

**A STOCK PRICE PREDICTION MODEL BY
THE NEURAL NETWORK APPROACH**



**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(COMPUTER SCIENCE)
FACULTY OF GRADUATE STUDIES
MAHIDOL UNIVERSITY
2003**


**ISBN 974-04-3338-3
COPYRIGHT OF MAHIDOL UNIVERSITY**

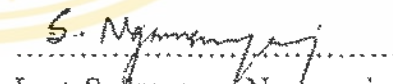
Thesis
Entitled

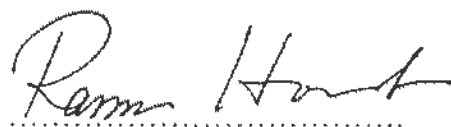
A STOCK PRICE PREDICTION MODEL BY
THE NEURAL NETWORK APPROACH

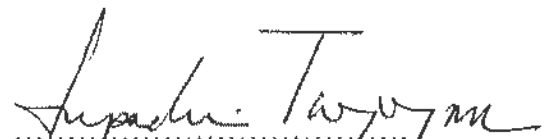


Prapaphan Pan-O
Miss Prapaphan Pan-O
Candidate


Assoc.Prof. Supachai Tangwongsan, Ph.D.
Major-Advisor


Lect. Sudsanguan Ngamsuriyaroj, Ph.D.
Co-Advisor


Assoc.Prof. Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies

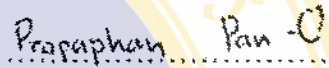

Assoc.Prof. Supachai Tangwongsan, Ph.D.
Chair
Master of Science Programme
in Computer Science
Faculty of Science

Thesis
Entitled

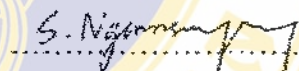
A STOCK PRICE PREDICTION MODEL BY THE NEURAL NETWORK APPROACH

was submitted to the Faculty of Graduate Studies, Mahidol University
For the degree of Master of Science (Computer Science)

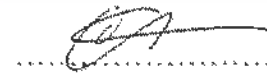
on
May 9, 2003

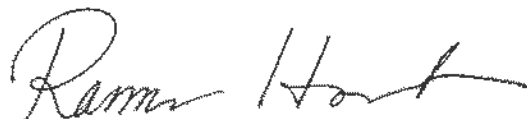

Miss Prapaphan Pan-O
Candidate



Assoc.Prof. Supachai Tangwongsan, Ph.D.
Chair


Lect. Sudsanguan Ngamsuriyaroj, Ph.D.
Member


Asst.Prof. Chomtip Pornpanomchai, Ph.D.
Member


Assoc.Prof. Chinda Achariyakul, Ph.D.
Member


Assoc.Prof. Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies
Mahidol University


Prof. Amaret Bhumiratana, Ph.D.
Dean
Faculty of Science
Mahidol University

ACKNOWLEDGEMENT

I am most grateful to my thesis advisor, Dr. Supachai Tangwongsan, for his guidance, invaluable suggestions and helpful discussions given to me throughout the course of this thesis.

I am also grateful to Lect. Sudsanguan Ngamsuriyaroj, Asst.Prof. Chomtip Pornpanomchai and Assoc.Prof. Chinda Achariyakul for their invaluable comments and for serving as my thesis committee.

My special thanks are extended to Mr. Pornchai Po-Aramsri and Mr. Ananta Srisuphab for their academic and personal advices. I also would like to thank my good friend, Ms. Keson Skulwong, Ms. Benjamas Limprasert and many friends at Inetasia Holding Limited for their love and help especially Ms. Nuttika Suppasilkanok and Mr. Kriengsak Sae-hiw.

Finally, I am grateful to my family and Mr. Somboon Tiewchareonchai for their constant love, encouragement and support. I would like to dedicate this thesis to my father, my mother and all the teachers who have taught me since my childhood.

Prapaphan Pan-O

A STOCK PRICE PREDICTION MODEL BY THE NEURAL NETWORK APPROACH

PRAPAPHAN PAN-O 4137586 SCCS/M

M.Sc. (COMPUTER SCIENCE)

THESIS ADVISORS: SUPACHAI TANGWONGSAN, Ph.D., SUDSANGUAN
NGAMSURIYAROJ, Ph.D.

ABSTRACT

This research proposes a stock price prediction based on the backpropagation neural network approach. The focus of this scheme is to adjust the neural network training methodology for improving the accuracy of both value prediction and value fluctuation direction of the desired value. The predicted results of the traditional learning methodology and those derived from using the approach will be compared.

The process used to train the network is to feed a large number of inputs to the network in order to reach the minimum mean squared error between the desired value and the actual output in each training cycle. This process may give the predicted value close to the desired value. We proposed the method to adapt the training methodology using the *Two-Step Continuous Fluctuation Direction (TSCFD)* of the desired value for calculating the changing weight. To perform the prediction task, the proposed system is tested using the time series data from 3 sources: namely, 2 closing stock prices with 500 trading days and a generated data series with 487 observations from Mackey-Glass Equation.

The general prediction measurements such as *Mean Squared Error (MSE)*, *Mean Absolute Error (MAE)* and *Mean Absolute Percentage Error (MAPE)* are used to determine how close the desired value is. Additionally, we use the *Prediction Of Correct Fluctuation Direction (POCFD)* to indicate the correctness of the predicted fluctuation direction while *Tolerance1%* and *Tolerance5%* are used to determine how close it is for the specified range of desired value. The model performance is measured with various settings of network parameters and topologies. The average of experimental results derived from using the TSCFD for predicting the next value point only by using 20-2-1 network can reach 75%, 90% and 80% accuracy by the *Tolerance1%*, *Tolerance5%* and *POCFD*, respectively. In conclusion, it was found that the experiments yielded quite satisfactory results and the research objectives were achieved.

KEY WORDS: NEURAL NETWORK / BACKPROPAGATION / PREDICTION /
STOCK PRICE / DIRECTION

118 P. ISBN 974-04-3338-3

ต้นแบบการพยากรณ์ราคาหุ้นในตลาดหลักทรัพย์ ด้วยวิธีของโครงข่ายนิวรอล (A STOCK PRICE PREDICTION MODEL BY THE NEURAL NETWORK APPROACH)

ประเภทพรรณ ปันไอ้ 4137586 SCCS/M

วท.ม. (วิทยาการคอมพิวเตอร์) สาขาวิชาเอกวิทยาการคอมพิวเตอร์

คณะกรรมการควบคุมวิทยานิพนธ์ : ศุภชัย ตั้งวงศ์สานต์, Ph.D., สุกสงวน งามสุริยโรจน์, Ph.D.

บทคัดย่อ

งานวิจัยนี้ได้นำเสนอต้นแบบการพยากรณ์ราคาหุ้นในตลาดหลักทรัพย์ด้วยวิธีการของโครงข่ายนิวรอล ซึ่งวิธีการที่นำเสนอ มุ่งมองที่การปรับปรุงวิธีการสอนโครงข่ายนิวรอลให้เกิดการเรียนรู้เพื่อให้เกิดความถูกต้องของผลการพยากรณ์ที่มากขึ้นในทั้ง 2 ด้านด้วยกัน คือ ความถูกต้องของค่าที่พยากรณ์ได้จริง และความถูกต้องของทิศทางการขึ้นลงของค่าที่ได้เหล่านั้น แนวทางการปรับปรุงกระบวนการเรียนรู้ของโครงข่ายนิวรอลที่ได้นำเสนอและวิธีการสอนโครงข่ายแบบเดิม ๆ จะถูกทำการเปรียบเทียบถึงผลที่เกิดขึ้น

การสอนโครงข่ายนิวรอลทำโดยการป้อนอินพุตจำนวนมากเข้าสู่โครงข่ายเพื่อให้เกิดการเรียนรู้ โดยการพยายามที่จะลดขนาดของค่ามัธยฐานของความผิดพลาดที่เกิดขึ้นยกกำลังสอง (Mean Squared Error) จากผลต่างระหว่างค่าคาดหวังกับค่าที่พยากรณ์ได้จริงระหว่างการสอนในแต่ละรอบ การสอนโครงข่ายแบบนี้เป็นเพียงความพยายามที่จะทำให้ได้ค่าที่จากการพยากรณ์ใกล้เคียงกับค่าคาดหวังมากที่สุด ดังนั้นเราจึงได้นำเสนอแนวทางการปรับเปลี่ยนรูปแบบการสอนโครงข่ายโดยมีการนำความต่อเนื่องของการเปลี่ยนแปลงทิศทางการขึ้น-ลงของค่าคาดหวังมาใช้ในการปรับเปลี่ยนวิธีการหาขนาดของความคลาด เพื่อจะนำไปปรับเปลี่ยนค่า weight เราเรียกวิธีการนี้ว่า *Two-Step Continuous Fluctuation Direction (TSCFD)* ซึ่งแนวทางการแก้ปัญหาที่นำเสนอนี้จะถูกทดสอบประสิทธิภาพโดยใช้แบบจำลองนี้ไปพยากรณ์ค่าของชุดข้อมูลทดสอบที่อยู่ในรูปแบบของอนุกรมเวลา จาก 3 แหล่งคือ ข้อมูลราคาปิดของหุ้น 2 ตัวในตลาดหลักทรัพย์ และข้อมูลที่ได้จำลองขึ้นจาก Mackey-Glass Equation

ตัววัดความถูกต้องที่ใช้ในการประเมินผลของระบบโดยทั่วไป มุ่งมองเฉพาะค่าความใกล้เคียงระหว่างค่าคาดหวังกับค่าที่พยากรณ์ได้เท่านั้น เช่น *MSE*, *MAE*, *MAPE* และนอกจากตัววัดดังกล่าวแล้วตัววัดที่เรียกว่า *POCFD* ซึ่งใช้วัดความถูกต้องของทิศทางการขึ้น-ลงของข้อมูล และตัววัดความถูกต้องของค่าความใกล้เคียง (*Tolerance*) ในช่วง 1% และ 5% ได้ถูกนำมาใช้ร่วมด้วย การทดสอบประสิทธิภาพของการทำงานของโครงข่ายนิวรอลทำโดยการตั้งค่าพารามิเตอร์และใช้โครงสร้างของโครงข่าย (network topology) ที่หลากหลาย ผลการทดลองที่ได้จากการเลือกใช้รูปแบบการสอนโครงข่ายแบบ TSCFD เพื่อใช้พยากรณ์ข้อมูลในอีก 1 จุดข้างหน้าสำหรับโครงข่ายแบบ 20-2-1 ให้ผลของ *Tolerance1%*, *Tolerance5%* และ *POCFD* สำหรับข้อมูลทดสอบจากทั้ง 3 แหล่งเฉลี่ยแล้วประมาณ 75%, 90% และ 80% ตามลำดับ โดยสรุปพบว่าผลการทดลองที่ได้เป็นที่น่าพอใจ และครอบคลุมวัตถุประสงค์ของงานวิจัย

CONTENTS

	Page
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER	
I INTRODUCTION	1
II PROBLEM STATEMENT	4
2.1 Motivations	4
2.2 Problem Identification	5
2.3 Literature Survey	6
2.3.1 Flaws of Technical Analysis	7
2.3.2 Time Series Prediction	8
2.3.3 Neural Network for Time Series Prediction	10
2.3.3.1 Limitations and Problems	11
2.3.3.2 Financial Prediction Modeling	12
2.3.3.3 Approaches to Stock Price Prediction	13
2.4 Objectives	18
2.5 Problem Scope	19
III APPROACH	20
3.1 Neural Network for Time Series Prediction	20
3.1.1 Training Methodology	21
3.1.2 Learning by Minimizing an Error	25
3.2 Prediction Strategies	27
3.2.1 Step-Training Method	27
3.2.2 The TSCFD Cost Function Learning Approach	30
3.3 Model Evaluation	34
IV SYSTEM DESIGN AND IMPLEMENTATION	39
4.1 System Overview	39
4.2 Structure Chart	40
4.2.1 Data Collection	41
4.2.2 Training	42
4.2.2.1 Network Setup	44
4.2.2.2 Data Preprocessing	48
4.2.2.3 Network Learning	49
4.2.2.4 Training Result Saving	51

CONTENTS (CONT.)

	Page
4.2.3 Testing	51
4.2.3.1 Pattern Loading	51
4.2.3.2 Performance Testing	52
4.2.3.3 Testing Result Saving	52
4.2.3.4 Error Reporting	52
4.3 Data Flow Diagrams	52
4.4 Module Specifications	57
4.5 File Structure	63
4.6 Input-Output Design	67
4.6.1 Training Screen Design	67
4.6.2 Testing Screen Design	67
4.7 System Tools and Resources	70
V EXPERIMENTAL RESULTS	72
5.1 Data Sources	72
5.2 Network Setting	77
5.3 Description of Experimental Results	78
5.3.1 Reports Classification	78
5.3.2 Phraseology Expression	80
5.4 Results	81
5.4.1 Source: Mackey-Glass Time Series	81
5.4.2 Source: Ahold Stock Price	86
5.4.3 Source: IBM Stock Price	92
VI DISCUSSION AND CONCLUSION	98
6.1 Discussion	98
6.1.1 Problems and Difficulties	98
6.1.2 Performance	99
6.1.2.1 Prediction Accuracy	99
6.1.2.2 Response Time	99
6.1.2.3 Storage Usage	100
6.1.2.4 Program Complexity	101
6.1.3 Limitations	101
6.2 Conclusion	101
VII SUGGESTIONS FOR FUTURE WORK	103
REFERENCES	104
APPENDIX BACKPROPAGATION DERIVATION	108
BIOGRAPHY	118

LIST OF TABLES

	Page
Table 5.1 Configurations and Error Summary Appling Traditional Training Method with a Traditional Cost Function for the Mackey-Glass Time Series	82
Table 5.2 Configurations and Error Summary Appling Step-Training Method with a Traditional Cost Function for the Mackey-Glass Time Series	83
Table 5.3 Configurations and Error Summary Appling Step-Training Method with a TSCFD Cost Function for the Mackey-Glass Time Series	84
Table 5.4 The Mackey-Glass Time Series Performances with Varying Training Size, Appling Step-Training Method with a TSCFD Cost Function	85
Table 5.5 Configurations and Error Summary Appling Traditional Training Method with the Traditional Cost Function for an Ahold Data Set	87
Table 5.6 Configurations and Error Summary Appling Step-Training Method with a Traditional Cost Function for the Ahold Data Set	88
Table 5.7 Configurations and Error Summary Appling Step-Training Method with a TSCFD Cost Function for the Ahold Data Set	90
Table 5.8 The Ahold Data Series Performances with Varying Training Size, Appling Step-Training Method with a TSCFD Cost Function	91
Table 5.9 Configurations and Error Summary Appling Traditional Training Method with the Traditional Cost Function for the IBM Data Set	92
Table 5.10 Configurations and Error Summary Appling Step-Training Method with a Traditional Cost Function for the IBM Data Set	94
Table 5.11 Configurations and Error Summary Appling Step-Training Method with a TSCFD Cost Function for the IBM Data Set	95
Table 5.12 The IBM Data Series Performances with Varying Training Size, Appling Step-Training Method with a TSCFD Cost Function	96

LIST OF FIGURES

	Page
Figure 3.1 Fully Connected Feedforward Backpropagation Neural Network with Three Layers	21
Figure 3.2 Processing Elements	23
Figure 3.3 Gradient Descent - Error and Weight	26
Figure 3.4 Traditional Training with 10 Moving Window Size by Using the Training Set of 700 Samples	28
Figure 3.5 Step-Training with 10 Moving Window Size by Using the Training Set of 30 Samples	30
Figure 3.6 Probably Cases of the Desired Value	32
Figure 3.7 The Prediction of Change in Direction	36
Figure 3.8 The Prediction of Correct Fluctuation Direction	37
Figure 4.1 A High-Level Diagram of the System	40
Figure 4.2 Structure Chart of the Neural Network for a Stock Price Prediction	41
Figure 4.3 Data Collection	41
Figure 4.4 Training a Set of Data	43
Figure 4.5 Network Setup	44
Figure 4.6 Sigmoid Function	47
Figure 4.7 Data Preprocessing	48
Figure 4.8 Network Learning	50
Figure 4.9 Context Diagram: A Stock Price Prediction Model by the Neural Network Approach	53
Figure 4.10 Data Flow Diagram Level 1: A Stock Price Prediction Model by the Neural Network Approach	53
Figure 4.11 Data Flow Diagram Level 2: Network Training	54
Figure 4.12 Data Flow Diagram Level 2: Network Testing	54
Figure 4.13 Training Screen	68
Figure 4.14 Selecting a Pattern Screen	68
Figure 4.15 Calendar of Training Screen	68
Figure 4.16 Time Consume Report of Training Screen	69
Figure 4.17 Testing Screen	69
Figure 4.18 Testing Screen after Performing the Prediction	70
Figure 5.1 Generated Data of Mackey-Glass Time Series	75
Figure 5.2 Daily Price of Ahold	76
Figure 5.4 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.1	82
Figure 5.5 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.2	83
Figure 5.6 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.3	85

LIST OF FIGURES (CONT.)

