \cdot e^{-b2_{i,k} \cdot I_{i,k}} + \frac{a3_{i,k} \cdot b3_{i,k} \cdot e^{-b3_{i,k} \cdot I_{i,k}}}{(1 + e^{-b3_{i,k} \cdot I_{i,k}})} \]

and

\[ \zeta_{i,k} = \begin{cases} \sum_{j,k} w_{j,k} O_{j,k} - d_i, & \text{if } k < l; \\ O_{j,k} - d_i, & \text{if } k = l. \end{cases} \]

4 RESULTS

A graphic interface has been designed for the ANSER-plus software using Motif. The components of Graphic interface are as given in figure 2. Using the ANSER-plus system with the NANN models has tested the heavy convective rainfall estimating. Since rainfall estimation is a complex, nonlinear, discontinuous task, NANN models are able to perform this well; simple neural network models are inaccurate at points of discontinuity. It has been demonstrated that by using artificial NANN models, rainfall estimation can be performed ten times faster (compared with decision tree techniques). Moreover, the average
neural network with function shape autotuning,

*Neural Networks*, 9(4), 1996, 627-641.


### Table 1 Satellite-derived Precipitation Estimates

<table>
<thead>
<tr>
<th>No.</th>
<th>Date</th>
<th>Location</th>
<th>Observation (Inch)</th>
<th>X/S Error %</th>
<th>MLP Error %</th>
<th>NANN Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 #</td>
<td>07/19/85</td>
<td>NY</td>
<td>6.0</td>
<td>+0.83</td>
<td>+14.0</td>
<td>+5.84</td>
</tr>
<tr>
<td>2 *</td>
<td>05/16/86</td>
<td>IL, MO</td>
<td>7.0</td>
<td>+10.0</td>
<td>-5.52</td>
<td>-4.28</td>
</tr>
<tr>
<td>3 *</td>
<td>08/04/82</td>
<td>WI</td>
<td>5.3</td>
<td>+62.08</td>
<td>-8.30</td>
<td>-3.27</td>
</tr>
<tr>
<td>4 *</td>
<td>05/03/87</td>
<td>MO</td>
<td>7.0</td>
<td>+47.86</td>
<td>-10.39</td>
<td>-5.26</td>
</tr>
<tr>
<td>5 *</td>
<td>05/26/87</td>
<td>IA</td>
<td>5.8</td>
<td>-18.62</td>
<td>-7.08</td>
<td>-2.89</td>
</tr>
<tr>
<td>6 #</td>
<td>08/13/87</td>
<td>KS</td>
<td>12.2</td>
<td>+16.62</td>
<td>+15.25</td>
<td>+7.28</td>
</tr>
<tr>
<td>7 #</td>
<td>08/12/87</td>
<td>KS</td>
<td>8.7</td>
<td>-30.57</td>
<td>-21.0</td>
<td>-16.32</td>
</tr>
<tr>
<td>8 #</td>
<td>07/19/85</td>
<td>IA</td>
<td>9.5</td>
<td>+96.21</td>
<td>+18.2</td>
<td>-17.28</td>
</tr>
<tr>
<td>9 #</td>
<td>08/22/85</td>
<td>KS</td>
<td>4.2</td>
<td>-22.86</td>
<td>-13.3</td>
<td>+10.81</td>
</tr>
<tr>
<td>10#</td>
<td>07/02/83</td>
<td>IL</td>
<td>5.2</td>
<td>+13.65</td>
<td>+18.1</td>
<td>+12.24</td>
</tr>
<tr>
<td>11#</td>
<td>05/01/83</td>
<td>MO</td>
<td>6.0</td>
<td>-31.17</td>
<td>+4.0</td>
<td>+2.62</td>
</tr>
<tr>
<td>12#</td>
<td>05/27/85</td>
<td>MO</td>
<td>4.2</td>
<td>+2.62</td>
<td>-18.5</td>
<td>-12.17</td>
</tr>
<tr>
<td>13#</td>
<td>09/05/85</td>
<td>KS</td>
<td>6.2</td>
<td>-5.16</td>
<td>-1.90</td>
<td>-0.78</td>
</tr>
<tr>
<td>14 #</td>
<td>07/15/84</td>
<td>IA</td>
<td>5.0</td>
<td>+78.4</td>
<td>+8.80</td>
<td>+6.20</td>
</tr>
<tr>
<td>15 #</td>
<td>07/16/84</td>
<td>TN</td>
<td>4.2</td>
<td>+19.52</td>
<td>+6.43</td>
<td>+3.24</td>
</tr>
</tbody>
</table>

X/S: Xie/Scofield's study; MLP: Multi Layer Perceptron as basic reasoning neural network
NANN: Neuron-Adaptive Neural Networks used in ANSER system
*: Training cases for training neural network(s); #: Test cases

Figure 2 Graphic User Interface of ANSER-Plus Expert System