	Page
Figure 5.7 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.4, Trained on a Training Size of 50 Samples	86
Figure 5.8 The Error Value of the 20-2-1 Network Selected from Table 5.5	87
Figure 5.9 The Error Value of the 20-2-1 Network Selected from Table 5.6	89
Figure 5.10 The Error Value of the 20-2-1 Network Selected from Table 5.7	90
Figure 5.11 The Error Value of the 20-2-1 Network Selected from Table 5.8, Trained on a Training Size of 25 Samples	91
Figure 5.12 The Error Value of the 20-2-1 Network Selected from Table 5.9	93
Figure 5.13 The Error Value of the 20-2-1 Network Selected from Table 5.10	94
Figure 5.14 The Error Value of the 20-2-1 Network Selected from Table 5.11	96
Figure 5.15 The Error Value of the 20-2-1 Network Selected from Table 5.12, Trained on a Training Size of 25 Samples	97
Figure A.1 Multilayer Perceptron: A Standard Feedforward Neural Network	108
Figure A.2 All Output y_k Depend on each Weight w_{ij} Between the Input and the Hidden Layer	113

CHAPTER I

INTRODUCTION

Time series are sets of observations or measurements over time, where their parameters or embedded properties are unknown. The lack of a prior knowledge is a very realistic situation in the real world problems and this is where the challenges come through. Time series analysis is now very widely used in many fields of engineering, biomedical diagnosis, physical science, finance and economics. Domains in which the time series analysis could apply to include: weather forecasting, economic growth prediction, which have been developed for analyzing and predicting time series i.e. the historical data is used to predict closing price of an individual stock market.

Neural networks that serve as powerful computational frameworks have gained much of popularity in business applications as well as in computer science and psychology. Neural networks have been successfully applied to the time series prediction, the classification analysis, and many other complex pattern recognition problems. In financial applications, neural networks are being used to forecast markets with an individual stock movement, international currency and commodities, among many others. It can be seen that they are beneficial as financially forecasting tools. In this regard, backpropagation neural network is one of the algorithms often used to perform this task.

In this research, we have applied the neural network to predict a series of a stock price with goals for improving its accuracy including magnitude (exactly predicted values) and their fluctuation direction, focusing to the neural network training and learning methodology. Two individual stocks with non-stationary and a Mackey-Glass time series are applied to simulate a prediction model. The future values are predicted by the historical data with the assumption that the future events are the same as the past events. The feedforward backpropagation neural network is

used as a tool in manifold hard-to-forecast data. Furthermore, a large number of predictive measurements have been applied to neural network model for comparison.

The distinctions between the present experiment and those previous ones as in the same area are focused on the following issues:

Firstly, we apply our model by using the time-window moving for our training methodology, which is named as the step-training method. The concept of this method is to use the training set in a short time period for predicting the next value point only.

Secondly, the aim is to focus on the accuracy of a value fluctuation direction, predicting the change in direction (up or down) corresponding to the prediction of the actual value that belongs to the data sets. Therefore, the proposed approach is to use a fluctuation direction adjusted weight instead of using the traditional cost function.

We conduct a series of experiments to evaluate the ability of a feedforward backpropagation neural network to adjust a set of weights in order to minimize its mean squared error (traditional cost function) and the two-step continuous direction movement driven based on the traditional cost function on both the traditional training and the step-training methods. Three kinds of data source have been used, namely, two sets of actual stock price and one from simulated data.

An overview of the thesis outline is given below:

Chapter I Introduction:

Chapter II Problem Statement:

Identifies and describes the motivation, problem statement, literature surveys, which include some relevant reviews on the related work, as well as objective and scope of the research.

Chapter III Approach:

Explains the method applied in neural network for a stock price

prediction. Training methodology and learning algorithm by adjusting the traditional cost function are described.

Chapter IV System Design and Implementation:

Presents detailed design and model development methods, which include system overview, as well as system design and implementation.

Chapter V Experimental Results:

Shows the setup of the experiment and the model's results of forecasting the future values.

Chapter VI Discussion and Conclusion:

Discusses and concludes the study with an analysis of the performance, problem and difficulties that found throughout the development phase.

Chapter VII Suggestions for Future Work:

Presents the way for future work.

CHAPTER II

PROBLEM STATEMENT

This chapter discusses the motivations, the problem identification and literature surveys that are relevant to the present work. The objectives and the problem scope are also described in this chapter.

2.1 Motivations

In today business, the investment in finance undoubtedly requires the continuous flow of information. Knowledge of basic investment information is suite important to making investment decisions. At present, many finance experts, including investors, use their experiences based on stock market conditions to predict changes in stock market prices. This information is derived from market conditions in the past and the present. In prediction, most of the experts use their own judgment with their own knowledge, experience and the statistical model by applying their knowledge to analyst the data. Problems concerning the prediction of a time series have existed for a very long time. Ever since the creation of the first financial markets such as a stock price prediction, people have tried (and usually failed) to predict the future. This is not quite a good start for any people who would like to invest without the financial knowledge and experiment.

John [2] indicated that millions of dollars flow through the stock markets on a daily basis. Huge sums of money change hands in a frenzy of investor activity to buy stocks with promising futures and sell stocks with disappointing value. Immense fortunes can be made by those who are able to anticipate stock behavior and take acting accordingly. The opportunities in the stock market are endless, but it is not a game without risk. Few will ever achieve the success of a Warren Buffet, and many are not willing to take the risks on their own and instead entrust their investments to mutual funds, bonds, and other financial institutions with historically better results.

But, what if a normal person, sitting at their home computer, had a tool to help them anticipate the behavior of the stock market. A tool that could effectively aid them in their investment decisions and help them achieve better results than they can currently achieve on their own or through their financial institutions. However, applying the artificial intelligence tools in the prediction process will be a next step. This tool may be a neural network.

The neural network approach has been applied with success to solve prediction problems in many disciplines. The stock price prediction [3,4,5,6,7,8,9,10,11,12,13,14,15,16,17] is no exception. Results have been very promising and research continues in this growing field. Few researchers have explored in detail the neural networks as time series prediction models. But no one has clearly stated the limitations, problems, guidelines and approaches to compare the prediction results by applying neural network models for time series prediction.

This work uses stock prices time series data as a problem set of the prediction model because of the reasons stated above. Since the method used in this research, feedforward backpropagation neural network, has already been proven to increase the value of prediction accuracy, stock price prediction will provide an interesting challenge and further improvement on their prediction accuracies is of great importance.

2.2 Problem Identification

Financial data is an interesting data set not only because it has potential for profits, but also because it is a challenging one. The assumed underlying function is influenced by many external factors. There are an unlimited number of things that can happen to effect the valuation of a company, or the exchange rate of a currency. A political event or war may raise costs or prohibit trade thereby decreasing the profitability of an industry. An early winter may damage a harvest and hurt the economy of an agricultural country, which in turn effects the currency valuation of that country. If the example of a butterfly flapping its wing affecting the weather

pattern far away has any merit, the financial markets are certainly influenced by an innumerable string of influences [7].

In a stock price prediction area, the predicted value of the target may not be quite good enough but the profit earning is desired. This result includes the value prediction accuracy and value fluctuation direction correctness of the target. We are not only concerned about the predictability in terms of the exactly predicted value but we are more interested in the benefit earning which include both the value prediction accuracy and value fluctuation direction of the target.

Authors of neural network based systems use the network's ability to create its own representation of the relationship between influencing factors and stock performance, based on historical market data. The network approach means that, unlike other statistical methods, a network does not require a detailed economic model of the underlying relations to be formulated in advance to be able to create a representation of the non-linear dynamics of market behavior [18].

Although the use of neural networks in financial time series prediction is not particularly novel in itself. The majority of the literature studied for this work seems to follow in a similar vein in placing equal emphasis on the vagaries of the stock market as it does on neural networks, and the thrust is directed merely at proving the concept of the neural network as a valid modeling tool. It should be noted, neural networks are not making decision themselves. Rather, a network is used to make a presentation of possible answers, and then a person uses this presentation of data as an input to make decisions. The output of a neural network may be one of many inputs to a person's decision process.

2.3 Literature Survey

This section described a review on some of the literature that relates more directly to this research, and presents an analysis of the trends of the current work. A brief description of the flaws of technical analyst, overview of advanced trading

technology, time series prediction and neural network for time series prediction will be presented as follows:

Over the last few years, trading technology has evolved from video-display information to digital information feeds, from stand-alone PCs to integrated networks of workstations, from simple graphics to 3-D images of the behavior of financial markets. The next logical step is towards machine learning, which along with related techniques will be used to recognize patterns, to automate trading decisions, and to optimize portfolios. Such techniques enhance performance while reducing the risks of transacting in financial markets.

As classical thinking about the behavior of financial markets came under increasing scrutiny, and as initial results of trading systems based on neural networks, genetic algorithms, and fuzzy logic added fuel to the fire, a storm of criticism arose. Research on non-linear dynamics, chaos and complexity theory, rescaled range analysis, and the fractal market hypothesis led to the questioning of traditional approaches to trading based on fundamental and technical analysis [19].

In the next section, we review some of the major flaws of these traditional approaches for trading in financial markets.

2.3.1 Flaws of Technical Analysis [19]

Traditional technical analysis and trading models based on technical analysis use simplified assumptions to make the design of models less complex and more manageable. Many traditional trading models are simple rule-based systems that use “what if” scenarios. Some of these rule-based systems have fewer than ten rules. The simpler ones use various kinds of moving averages, technical analysis indicators, or other pattern descriptors.

Rule-based systems are static and deal only with symbolic information where inputs, thresholds, and decision-rules change in discrete steps. Such systems have a difficult time dealing with non-linearity. As a consequence, rule-based systems must be frequently fine-tuned according to changing market circumstances. Continual fine-

tuning prohibits accurate performance measurements; relative performance versus relative risk is seldom computed.

Rule-based systems assume that the influences of various factors on one another can be separated and thus that all factors are independent. They filter out dependencies, orthogonalize inputs, or are based on underlying assumptions of a Gaussian distribution of price changes.

The problem with these assumptions is that many factors that affect the behaviour of prices are eliminated because they “interfere” with a deeper understanding of actual behaviour of the underlying system. Furthermore, it is traditional to treat time as unimportant. It is often assumed that when an event perturbs the system, it will revert to equilibrium after a short time span. In other words, one assumes that markets have short-term memory.

It is also assumed that traders react to the same information in a linear fashion, that groups of traders are not prone to fashion, and that in the aggregate all traders are retinal (even if they do not behave rationally individually). In fact, the main assumption is that trader skills and experience are equally distributed. If all information is already discounted in the price, no trader or group of traders can outperform the markets.

2.3.2 Time Series Prediction

Time series forecasting, or time series prediction, takes an existing series of data $x_{t-n}, \dots, x_{t-2}, x_{t-1}, x_t$ and forecasts the x_{t+1}, x_{t+2}, \dots data values. The goal is to observe or model the existing data series to enable future unknown data values to be forecast accurately. Examples of data series include financial data series (stocks, indices, rates, etc.), physically observed data series (sunspots, weather, etc.), and mathematical data series (Fibonacci sequence, integrals of differential equations, etc.). The phrase “time series” generically refers to any data series, whether or not the data are dependent on a certain time increment [5].

Time series prediction encompasses many areas. Some examples are the prediction of weather patterns, horseracing models, and financial market prediction. The goal of time series prediction is to predict the future value of a variable that varies in time based on knowledge of historical data. Typically, the predicted variable is continuous, so that time series prediction is usually a specialized form of regression.

Several *difficulties* can arise when performing time series forecasting [5]. Depending on the type of data series, a particular difficulty may or may not exist.

- The first difficulty is a *limited quantity of data*. With data series that are observed, limited data may be the foremost difficulty. For example, given a company's stock that has been publicly traded for one year, a very limited amount of data are available for use by the forecasting technique.
- The second difficulty is *non-stationary*, data that do not have the same statistical properties (e.g., mean and variance) at each point in time. A simple example of a non-stationary series is the Fibonacci sequence: at every step the sequence takes on a new, higher mean value.
- The third difficulty is *forecasting technique selection*. From statistics to artificial intelligence, there are myriad choices of techniques. One of the simplest techniques is to search a data series for similar past events and use the matches to make a forecast. One of the most complex techniques is to train a model on the series and use the model to make a forecast. Neural networks are examples of the techniques.

The beginning of *traditional time series analysis* might have been in the end of the 20's when Yule invented the autoregressive method (AR) in order to predict sunspots. The general AR-model expresses future values of the time series as a linear combination of past values plus a random noise component:

$$y(t) = \sum_{m=1}^d a_m y(t-m) + e(t) \quad (2.1)$$

Another common model for time series analysis is the moving average model (MA)

$$y(t) = \sum_{n=1}^M b_n e(t-n) \quad (2.2)$$

The above equation describes a situation where the time series y is controlled by an external time series e in a linear and deterministic relation

Combining the two equations yields the Auto Regressive and Moving Average (ARMA) model, which dominated the time series analysis for more than 50 years [14]. The classic (1970s) time series prediction approach uses a Box Jenkins algorithm to transform the data from a stationary process into ARMA. The method allows estimation of actual pulses or inclusion of forecast pulses that do not fit the ARMA methodology. A tournament is used to select the best fit from 2 different possible models. Iteration on the parameters is then used to find increasingly better solutions.

2.3.3 Neural Network for Time Series Prediction

There are generally two types of prediction models, linear and non-linear. Non-linear models are wide-ranging and are more difficult to specify. Within the non-linear category, there are both parametric and non-parametric models. Neural networks are non-parametric. This means that the underlying function, which is non-linear, is not prescribed, nor is it explicitly predicted.

Neural networks are one of the most innovative analytical tools to surface in the financial arena. A neural network is an excellent survey approach for those new to neural network applications in the finance industry. A variety of neural network approaches have been applied to the prediction of non-linear time series such as an individual stock price prediction.

2.3.3.1 Limitations and Problems

There are limitations to the feedforward, backpropagation architecture. Backpropagation requires lots of supervised training, with lots of input-output examples. Additionally, the internal mapping procedures are not well understood, and there is no guarantee that the system will converge to an acceptable solution. At times, the learning gets stuck in a local minima, limiting the best solution. This occurs when the network system finds an error that is lower than the surrounding possibilities but does not finally get to the smallest possible error.

Neural networks [7] have often been applied to time-series prediction problems. The ability of neural networks to find patterns in highly dimensional data gives them an advantage over other methods applied to this domain. Time series problems are similar to regression problems. They represent models that perform function approximation in order to identify underlying functions in a set of data. The goal is to predict a future value of data given a set of previous values. For example, at time τ , predict the value of $x_{\tau+1}$. The values that are used to make this prediction are x_{τ} , $x_{\tau-1}$, $x_{\tau-2}$, ... $x_{\tau-n}$.

In order for time series prediction to be effective, a pattern must be present for the data. This pattern is an underlying function that is often affected by external influences. Influences on a function will create noise that makes the functions harder to isolate and identify. The removal of noise and identification of patterns are the goal of applying neural networks to time series problems.

To construct the neural network models, there are a number of parameters that have to be set before the network can be trained, such as learning rates, momentum factor, training algorithm, preprocessing and network architecture. There are no rules about the parameter settings best suit the type of problem.

2.3.3.2 Financial Prediction Modeling

Financial time-series prediction is a very popular area of research. The potential benefits to be realized from improving financial predictions are enormous. Even a slight improvement of predictions can be very profitable for a company working in this area. There are many types of problems in this domain. Foreign currency exchange rates, market indexes, commodities, and individual stocks are all possible applications. Companies such as investment banks, brokers and private individuals are all spending time trying to find successful solutions. There is much debate as to whether or not it is even possible to make predictions concerning this data.

The use of artificial intelligence and neural networks in particular has become a fast developing area and as such there are a growing number of papers and research projects becoming available. We have investigated several different financial predictions modeling of neural networks as described below.

- *Feedforward backpropagation neural networks* are used in many different types of applications. This architecture has spawned a large class of network types with many different topologies and training methods. Its greatest strength is in non-linear solutions to ill-defined problems.

The typical *backpropagation network* has an input layer, an output layer and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there is just one or two. Some work has been done which indicates that a maximum of four layers (three hidden layers plus an output layer) are required to solve problems of any complexity [19]. Applications of feedforward neural networks can be found in [4,11,12,13,21].

- *Recurrent neural networks* have an architecture where the outputs from intermediate neurons are fed back to the input layer. In this way the network will contain memory of previous values of the

input data. A single value $y(t-1)$ is used as input and a single value $O(t)$ is produced as output from the neural network. The temporal dependencies are modeled in the weights in the feedback loops in the network [14]. Applications of recurrent networks can be found in [21].

- *Modular neural networks* are a system of many separate networks (usually backpropagation), and each of them learns to handle a subset of the complete set of training cases. It is therefore able to improve the performance of backpropagation where the training set can be naturally divided into subsets that correspond to distinct subtasks. The network consists of several networks called “local experts” connected by a gating network that allocates each case into one of the local experts. The output of that local expert is compared to the actual output and the weights are changed locally only for that expert and for the gating network. Applications of modular networks can be found in [7,15,21].

2.3.3.3 Approaches to Stock Price Prediction

Backpropagation is one of the most popular learning algorithms for feedforward networks. Basically, it is a gradient descent technique to minimize some error criteria.

The criteria of forecasting model performance in traditional time series forecasting are cost functions. The cost functions are generally based on the goodness-of-fit between the target and the predicted value. Neural networks especially backpropagation networks are similar to conventional regression estimators except for their non-linearity. Therefore, the cost functions are used in the same way as in regression models to judge the goodness of the model fitting [8].

The researches discussed in this section apply the training and learning techniques to the neural network models. Many improvement strategies have been developed to increase the prediction accuracy. Financial data is most widely used as a data set to test the model.

- Rothermich [7] applies a negatively correlated approach with neural network ensembles to time series prediction of financial index. A neural network ensemble is a collection of neural networks. The ensemble generates a single output by combining the outputs of each of its networks. The general idea of using ensembles is that a team can provide a better result than a single expert.

Negative correlation is a statistical concept that refers to the correlation between two variables. Two variables are negatively correlated if one variable increases as a second variable decrease and vice versa. The goal of applying negative correlation with neural network ensembles is to improve generalization. Rothermich applies a backpropagation algorithm for an ensemble of networks with the negative correlation approach.

Comparisons were made using the Root Mean Squared Error. The results showed that the single network had lower performance than either of the ensemble approaches.

- Op't Landt [4] develops neural networks, suited for stock price prediction, that is, to predict the stock price for a number of companies. The predictions are made using feedforward neural networks. Feedforward neural networks can be used for the prediction of the next value in a time series. The idea of this work is to offer not only the present input, but also $N-1$ previous inputs to the network, together called a "time window". Learning is based on an error measure. Two error measures of MSE and POCID are used to train the network.

This work has run the program using the stock data of Ahold from 1st Jan 1996 to 1st Jan 1997. We use this data set to experiment on our model also.

The conclusion reveals that the model using the MSE as an error measure for training produces slightly better results if we consider the predicted value accuracy produced by the MSE. The model using the POCID error measure for training however performs much better if we consider the percentage of correctly predicted value direction.

- Traditional backpropagation neural networks training criterion is based on goodness-of-fit which is also the most popular criterion for forecasting. However, in the context of financial time series forecasting, we are not only concerned at how good the forecasts fit their target. In order to increase the forecastability in terms of profit earning, Yao *et al.* [8] proposes a profit-based adjusted weight factor for backpropagation network training. Instead of using the traditional least squared error, Yao adds a factor, which contains the profit, direction, and time information to the error function. The error function in this work is Ordinary Least Squared function.

Three models, namely Refense's Discounted Least Squares (DLS) model, Directional Profit (DP) model and Time dependent Directional Profit (TDP) model are implemented to test. They are benchmarked with traditional Ordinary Least Squared (OLS) model. Four major Asian stock market indices (HS, KL, NK, ST, DJ) are applied to these four models.

The results show that this approach does improve the forecastability of neural network model, for the financial application domain.

- Masri et al. [17] focus on the use of recorded time series and some market indicators to estimate future value as a function of the pass values in predicting individual counter price of Kuala Lumpur Stock Exchange (KLSE). They study the effect of using some data representation functions to the prediction results. They introduce the modified returns function that designed to take account historical changing input-output relationship.

They create feedforward, layered and fully connected neural networks. The training algorithm is the standard backpropagation with momentum term. In term of percentage of correct directions, results show that networks which use the modified returns function is 32.52% better than networks that use the standard relative variable difference function.

- Fieldsend [22] develops a neural network training method with multiple and competing objectives, through the framework of Evolutionary Strategy (ES). There are three objective error measures that used, Mean Absolute Percentage Error (MAPE) minimization, Root Mean Squared Error (RMSE) minimization and the maximization of the Percentage Direction Success error measure.

Experimental is used the Santa Fe competition data. Identical networks are trained with different and competing error measures for training multi-layer perceptron. The first set of network is trained by using the standard BP algorithm, the second using the standard Euclidean ES approach. The remaining three sets of networks are trained by using the multi-objective ES methodology, the first with an equally weighted Euclidean/Direction Success (E/D) fitness function, the second with an equally weighted Euclidean / MAPE (E/M) fitness function and

the third with and equally weighted MAPE / Direction Success (M/D) fitness function.

The experiment are trained by using standard backpropagation and standard Euclidean ES approach, shows that neural networks trained with the Direction success error measure alongside other error terms are much more likely to perform better on this error measure when compared with networks trained in the traditional fashion. The results on neural networks trained with MAPE alongside other errors are inconclusive.

There are many approaches that use neural networks for time series prediction in many areas, for example, modular neural networks [7,15,21], recurrent neural networks [21] and feedforward neural networks [3,8,11,12,13,21]. Some of them have been improved and adjusted for the training process approach [8].

So far, there is not much research that focuses on the performance measures including the correct fluctuation direction of the prediction measurement [4]. Generally, the researches that concern the correct fluctuation direction of the predicted data are intended for the financial time series prediction such as stock price prediction, foreign currency exchange rates prediction i.e.. There are also many researches that concentrate on the way to increase the prediction accuracy by optimizing the data pre-processing approach [23,17], using only one input factor (*univariate time series*) to feed the network such as daily closing price [4,23,27], using many financial factors (*multivariate time series*) to feed the network as the input parameters such as high, low, open close daily price, trading volume [17].

We have made a survey on the papers that measure the prediction accuracy in terms of correct fluctuation direction of the target and found that there is a main point that has not been clearly elaborated. Some papers do not clearly describe the way to measure the correct fluctuation direction of the

predicted value while other papers use different ways to measure the correct fluctuation direction [24,25,29].

There are many researches exploring the neural network's problem, guideline, and limitation for time series prediction approach [25] but they do not focus on the way to maximize the benefits to be derived from the network model, which means the maximum prediction accuracy for both the values and fluctuation direction of the target.

The traditional cost function used to train the network, try to reach a least squared error when the neural network is training, may not yield results good enough to capture both the value prediction accuracy and value fluctuation direction correctness of the target. How can we improve the issue concerned?

From the above discussion, we identify four following research problems for existing training and learning methodology:

- How to capture both value prediction accuracy and value fluctuation direction of the target for a stock price prediction?
- How to add the value fluctuation into the training and learning processes?
- How many sizes of the training size are there to minimize the training process?
- Should we use the older data to train the network for a stock price prediction domain?

2.4 Objectives

Based on the afore-mentioned problems, the objectives of this research are as follows:

- 1) To improve the predicted accuracy of a stock price data especially for both the value prediction accuracy and value fluctuation direction of the target.
- 2) To study the development of a stock price prediction model by using the historical data in terms of non-linear prediction.
- 3) To study the neural networks in a stock price prediction domain that covers the feedforward backpropagation neural network, a learning algorithm and cost function.

2.5 Problem Scope

This research has drawn the scope of study and experiment for the sake of feasibility as follows:

- 1) Feedforward backpropagation neural network is used to support the model proposed.
- 2) This research essentially uses the historical data to observe the future value by delimiting the problem. We only concentrate the univariate data set, the set of network input are used as same as the set of network output. The prediction of values is based on a hypothesis that future events are still the same as past events.
- 3) Different performance measures are chosen to evaluate the model. Six quantitative prediction accuracy measurements are calculated and examined in terms of MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and percentages of POCFD. The calculation methods are given in Section 3.3 .

In the next chapter, we will discuss an approach of our model consisting of neural networks for a stock price prediction, prediction strategies and model evaluation.

CHAPTER III

APPROACH

In this chapter, we propose the prediction strategies by adjusting the training and learning methodology of the feedforward backpropagation neural network that is used for the time series prediction. The approach used to handle non-linear time series prediction is to build the models based on short time periods only. The adaptations of the training methodology, cost function and performance measures are considered. We start with an overview of the training methodology in backpropagation neural network, and then explain each technique, which we use in the research. Finally, we describe the way to evaluate the model for fitting our objective.

3.1 Neural Network for Time Series Prediction

One major advantage of applying the neural network is for the forecasting process. It can be trained using an arbitrary amount of data that is available. Furthermore, a neural network is completely free of any bias that might exist with a human. Results of a neural network will not change in any way.

A feedforward neural network that is trained by backpropagation of error throughout a multi-layer network is referred to as a backpropagation neural network. Typical backpropagation network architecture is composed of an input layer and an output layer, usually separated by one or more hidden layers. This arrangement assumes that the layers are fully connected, which means that each neuron in the input layer is linked to each neuron in the hidden layer, with similar connections between the neurons in the hidden and output layers. For time series data, the number of output neuron is fixed to one.

To perform the prediction task, there are some parameters, which we have to use during the training period. The training process takes all these parameters as

input. This process makes the neural network learn. During the training, a neural network is iteratively presented with the training data and the weights in the network are adjusted from time to time till the desired input-output mapping occurs.

There are many factors that will affect the performance of a neural network. The amount of data used to train, the format of the data, the length of time it is trained and various network parameters will have some impacts on its performance. We can say that the training set and the parameters setting are the most significant. The network design issues are determined, and often many experiments are needed to achieve a satisfactory design. The area of training and learning methodology is one such key factor for the development of practical neural network models. That area has inspired us to conduct this research.

This section, a briefly discusses the training methodology and the learning mechanism in backpropagation neural network.

3.1.1 Training Methodology

Typical backpropagation neural network architecture is presented in Figure 3.1

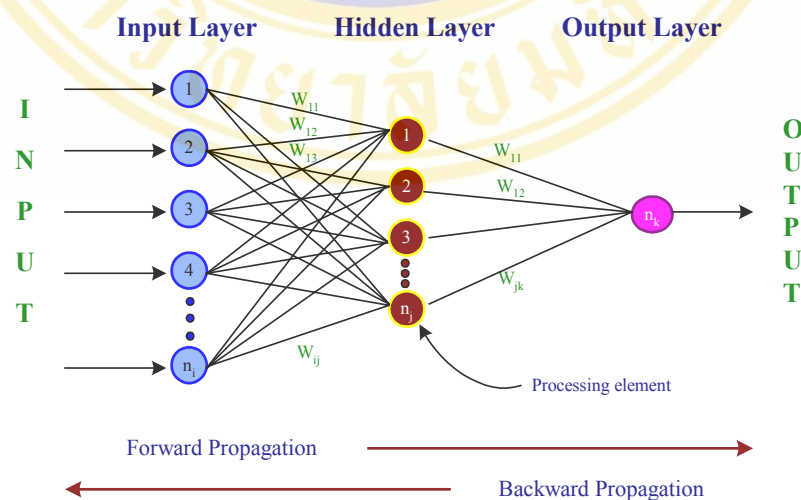


Figure 3.1 Fully Connected Feedforward Backpropagation Neural Network with Three Layers

The training algorithm including the flow of data can be briefly described through the following steps:

- 1) *From the input layer to the hidden layer*: the input vector is presented to the input layer and then sent to the hidden layer.
- 2) *Processing in the hidden layer*: neurons in the hidden layer receive the weighted input and transfer it to the output layer using one of the activation functions.
- 3) *Feedforward computation*: as information propagates through the network, all the summed input and output states are computed in each processing element.
- 4) *Processing in the output layer*: for each processing element, the scaled local error is computed and used to determine the weight increment or decrement.
- 5) *Backpropagation from the output layer back to the hidden layer*: the scaled local error and weights increments or decrements are computed for each layer backward, ending at the hidden layer, and then the weights are updated.

We use three-layer backpropagation neural network to give an explanation of the equations for information processing in the three-layer backpropagation neural network that used in this research. An input vector $I^{(i)} = (i_1, i_2, \dots, i_n)$, is applied to the input layer of the network. The input-layer units distribute the values as $O^{(i)} = (o_1, o_2, \dots, o_n)$ to the hidden-layer units. The process of each processing element can be drawn in Figure 3.2.

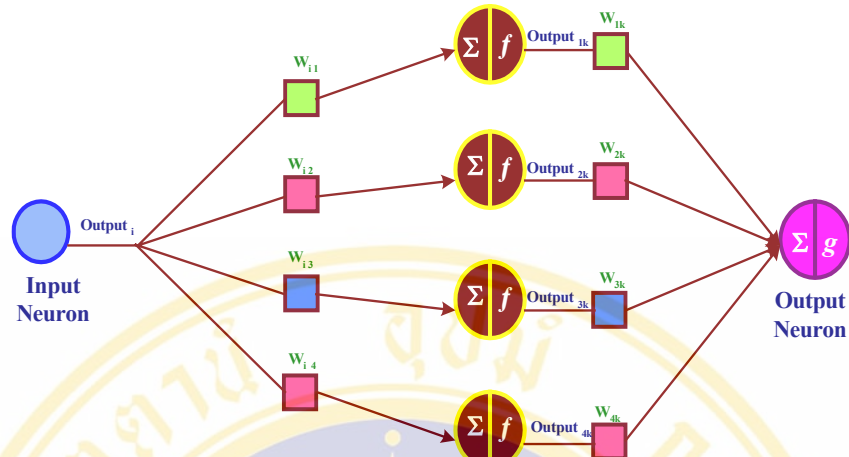


Figure 3.2 Processing Elements

- 1) The network input $i_j^{(h)}$ of j^{th} neuron in hidden layer, h , is calculated by

$$i_j^{(h)} = \sum_{i=1}^N w_{ij}^{(h)} o_i^{(i)} \quad (3.1)$$

- 2) The output value of the j^{th} neuron in hidden layer is denoted by $o_j^{(h)}$. It is calculated with the formula

$$o_j^{(h)} = f_j(net) \quad (3.2)$$

where net refer to $i_j^{(h)}$ and $f_j(net)$ is the activation function for hidden layer.

- 3) The network input $i_k^{(o)}$ of k^{th} neuron in output layer, o , is calculated by

$$i_k^{(o)} = \sum_{j=1}^M w_{jk}^{(o)} o_j^{(h)} \quad (3.3)$$

- 4) The output value of the k^{th} neuron in output layer is denoted by $o_k^{(o)}$. It is calculated with the formula

$$o_k^{(o)} = g_k(net) \quad (3.4)$$

where net refer to $i_k^{(o)}$ and $g_k(net)$ is the activation function for output layer.

- 5) The error term of the output neuron, $\delta_k^{(o)}$, is the current error multiplies by derivative of activation function that defined by

$$\delta_k^{(o)} = (d_k - o_k^{(o)}) \cdot g'_k(net) \quad (3.5)$$

where d_k is the desired value and $g'_k(net)$ is derivative of the activation function for output layer.

- 6) The error term of each neuron in hidden layer, $\delta_j^{(h)}$, is defined by

$$\delta_j^{(h)} = f'_j(net) \cdot \sum_{k=1}^P \delta_k^{(o)} w_{jk}^{(o)} \quad (3.6)$$

where $f'_j(net)$ is derivative of the activation function for hidden layer.

- 7) The weight change in output layer, $\Delta w_{jk}^{(o)}$, and hidden layer, $\Delta w_{ij}^{(h)}$, at time t is calculated by

$$\Delta w_{jk}^{(o)}(t) = \eta \delta_k^{(o)} o_j^{(h)} + \alpha [\Delta w_{jk}^{(o)}(t-1)] \quad (3.7)$$

$$\Delta w_{ij}^{(h)}(t) = \eta \delta_j^{(h)} o_i^{(i)} + \alpha [\Delta w_{ij}^{(h)}(t-1)] \quad (3.8)$$

where η is the learning rate and α is the momentum factor.

- 8) The new weights for training step t are based on the weights at training step $t-1$. The weight update formulas for backpropagation with momentum of output and hidden layer are

$$w_{jk}^{(o,new)} = w_{jk}^{(o)} + \Delta w_{jk}^{(o)}(t) \quad (3.9)$$

$$w_{ij}^{(h,new)} = w_{ij}^{(h)} + \Delta w_{ij}^{(h)}(t) \quad (3.10)$$

In Equation 3.2, $f_j(net)$ is the sigmoid activation function of the hidden neuron. To avoid saturating the activation function, which makes training the network difficult, the training data must be scaled appropriately. Similarly, before training, the set of weights are initialized to appropriately scaled values.

$$f_j(net) = \frac{1}{1 + e^{-net}} \quad (3.11)$$

$$\begin{aligned} f'_j(net) &= net.(1 - net) \\ &= i_j^{(h)}.(1 - i_j^{(h)}) \end{aligned} \quad (3.12)$$

In Equation 3.4, $g_k(net)$ is the linear activation function of the output neuron.

$$g_k(net) = net \quad (3.13)$$

$$g'_k(net) = 1 \quad (3.14)$$

Backpropagation derivation is provided in Appendix A.

3.1.2 Learning by Minimizing an Error

In most learning networks, the difference between the actual output and the desired output values is calculated. This raw error is transformed by the error function to match particular network architecture. The neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error or local error.

The current error is typically propagated backward to the previous layer. The current error is scaled in some manner (often by the derivative of the activation function) that considered with the selected activation function. Normally, this back-propagated value, after being scaled by the cost function, is multiplied against each of the incoming connection weights to modify them before the next learning iteration.

The idea is to calculate an error each time the network is presented with a training vector and to perform a gradient descent on the error considered as function of the weights. There will be a gradient or slope for each weight. Thus, we find the weights, which give the minimal error. The situation is as illustrated in Figure 3.3.

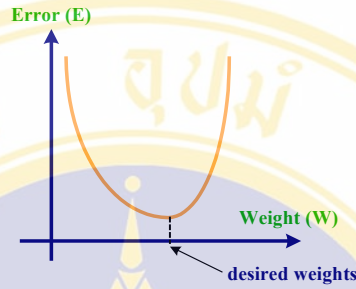


Figure 3.3 Gradient Descent - Error and Weight

Formally, for each pattern p , we assign an error E_p , which is a function of the weights; that is $E_p = (w_1, w_2, w_3, \dots, w_n)$. Typically, this is defined by the difference between the desired value, d , and the output value, o . Thus (for a single node)

$$E_p(w) = \frac{1}{2}(d - o)^2 \quad (3.15)$$

The total error, E , is then just the sum of the pattern errors.

$$E = \sum_p E_p \quad (3.16)$$

Optimizing the weight values to minimize $E_p(w)$ is done via a gradient-descent algorithm, which computes the gradient of the cost function with respect to the current value of the weights. The weight vector is then iteratively updated by adjusting it in the direction in which the error decreases most rapidly. The weight vectors will converge to a point at which E is minimized.

Mean of square error is the stopping criteria that used to stop the training. That means training the neural network on a data set is done by iterative minimization of

the mean of square error that produced during the network is training. The definition is depicted below.

$$MSE = \frac{1}{2N} \sum_{i=1}^N \sum_{k=1}^M (d_k - o_k)^2 \quad (3.17)$$

where the error is over N training patterns of k output node, d_k and o_k are desired value and actual output respectively. In this research, the output layer has one neuron only, and so k is set to 1.

When the mean of squared error is acceptable, the network is reached for the most important acceptance test. The most basic architecture uses this error directly. In our approach, we propose the modified raw error to train the network to fit our specific purpose.

3.2 Prediction Strategies

The previous section provides the traditional training methodology to train the network but in this section, we propose the way to enhance the neural network performance by enforcing the learning networks to meet the objective. This proposal is designed for the time series prediction domain. We wish this yielded many benefits if it were applied to financial prediction modeling, which requires the prediction results for both fluctuation size and direction (up or down) to support the decision making.

3.2.1 Step-Training Method

Once the network has been modeled, the network can be trained on a selected data set. The data set are used for training consists of a set of input vectors and corresponding output values. Training is the way in which neural networks learn. The aim of training is for the network to produce a set of weights. Optimal values of these weights will allow the network to perform the better result.

The training set needs to be quite large as well as contain all necessary information if the network is to learn the features and their relationships that are important. Many researches are investigated in the area of the analysis of the time series (the prediction of future values based on historical information). What will happen if we use the oldest data to perform the value prediction?

Good performance has to come from careful taking of the available training data. The important issue in time series prediction that we are concerned about is “How large a training set needs to be trained?” A long time series is available to train and test the network performance. Large data set is needed but this may a problem of real world data.

For most traditional neural network training to train the network requires a large data set. The network is learning or trying to produce a generalized set of weights and then use this set of weights to project the future value of this training set for a short time period [9,26].

Figure 3.4 shows the example of the traditional training method with the training size of 700 samples and the window size (number of input nodes) is set to 10. For about this configuration, the network will be trained on 689 patterns.

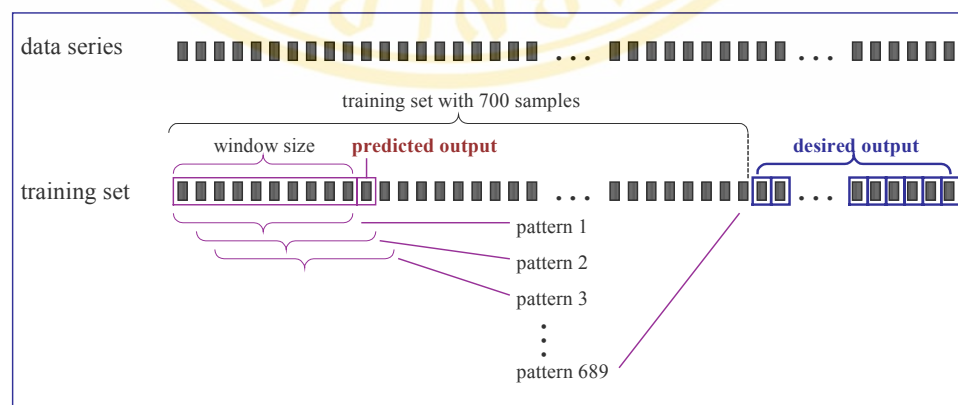


Figure 3.4 Traditional Training with 10 Moving Window Size
by Using the Training Set of 700 Samples

When we train the network with a large data set, the problems may arise. This is one of the problems that we are concerned about in this research. Our hypothesis is that a large training data set does not contain much more useful information but may contain more noise or obsolete information that does not support the current situation.

That is, certain properties of the series may change over time, and looking too far back to the past may find things that are no longer valid. To train the network with short-past data series may produce better results because the neural networks are trained from time to time with the most recent data.

Figure 3.5 presents the example of a training diagram using a training set with short time period. Training set 1 is trained on the historical data with 30 samples; window size is 10 (number of input nodes). During the network training; the time-window is shifted one by one until the network has been trained on all patterns in this training set. The result, set of weights, that we get from the training is the same as the traditional training. But we deploy this set to forecast only one step ahead, d_{t+1} , of the training data set that is used to train. If we would like to forecast d_{t+2} , we need to retrain the network on the updating training set with the latest data. We call this training method is the step-training method.

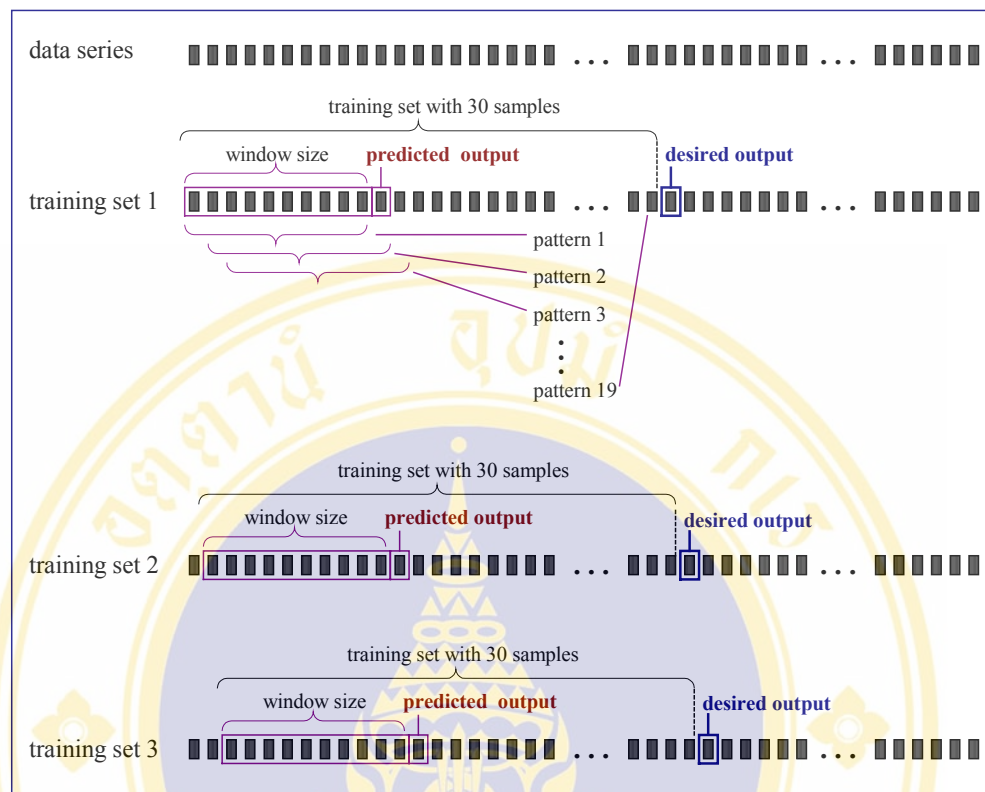


Figure 3.5 Step-Training with 10 Moving Window Size
by Using the Training Set of 30 Samples

3.2.2 The TSCFD Cost Function Learning Approach

During network training, for the input layer, the activation is then propagated to the hidden layers and finally to the output layer of the network. In the next step, the actual output is compared with the desired value. Error values are assigned to each neuron in the output layer, and then these values are propagated back from the output layer to the hidden layers. The weights between the neurons are changed by the backpropagation-learning rule. The neural network learns the input/output mapping by a stepwise change of the weights, and minimizes the difference between the desired and actual output values. The training process involves several iterative steps that modify the weights until the network's output error falls to a certain lowest point, at which the network can be said to have 'learnt' the training set.

It is never possible to make perfect predictions (100% accuracy) or zero mean squared errors. Performance is made depending on the sizes of mistakes. High accuracy (low MSE) may be achieved by a network, which correctly predicts only small moves, which occur most of the time.

If training is done with the traditional cost function, try to minimize the raw error as described in Equation 3.11. It turns out not to be good enough when the performance is measured in terms of the system objective (profit earning includes fluctuation size and direction), and so adjusting cost function will be used.

Many researches in a time series prediction only consider the good predicting result of their target. But if we are concerned the financial time series prediction such as a stock price prediction, an exact value is not the only objective. It is believed that the fluctuation direction of the predicted values is more important than the predicted values themselves.

In order to increase the prediction accuracy in terms of profit earning, fluctuation direction accuracy is used as a measure of the percentage of the time in which the direction of the forward prediction is correct. A neural network may learn more from the error changing. We propose in this research a fluctuation direction adjusted weight instead of using the traditional cost function. We addressed it as a Two-Step Continuous Fluctuation Direction-based raw error value, which we name as TSCFD cost function.

The below figure presents all of the probable cases of the desired value that may happen.

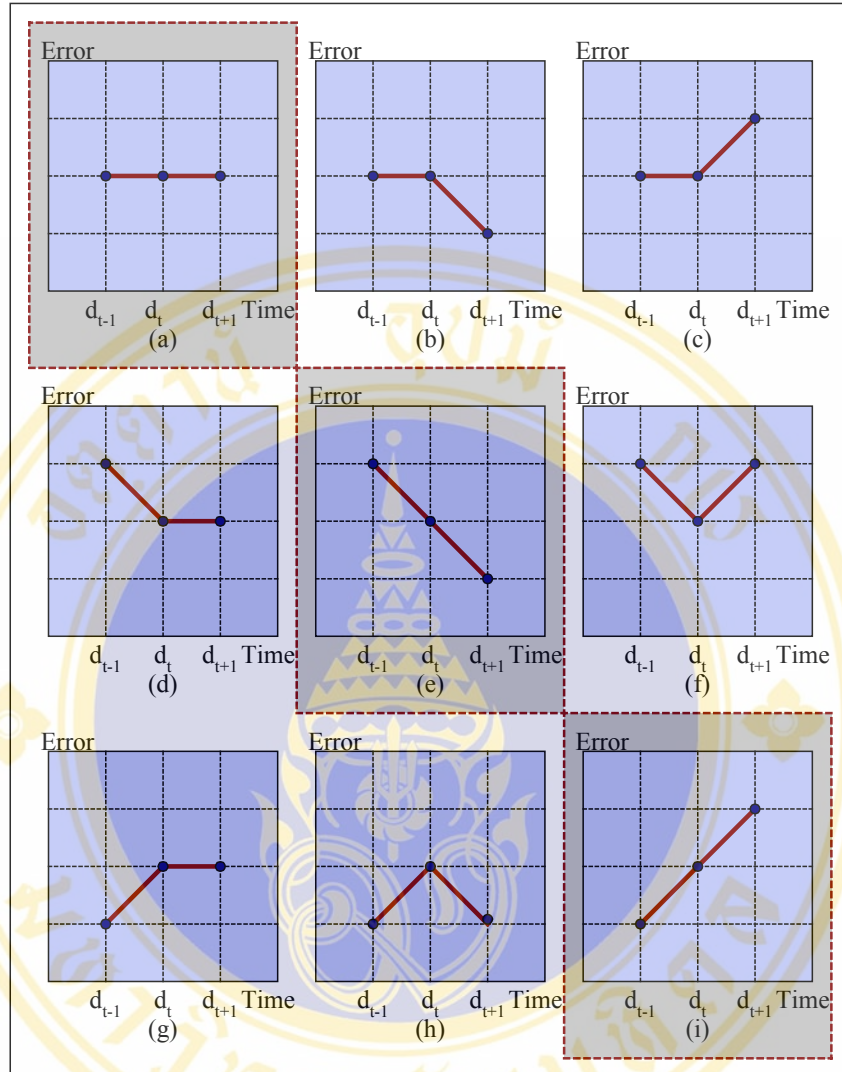


Figure 3.6 Probably Cases of the Desired Value

where d_t is the desired value at time t , current training pattern, and d_{t-1} and d_{t+1} is the desired value of the previous and the next pattern respectively.

Figure 3.6 (a), (e) and (i) are named as two-step continuous fluctuation movement.

In our approach the x value refers to the d_k value that stated in Equation 3.5. That is for delta rule with the two-step continuous fluctuation direction.

$$\delta_k^{(o)} = (x - o_k^{(o)}) \cdot g'_k(\text{net}) \quad (3.18)$$

According to Figure 3.6 (a), the desired values of previous, current and the next patterns are the same. So x value in Equation 3.18 can be set to d_t , which is the same as d_{t+1} .

This rule may be incorporated into a training algorithm, similar to the previous one described in Section 3.1.2, but there is one important thing that is different from the previous one. In Equation 3.18, we activate the two-step continuous fluctuation direction to an error of each neuron in the output layer.

We propose TSCFD cost function to minimize the difference between the desired value and actual output value in the output layer when the error is back propagated through the network for updating the weights. This learning process attempts to activate the direction through the network when the network is trying to minimize the error.

The TSCFD cost function is the error function that tries to maximize the correct direction for the predicted output. When the network is training, we try to minimize the raw error that is produced from the network, which changes the idea to select the desired value that is used for raw error calculation, to another one, referred to x value in Equation 3.18.

As mentioned above, we can define the value of x as shown in Equation 3.19.

$$x = \begin{cases} d_{k,t} & \text{if } (d_{k,t} - d_{k,t-1}) = 0 \text{ and } (d_{k,t+1} - d_{k,t}) = 0 \\ d_{k,t+1} & \text{if } (d_{k,t} - d_{k,t-1}) * (d_{k,t+1} - d_{k,t}) > 0 \\ d_{k,t} & \text{otherwise} \end{cases} \quad (3.19)$$

where x is the desired value when we trained the network with TSCFD cost function.

One of the stopping criteria that we use to stop the training is the MSE error measure, which tries to reach the least mean squared error. MSE is used to adjust the predicted output to be close to the desired value.

We expect better learning of the difference between the continuous desired output and the predicted output.

The step-training method and the TSCFD cost function that are applied to our approach with different configurations will be experimented on three data sets. Then, the configurations that give the best results are selected to test the prediction model, of which the consequences are presented in chapter 5.

3.3 Model Evaluation

General error measures such as MSE, MAE, and MAPE are most widely used to evaluate the forecasting model. The results, the difference between the desired value and the actual output from a neural network, derived from these error measures are meaningless without an evaluation of the system design for the fluctuation direction of the predicted output. If we would like to check the predicted value fluctuation direction that we have achieved, the neural network should be tested and evaluated on the basis of both the accuracy of predicted value and its fluctuation direction evaluation.

A way to measure the performance of a predictive algorithm of the network is needed in two phases of the developmental cycle of the predictive system. First, the response time of the training algorithm during the modeling phase can be evaluated. Secondly, accuracy measurement can be made by evaluating the testing data using various criteria.

In the testing and evaluation phases, the performance of the neural network can be captured by various measures; we only focus on the accuracy measures. The accuracy measures to evaluate and compare the predictive model used in this research are as follows:

- *Mean Squared Error* (MSE) is the average squared predicted error between actual and predicted values for the forecasts made over a series of N forecast patterns. The definition of MSE is

$$MSE = \frac{1}{N} \sum_{k=1}^N (d_k - x_k)^2 \quad (3.20)$$

- *Mean Absolute Error* (MAE) defines the difference between actual and predicted values for the forecasts made as a ratio of the current value.

$$MAE = \frac{1}{N} \sum_{k=1}^N |d_k - x_k| \quad (3.21)$$

- *Mean Absolute Percentage Error (MAPE)* defines the difference between actual and predicted values for the forecasts made as a ratio of the current value in percentage.

$$MAPE = \frac{100}{N} \sum_{k=1}^N \left| \frac{d_k - x_k}{d_k} \right| \quad (3.22)$$

- Percentage of error range, called “*Tolerance*”, defines the difference between actual and predicted values for the forecasts made by pointing them to the specified range.

If we set the Tolerance measurement to 5%, we call it “Tolerance 5%”, which means that the error occurs if the actual value exceeds the absolute value of the desired value plus five percent and minus five percent also. If the desired value is 100, we can set the Tolerance 5% value to 1 when the predicted value is between 95-105. So Tolerance 5% means that any actual values that fall within the 5% range of the desired are considered correct.

$$Tolerance \ n\% = \begin{cases} 1 & \text{if } (d_k - \frac{d_k * n}{100}) \leq x_k \leq (d_k + \frac{d_k * n}{100}) \\ 0 & \text{anyelse} \end{cases} \quad (3.23)$$

Suppose the Tolerance 5% is 80% it means this network is able to predict the future values with an accuracy of 80% with Tolerance of 5% over this testing period.

The percentage of false Tolerance value close to one hundred percent is preferable.

- *Prediction Of Change In Direction (POCID)*[4]: It measures the ratio in percentage of the number of times the actual and predicted values move in the same direction to the total number of predictions.

A POCID error occurs if the desired and the predicted values change in opposite directions (one increases, and the other decreases) as illustrated in Figure 3.7. The POCID of determination can have several possible values:

$$D = \begin{cases} 1 & \text{if } (d_k = d_{k-1}) \\ 1 & \text{if } (d_k - d_{k-1}) \cdot (o_k - o_{k-1}) > 0 \\ 0 & \text{anyelse} \end{cases} \quad (3.24)$$

$$POCID = \frac{100}{N} \sum D \quad (3.25)$$

where d_k and o_k represent the desired and predicted (actual) values respectively. The number of data points forecast is N .

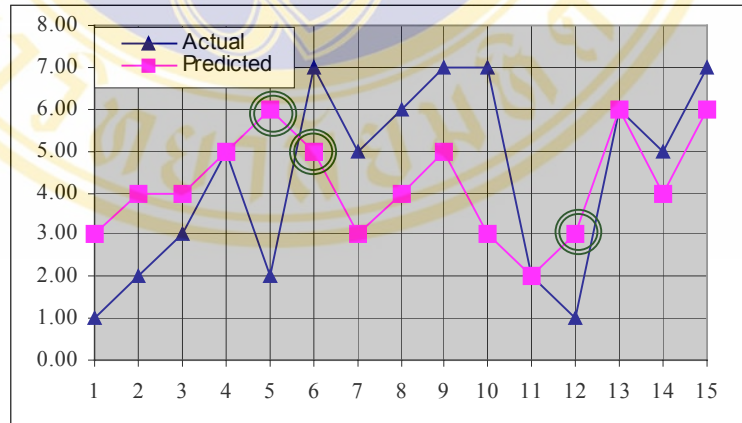


Figure 3.7 The Prediction of Change in Direction

The errors are marked by the cycles. The POCID close to one hundred percent is preferable.

- *Prediction Of Correct Fluctuation Direction (POCFD)*: In this research, we are concerned about the value fluctuation direction as well as the exactly predicted value.

A POCFD error occurs if the different value between the previous desired and the current desired values is positive and the different value between the previous desired and the predicted values is negative as illustrated in Figure 3.8. The formula used to calculate the POCFD is written as

$$D = \begin{cases} 1 & \text{if } (d_k = d_{k-1}) \\ 1 & \text{if } (d_k - d_{k-1})(o_k - d_{k-1}) > 0 \\ 0 & \text{anyelse} \end{cases} \quad (3.26)$$

$$\text{POCFD} = \frac{100}{N} \sum D \quad (3.27)$$

where d_k and o_k represent the desired and predicted (actual) values respectively. The number of data points forecasted is N .

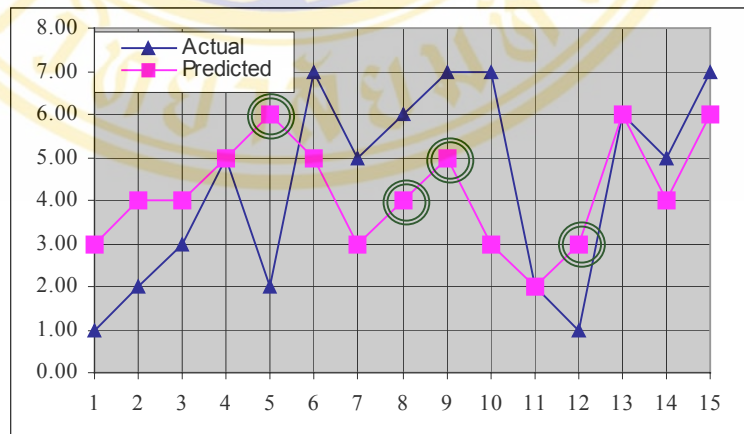
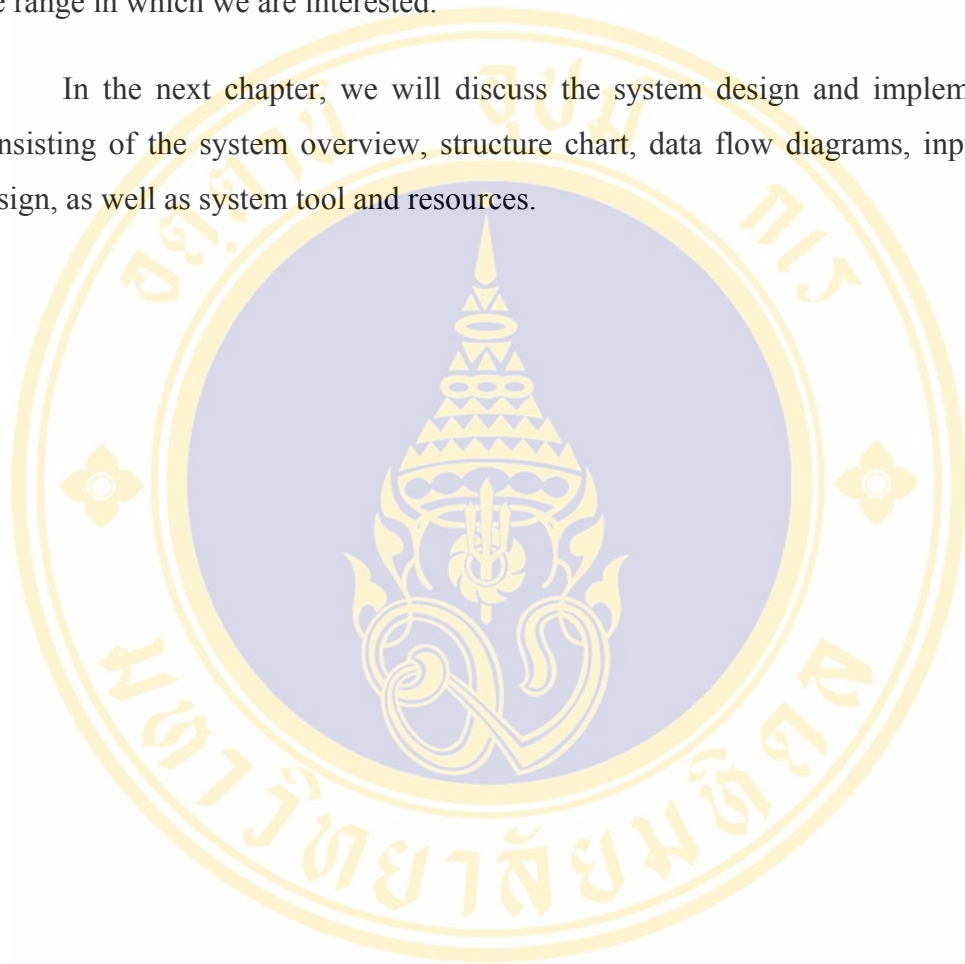


Figure 3.8 The Prediction of Correct Fluctuation Direction

The errors are marked by the cycles. The POCFD close to one hundred percent is preferable.

In our model, we use POCFD to evaluate the predicted fluctuation direction of the neural network output, and use MSE, MAE and MAPE to evaluate the predicted value accuracy of neural network output. In particular, we use Tolerance 1% and Tolerance 5% to evaluate the difference between predicted output and desired value in the range in which we are interested.

In the next chapter, we will discuss the system design and implementation, consisting of the system overview, structure chart, data flow diagrams, input-output design, as well as system tool and resources.



CHAPTER IV

SYSTEM DESIGN AND IMPLEMENTATION

This chapter explores the system design and implementation. Several important aspects of the system design are given. We start with the system overview and structure chart of the neural network for a stock price prediction processing task and the steps to develop the prediction model. In the latter part, the details of all neural network components are described. The remainder of the chapter discusses the data flow diagrams, module specifications, file structure and the input/output design as well as system tool and resources.

4.1 System Overview

Three-layer feedforward neural network is used to simulate the prediction model. A neural network is trained by using a backpropagation algorithm. In our model, we have simulated a neural network for a stock price prediction with an input layer of n_1 neurons, one hidden layer of n_2 neurons and an output layer of one neuron, n_3 . Therefore, it is an n_1 - n_2 - n_3 topology. There are various network parameters that need to be determined. These include the number of moving window sizes (n_1), number of hidden neurons (n_2), learning rate, momentum factor and the network's stopping criteria. All those parameters affect the prediction accuracy results.

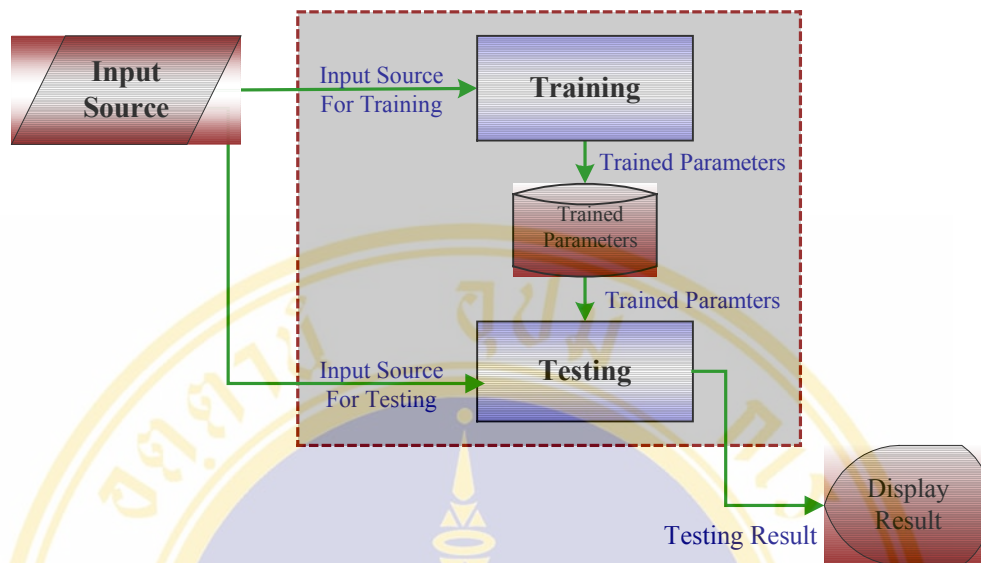


Figure 4.1 A High-Level Diagram of the System

A high-level diagram of the system used to describe the overview of a stock price prediction development is shown in Figure 4.1. We collect the raw data from the Internet data source and store them all in the database. Two main modules of the system are training and testing modules. The results will come up after the process has passed through from the first step to the last one.

4.2 Structure Chart

The structure chart is commonly used to show how system modules are organized. The structure chart shown in this section depicts the processing tasks in our prediction model. The main processing tasks are composed of 3 major processes; data collection, training and testing, as shown in Figure 4.2.

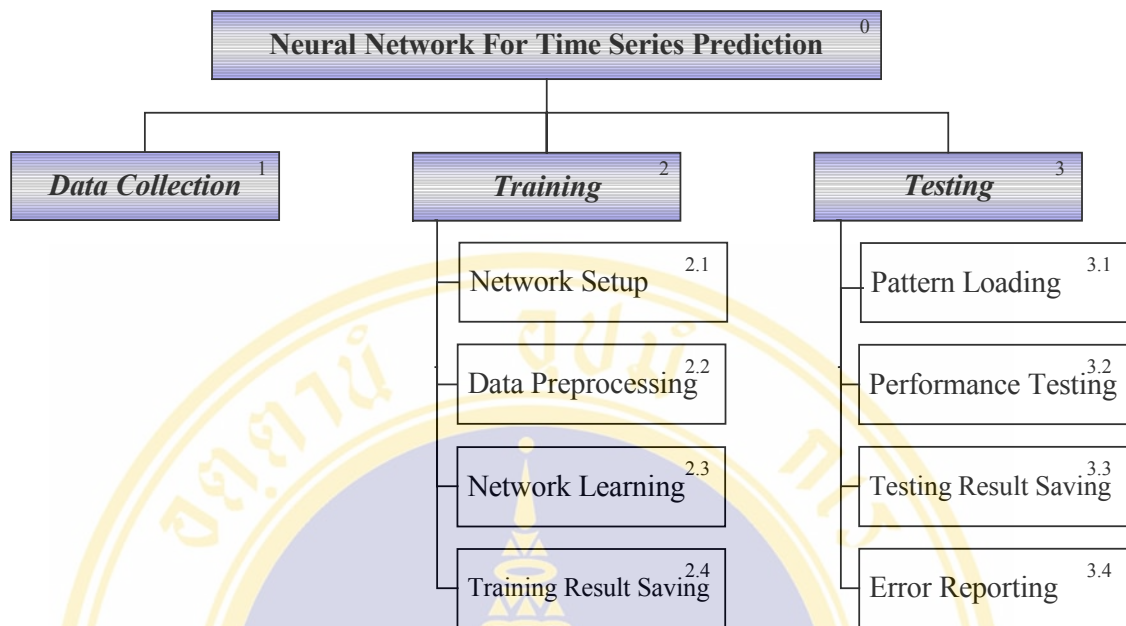


Figure 4.2 Structure Chart of the Neural Network for a Stock Price Prediction

The description of the processing tasks of the system is stated below.

4.2.1 Data Collection

In building a neural network, the most important component is the data availability and how the data is used to train the network. This process includes the handling of missing data points to be fed to the network. Figure 4.3 reveals a process of data collection.



Figure 4.3 Data Collection

In our model, we use two kinds of data. One is the actual stock price data and the other is the Mackey-Glass time series data, which are generated from a given

equation. All of the data sets that are used in this model are simply downloaded from Internet. The highly dynamic nature of the data relating to each of the input vectors, especially financial data, means that the only efficient way to access the up-to-date information is through electronic methods, and the Internet is the most widely available and easily accessible one.

The Yahoo Search Engine has a very good financial service that allows the user to access the up-to-date financial information. This is the service that is used to obtain the data relating to both stocks, Ahold and IBM. We only use the daily data of closing stock price to train the model. For Ahold data series, the received data covers the period from Jan 1995 to Dec 1997. For IBM data series, the received data covers the period from Jan 2000 to Dec 2002.

As required, we arrange all data in time sequence and verify that all variables have exactly the same data format, and that all unique dates have been attached to data derived from all data source. For Ahold and IBM stock price data, the available format derived from Yahoo is appropriate to our model. For the Mackey-Glass time series, we have downloaded them from the Internet. And found that those data series are not valid to feed the network. We also verify that each observation represents a set of Mackey-Glass data by adding uniqueness date to those series. So there are no extra or missing values in the data set.

4.2.2 Training

Training a neural network to learn patterns in the data involves iteratively presenting it with examples of the correct known answers. The objective of training is to find the set of weights between the neurons that determine the global minimum of the cost function. Figure 4.4 illustrates the overview of the training process that is used in our model.

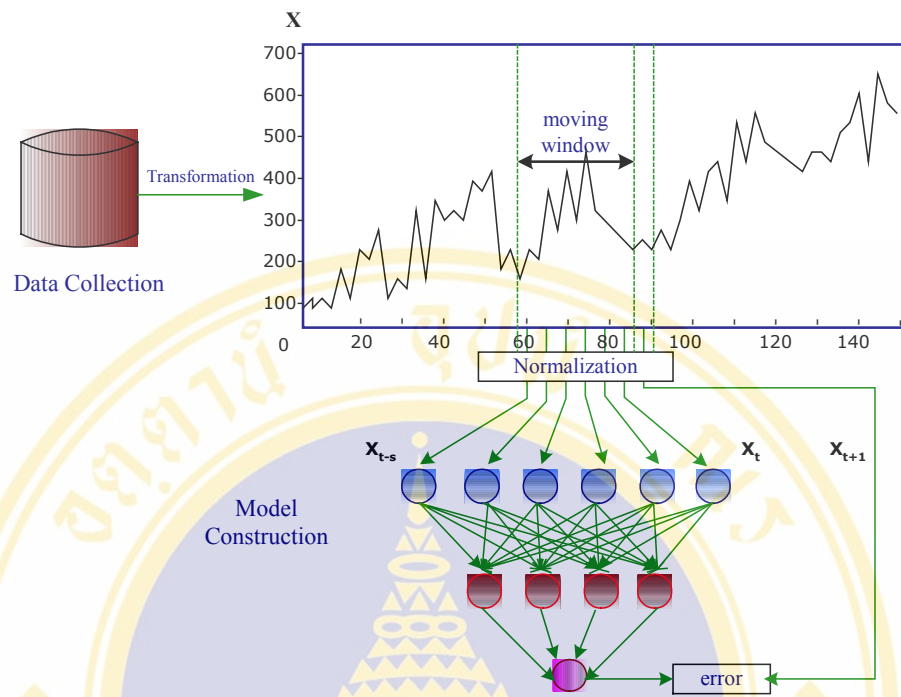


Figure 4.4 Training a Set of Data

A short description of the training process presented in Figure 4.4 is made below.

A number of data points of the data series (the moving window $x_{t-s}, x_{t-s+1}, x_{t-s+2}, \dots, x_t$) are transformed into an appropriate format to be used by the network and then pass through the normalization function for scaling the data as the certain period that corresponds to the selected activation function of the hidden layer. The size s of the moving window corresponds to the number of input units of the neural network. In a forward path, these activation levels are propagated over one hidden layer to one output neuron. The error used for the backpropagation-learning algorithm is computed by comparing the value of the output neuron with the transformed value of the data series at time $t+1$. This error is propagated back to the connections between output and hidden layers and to those between hidden and output layers. After all weights have been updated accordingly, one presentation has been completed. Training a neural network with the backpropagation algorithm usually requires that all representations of the input set (called one epoch) should be presented many times.

The details of all processing tasks of the training process are described as follows.

4.2.2.1 Network Setup

The network setup process provides many categories for setting up a network. These features support the defined value of the network size, training methodology, cost function selection and any parameters. We are concerned about the process in the two color blocks because we have taken these methodologies to improve the prediction accuracy of our work. Figure 4.5 reveals network setup.

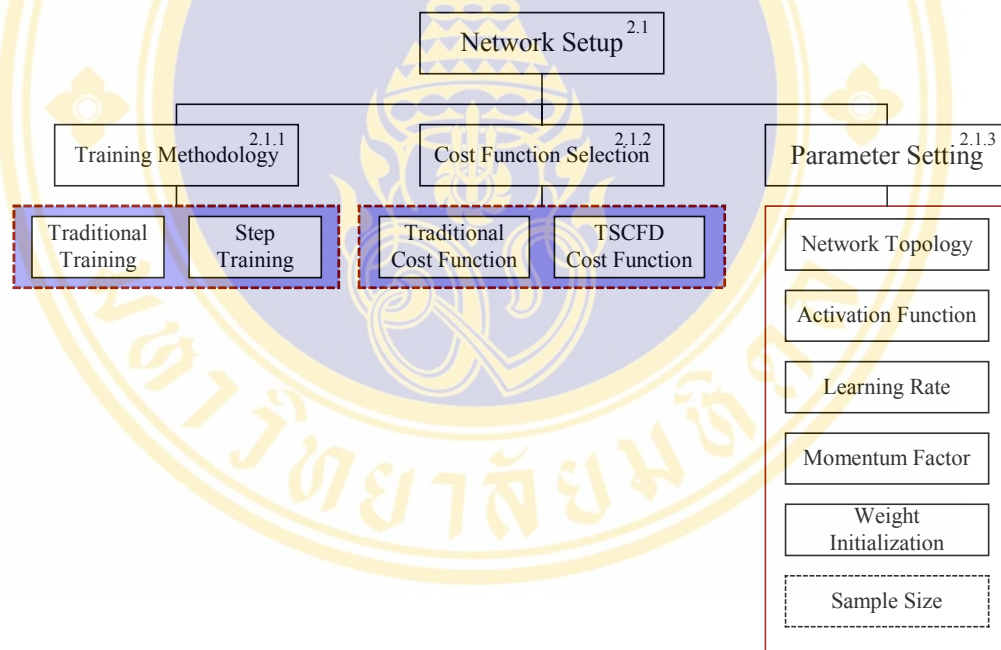


Figure 4.5 Network Setup

4.2.2.1.1 Training Methodology

In this process, we consider two training methods to train the network. One is our approach, the step-training method, and the other is the traditional training method.

- *Step-Training Method*: this module does not require a large set of input data for training (predicting) the network. This method is designed to test only one data point extending from the training data set.
- *Traditional Training Method*: this method requires long input time period to train the network. The sets of data for training and testing have been obviously separated. The testing data set is mostly specified, extending from the training data set for approximately six months or about 200 data points and over.

4.2.2.1.2 Cost Function Selection

This process is to identify and build the different training with the selected cost function. Two kinds of cost functions that we are concerned about the traditional and the TSCFD cost functions, the latter of which belongs to our approach.

4.2.2.1.3 Parameter Setting

The neural network requires several parameters to be fed to the network. These parameters affect the network accuracy. If they cannot be set for the model to train the network well, set of weights are needed to re-initial.

- *Network Topology*: There are an infinite number of ways to construct a neural network. A feedforward with backpropagation network architecture with three layers (input-hidden-output) is used. Our model is designed for only one output neuron corresponding to the value of the next-day forecasting. The number of input neurons and the

hidden neurons are not specified a priori. These network parameters will be selected through the experimentation.

The number of input nodes is a very critical factor in neural network modeling and forecasting, as it corresponds to the number of lagged past observations related to the future values of a data series. Since there is no theory suggesting the appropriate number of input nodes, we resort to experimentation in the network construction process. Five levels of the number of input neurons selecting from 5,6,10,15 and 20 will be used in this study.

However, there is no theory to specify the number of hidden neurons that suits the input neurons for any problem domain. But we explore some researches, which try to optimize this value. There are two formulas to stipulate the number of hidden neurons that is optimized with the number of input neurons. These formulas will be stated in chapter 5.

- *Activation Function* is a function that is used to transform the activation level of a neuron into an output signal. We have fixed both of the activation functions that are used in the hidden layer and output layer. Sigmoid function is employed in the hidden layer and the linear function is utilized in the output layer. The function $f(.)$ is a sigmoid function of net input value.

$$f(net) = \frac{1}{1 + e^{-net}} \quad (4.1)$$

To employ the kind of activation function to the hidden layer affects the range of input value determining the normalized function of the input data.

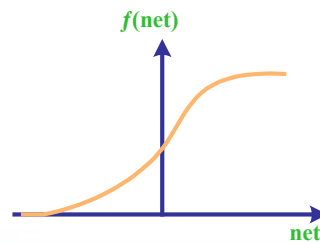


Figure 4.6 Sigmoid Function

Figure 4.6 illustrates the possible value that is given from the sigmoid function. This is the reason why we use the Min-Max normalization function to scale the neuron network inputs.

- *Learning Rate (η)* is a scaling factor that tells the learning algorithm how strong the weights of the connections should be adjusted for a given error. A higher η can be used to speed up the learning process, but if η is too high, the algorithm will “step over” the optimum weights.
- *Momentum Factor:* The momentum parameter (α) is other number that affects the gradient descent of the weights. To prevent each connection from following every little change in the solution space immediately, the momentum term is added that keeps the direction of the previous step, thus avoiding the gradient descent into local minima. The momentum term is constant across presentations.
- *Weight Initialization:* At first, the first weight is taken randomly with the random function where the first weight is ranging between $[0,1]$. Typically, relatively small weights are used to initialize the network. It is advisable to train the network with different sets of weight, since these initial conditions can affect how the network trains and ultimately performs.

- *Sample Sizes*: This parameter needs to be specified when we set the training methodology to the step-training method. This step will be described more in Section 0.

4.2.2.2 Data Preprocessing

Once input data has been selected, to help a neural network produce accurate forecast, the selected raw input data typically must be preprocessed. By reducing the number of inputs to the network, preprocessing helps it learn more easily. Two widely used preprocessing methods are known as transformation and normalization.

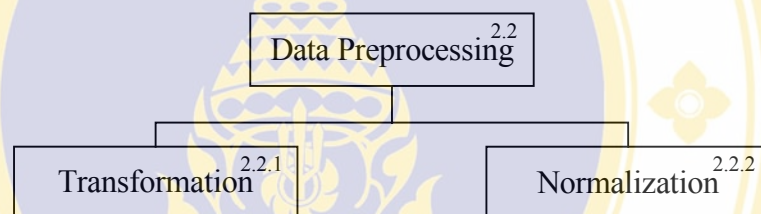


Figure 4.7 Data Preprocessing

Transformation is used to manipulate one or more raw data input to generate a single network input. Normalization is a transformation that distributes data evenly and scales it into an acceptable range for the network usage.

4.2.2.2.1 Transformation

After data patterns have been loaded to the model, the data transformation process is used to transform raw data from the database into an array format of training pattern (input / output pattern). In this model, we use the same set of data to be presented as the network input and output (univariate data series).

4.2.2.2.2 Normalization

To facilitate neural network training, the selected raw input data must be preprocessed. The Min-Max normalization technique is used, and the given results are fed into neural networks as an input. It is a simple linear method of scaling data. At a minimum, data must be scaled into the range used by the network's input neurons. This is typically in the range of zero to one. We normalize the input, x_i , using the following function as follows:

$$x'_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \quad (4.2)$$

where x_{\min} and x_{\max} are the lowest and highest values of the inputs respectively. The maximum and minimum values are gathered from the training data sets.

4.2.2.3 Network Learning

This process can be used after the data preprocessing and network setup processes have already been performed. The necessary information of the network and training patterns has to be completely created before the network training process can begin. There are three processing tasks in the network training process; feedforward learning, error backpropagation and stopping criteria specification.

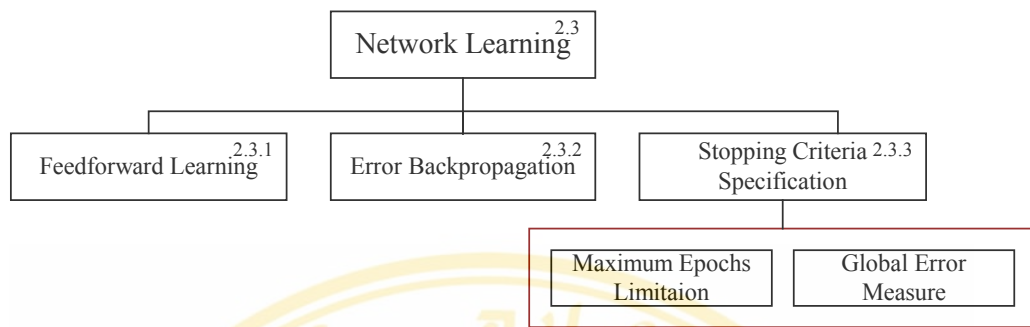


Figure 4.8 Network Learning

We ask to firstly present the stopping criteria specification. In this model, we use two common criteria to stop training a network:

- *Maximum Epochs Limitation*: During iterative training of a neural network, an Epoch is a single pass through the entire training set. Stop training after the error fails to improve by a given amount of training iteration over a given number of epochs. Our model can be experimented with several of the maximum epochs setting as we have provided the place for this parameter setting.
- *Global Error Measure*: We use MSE error measure to stop the training. Training of the network is stopped when the MSE summed over all patterns increases.

We can formulate the backpropagation training steps as shown below.

- 1) Initialize the network by assigning all required parameters.
- 2) Feedforward the input through a network.
 - a) Calculate the hidden level activation by taking the weighted sum of all input neurons for each hidden neuron.
 - b) Obtain the output of each hidden layer by inputting the activation level into the sigmoid function.

- c) Calculate the output neuron activation levels by taking the weighted sum of all hidden neurons for each output neuron.
 - d) Obtain the output neuron by inputting the activation level of the output into the linear function.
- 3) Calculate error of the output neuron.
 - 4) Back-propagate errors and update the connection weights of each inter-connection.
 - 5) Repeat steps 2-3 for each training pattern.
 - 6) Repeat training until the maximum number of epochs is reached or a stopping condition is met.

4.2.2.4 Training Result Saving

When the training process is finished, all sets of weights, important data and parameters of an acceptable network will be saved in the database.

4.2.3 Testing

Once the training is completely performed, it has to be evaluated with some new testing data sets, which have not been seen before. There are four main processes of pattern loading, performance testing, testing result saving and error reporting. Details of each process are described below.

4.2.3.1 Pattern Loading

This is the first process that is used to test the network performance. The first step is to select the trained pattern, which we would like to evaluate. Then we have to specify the testing period if this pattern was trained by using the traditional training method.

4.2.3.2 Performance Testing

During the testing phase, the produced output, x'_i , must be de-normalized (post-processing). Ideally, the normalization should be reversible with little or no loss in accuracy. The outputs of the neural network will be rescaled back to the original value, x_i , using the same formula with the training process as defined below:

$$x_i = (x'_i * (x_{\max} - x_{\min})) + x_{\min} \quad (4.3)$$

where x_{\min} and x_{\max} are the same values that are used in the normalization process.

4.2.3.3 Testing Result Saving

This process is used to save the testing result in the database for any usability, such as being exported to worksheet for plotting the prediction graph etc. There are not many kinds of data that need to be saved. They consist of scaled prediction value, scaled desired value, actual prediction value, actual desired value, and any missing errors that are calculated from many kinds of prediction measurements corresponding to the linked date.

4.2.3.4 Error Reporting

After the testing process is finished, the prediction results are produced. This process shows the detail of the testing results with the missing error that are calculated from many kinds of prediction measurements.

4.3 Data Flow Diagrams

A Data Flow Diagram (DFD) is most commonly used to represent a conceptual model of systems. A DFD does not give detailed descriptions of modules but graphically describes a system's data and how the data interacts within the system.

This section illustrates the data flow diagrams of the neural network for a stock price prediction model. The data flow diagrams in Figure 4.9 through Figure 4.12 show how data is processed by a system in terms of inputs and outputs.

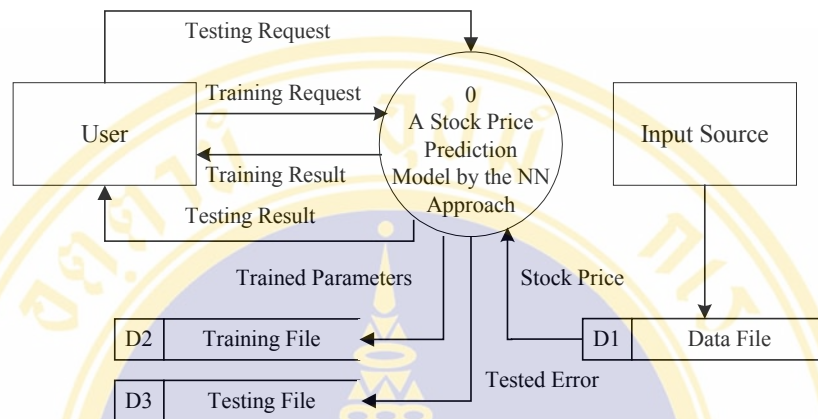


Figure 4.9 Context Diagram: A Stock Price Prediction Model by the Neural Network Approach

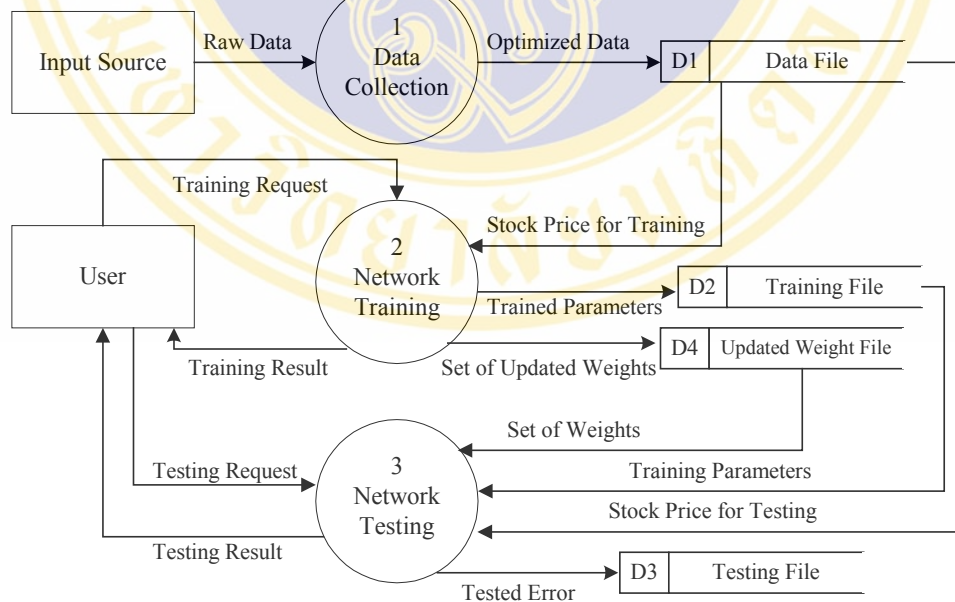


Figure 4.10 Data Flow Diagram Level 1: A Stock Price Prediction Model by the Neural Network Approach

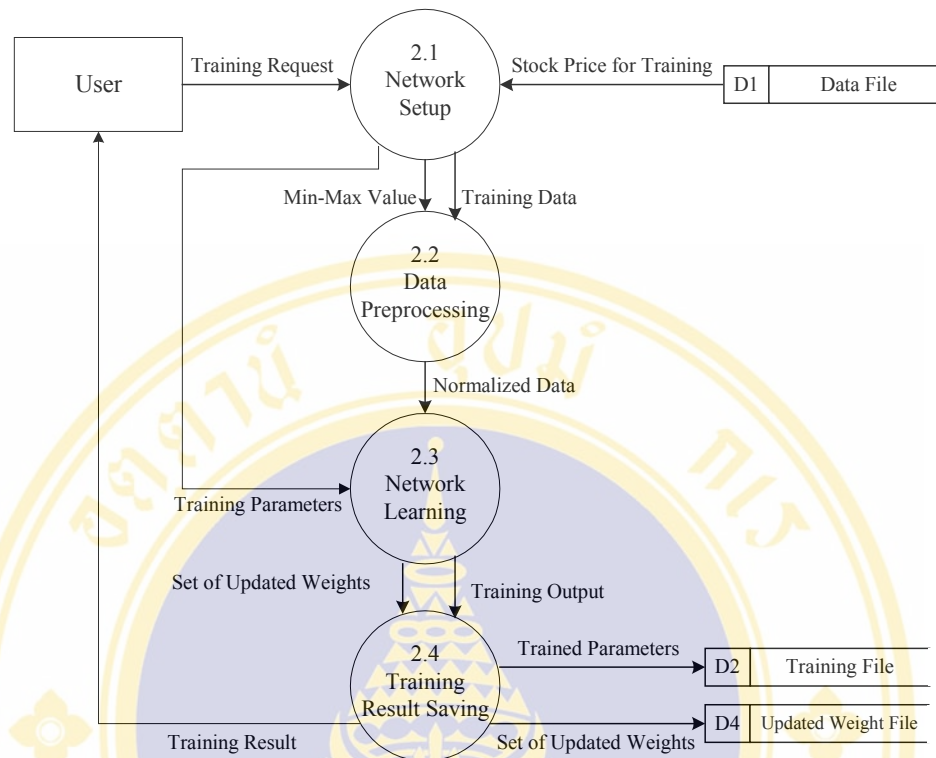


Figure 4.11 Data Flow Diagram Level 2: Network Training

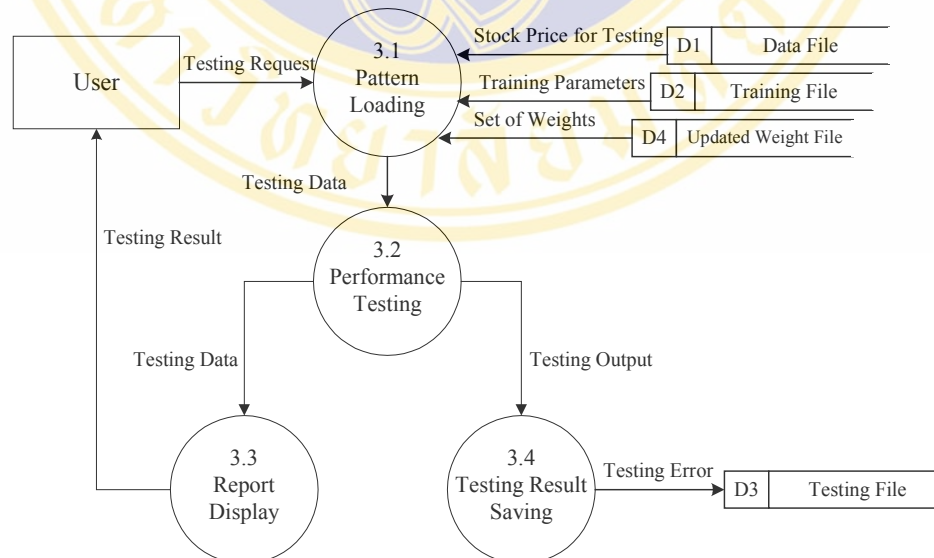


Figure 4.12 Data Flow Diagram Level 2: Network Testing

• Process Description

The following are the processes description of the Data Flow Diagram that is presented above.

Module Name	: 0. Neural Network For A Stock Price Prediction
Description	: Neural Network For A Stock Price Prediction model processes the raw data form the Internet data source and then transform them to the database for training and testing the model accuracy follows the object propose as described in chapter 3.
Input	: Raw data, stock price or Mackey-Glass data series, training parameters and set of updated weights.
Output	: Error of each training epoch, set of updated weights, prediction values with their direction movement, testing error performed as described in chapter 3.
Module Name	: 1. Data Collection
Description	: This process is used to adjust the raw data for suiting the network format and saved them to the database.
Input	: Raw data from Internet data source.
Output	: Data with uniqueness date for all records.
Module Name	: 2. Network Training
Description	: This process is used to train network with feedforward backpropagation learning.
Input	: Training request from user, network parameter and training set of data (a stock price data set or a Mackey-Glass data set).
Output	: Error in every epoch, the final updated weights and total training time consume.
Module Name	: 3. Network Testing
Description	: This process is used to evaluate the network performance after completely trained the network with specific data set.
Input	: Set of data series, set of weights and training parameters
Output	: Prediction values with their direction movement and missing error.

Module Name : 2.1 Network Setup
Description : This process get all training parameters from the input screen and transform them to the program variable.
Input : Training parameters from the input screen.
Output : Program variable of those parameters, Min-Max value.

Module Name : 2.2 Data Preprocessing
Description : This process is used to transform the data series to array format to feeding into the network and then normalized all data series data to the specify range, between [0,1].
Input : Data series (a stock price data or a Mackey-Glass data)
Output : Normalized data as array format.

Module Name : 2.3 Network Learning
Description : This process perform network to learn form the selected data series, training parameters and any specify condition (type of training methodology, type of cost function)
Input : Normalized data and training parameters.
Output : Training result with total training time consumes and set of updated weighs.

Module Name : 2.4 Training Result Saving
Description : This process is used to save all training result, total training time consume and set of updated weights to the database for preparing to feed into the testing process.
Input : Training result with total training time consumes and set of updated weighs.
Output : New record in the database.

Module Name : 3.1 Pattern Loading
Description : This process is used to load the selected training pattern from the database and then display them at the testing screen.
Input : Training pattern, testing period (date_form and date_to).
Output : Display all data on the testing screen.

Module Name : 3.2 Performance Testing
Description : This process is used to evaluated the model by specify the training pattern.
Input : Training parameters, normalized data, testing period and set of weights.
Output : Prediction values with their direction movement, all missing error of five performance measurements.

Module Name : 3.3 Testing Result Saving
Description : This process is used to save all testing result and their error into the database.
Input : Testing result with their error.
Output : New record in the database.

Module Name : 3.4 Result Display
Description : This module displays all testing result with their corresponding training parameter on the testing screen.
Input : Output from the process 3.1: Pattern Loading and 3.2: Perform Testing.
Output : Display all testing result on the testing screen.

4.4 Module Specifications

The module specification represents the programming logic of each program module in English statements. As stated in Section 4.2 , there are two main modules that is described, training and testing modules. The specification of each module is described below.

Main Module	Sub Module
Training Module	Network Setup Data Preprocessing Initial Weight Feedforward Training Check Stopping Condition Error Backpropagation Training Training Result Saving
Testing Module	Pattern Loading Performance Testing Data Post-processing Testing Result Saving Error Reporting

Module Name : Training
Description : Perform the network for a stock prediction
Input : Training pattern, network parameters, type of training methodology, type of cost function
Output : Generalize set of weights, TIME_COMSUME
Logic Summary :

1. Select training pattern (data source)
2. Preprocess the data series of the selected pattern
3. Get the network parameters (number of input neurons, number of hidden neurons, training period, maximum epochs, learning rate, momentum factor, type of training methodology and type of cost function)
4. Set Loop = 1
5. Process feedforward training
6. Sum error of all pattern, desired value – actual output, by applying the MSE error measure
7. Check the training error after perform feedforward training on each pattern
 - 7.1 IF the current error > previous error, then
 - Save training results (weights and training parameter)
 - 7.2 IF the current error ≤ previous error, then
 - Check the loop value and the maximum epoch
 1. IF Loop < Max Epochs, then
 - Process error backpropagation procedure
 - Set Loop = Loop +1
 2. IF Loop = Max Epochs, then
 - Save training results (weights and training parameter)

Module Name : Network Setup
Description : This module is used to get required parameters from the training screen and transform to the program variable for feeding to the network.
Input : #INPUTNODE, #HIDDENNODE, LEARNING_RATE, MOMENTUM, MAX_EPOCH, Training Period (DATEFROM, DATETO), Type of Training Methodology (IS_STEP_TRAININ), Type of Cost Function (IS_TSCDM)
Output : Set of weights, TIME_COMSUME
Logic Summary :

1. Get training patterns by retrieving data from the database.
2. Count the record of all selected data (ALLDATA)
3. Set #OUTPUTNODE = 1
4. Get parameters from the input screen.
5. Count numbers of all training pattern by

$$\#PATTERN = ALLDATA - \#INPUTNODE$$

Module Name : Data Preprocessing
Description : This module is used to preprocess the raw data, transformation and normalization.
Input : Raw data (training patterns)
Output : Normalized data that have the scale between [0,1]
Logic Summary :

1. Get training patterns by retrieving data from the database.
2. Transform raw data in array format, PRICE(i)
3. Find MIN, MAX value.
4. Normalize raw data by

$$O_i(i) = (PRICE(i) - MIN) / (MAX - MIN)$$
5. Perform the desired value in array format, Tk
6. Perform the desired direction value in array format, TDM(k)

Module Name : Initial Weight
Description : This module provides a function to generate initial random weight.
Input : #INPUTNODE, #HIDDENNODE, #OUTPUTNODE
Output : Initial set of weights
Logic Summary :

Initial random weights by calling the standard function of Microsoft Visual Basic 6.0

1. Initial random weights between input neurons and hidden neurons, $W_{ij}(i,j)$
2. Initial random weights between hidden neurons and output neurons, $W_{jk}(j)$

Module Name : Feedforward Training
Description : This module performs feedforward training.
Input : $O_i(i)$, set of weights
Output : O_k , Error of all patterns
Logic Summary :

1. If all $O_i(i)$ have not been process to the network then
 - a. Calculate input value of each hidden neuron: $I_j(j)$
 - b. Calculate output value of each hidden neuron: $O_j(j)$
 - c. Calculate input value of output neuron: I_k
 - d. Set input value = output value of output neuron.
 $O_k = I_k$
 End if
2. Calculate missing error of all patterns.

$$SUMERROR = SUMERROR + (T_k - O_k)^2$$

Module Name : Check Stopping Condition
Description : This module performs a function to stop the network training.
Input : SUMERROR, MAX_EPOCH
Output : Signal to stop or continue network training.
Logic Summary :

Check SUMERROR between current epoch and previous epoch.

If CURRENT_SUMERROR < PREVIOUS_SUMERROR then

- Continue training; go to “Error Backpropagation Training” module.
- Set LOOP = LOOP + 1

Else

If LOOP <= MAX_EPOCH Then

- Continue training; go to “Error Backpropagation Training” module.
- Set LOOP = LOOP + 1

Else

- Go to module “Save Training Result”

End If

End If

Module Name : Error Backpropagation Training
Description : This module performs backward training function.
Input : Oi(i), Oj(j), Ok, Tk
Output : Set of weights after perform adjusting
Logic Summary :

1. If all Ok have not been backward process to the network then
 - a. Calculate error value of output neuron: ERROR_K
 If type of training methodology, IS_TSCDM, is False then
 $ERROR_K = Tk - Ok$

Else

Select new desired value, Xk, by concentrating with the two steps continuous direction movement as described in Chapter 3.

$$ERROR_K = Xk - Ok$$

End If

- b. Calculate error value of hidden neuron:

$$ERROR_J, DELTA_Wjk, DELTA_Wij(I,j)$$

If all Oj(j) have not been backward process to the network then

- $ERROR_J = Oj(k, j) * (1 - Oj(k, j)) * (Wjk(j) * ERROR_K)$
- Calculate DELTA_Wjk(j)

If all Oi(i) have not been backward process to the network then

- Calculate DELTA_Wij(i,j)

End If

End If

End if

2. Calculate new connection weights between hidden and output neurons.
3. Calculate new connection weights between input and hidden neurons.

Module Name : Training Result Saving
Description : This module is used to save all training parameters and set of weights.
Input : All parameters from the input screen, set of weights, total time consume.
Output : Message popup on screen.
Logic Summary :

1. Save all parameters from the input screen, set of weights.
2. Calculate total training time consume and save this value to the database.
3. Display message on screen, “Successfully train”

Module Name : Testing
Description : This module is used to test the performance of the prediction model that trained by using each of training set.
Input : Trained patterns, set of weights, network parameters that send from the training’s procedure
Output : Prediction value (exact value), missing error
Logic Summary :

1. Load testing pattern (trained pattern) and any parameters the saved from training procedure (number of input neurons, number of hidden neurons, type of training methodology, set of weights)
2. Specify the testing period
 - a. IF training methodology \neq “step training”, then
 - Specify testing period
 - b. IF training methodology \neq “step training”, then
 - Testing period has the same value of the training period.
3. Predict the future value of the testing data set.
4. De-normalize the predicted value to actual scale.
5. Evaluate the testing set by calculating the missing error.
6. Save testing result. In this function, the calculated values are saved to the database, which thus contains the actual prediction and the missing error value of the testing set.
7. Display result

Module Name : Patterning Loading
Description : This module will retrieve parameters and set of weights that according to the selected trained pattern to perform the testing process.
Input : Trained patterns and new unseen data for evaluate the prediction performance.
Output : Trained pattern, #INPUTNODE, #HIDDENNODE, Testing Period (DATEFROM, DATETO), Type of Training Methodology (IS_STEP_TRAININ), Set of weights
Logic Summary :
 1. Get trained patterns by retrieving data from the database.
 2. Load trained information on the testing screen.
 3. Get parameters from the input screen and transform them to program variable.

Module Name : Performance Testing
Description : This module will retrieve parameters and set of weights that according to the selected trained pattern to perform the testing process.
Input : Trained patterns and new unseen data for evaluate the prediction performance, network parameters
Output : Prediction value (scaling data)
Logic Summary :
 1. Go to module “Data Preprocessing”, acts like a training process.
 2. Go to module “Feedforward Training”, acts like a training process.

Module Name : Data Post-processing
Description : This module will de-normalize the prediction value to the actual scale.
Input : Prediction value (scaling data), MIN and MAX value
Output : Prediction value (actual scale)
Logic Summary :
 1. Retrieve MIN, MAX of the selected trained pattern form the database.
 2. De-normalize all prediction value (scaling data) to the actual scale by

$$RE_SCALE_Ok = Ok * (MAX-MIN) + MIN$$

Module Name	: Testing Result Saving
Description	: This module is used to save all testing result into the database for using to generate the graphical report or diagram.
Input	: Prediction value (scaling and actual data), all calculated error (MSE, MAE, MAPE, POCDM, Tolerance 1% and Tolerance 5%)
Output	: -
Logic Summary	: <ol style="list-style-type: none"> 1. Calculate all missing errors. The calculation formulas have been given in Chapter 2. 2. Save all testing data into the database.

Module Name	: Error Reporting
Description	: This module displays all testing data that processed in the testing process.
Input	: Prediction value (scaling and actual data), all calculated error (MSE, MAE, MAPE, POCDM, Tolerance 1% and Tolerance 5%), direction movement of the desired values and prediction values.
Output	: All testing data as same as the input value.
Logic Summary	: Display all testing data in the testing screen after completely process.

4.5 File Structure

This part indicates the detail of each file (database table) in the research, which is divided into eight tables as follows.

- **Master Table**
 - 1) tblPattern
 - 2) tblTrainingInput
- **Transaction Table**
 - 1) tblTraining
 - 2) tblTesting
 - 3) tblWeight_Input_Hidden
 - 4) tblWeight_Hidden_Output
- **Temporary Table**
 - 1) tmpTraining_Result
 - 2) tmpTesting_Result

Table Name: tblPattern**Description:** Collect the information of the data source.

No.	Field Name	Description	Type	Size
1	<u>PatternId</u>	Primary key	Integer	-
2	PatternName	Name of Pattern	String	50
3	Active	Use for activate of de-activate the pattern	Yes/No	-

Table Name: tblTrainingInput**Description:** Collect all of training input.

No.	Field Name	Description	Type	Size
1	<u>PatternId</u>	Primary key	Integer	-
2	<u>Date</u>	Primary key	Date	-
3	InputValue	Raw data	Double	-

Table Name: tblTraining**Description:** Collect all of training results of each training.

No.	Field Name	Description	Type	Size
1	<u>TrainingID</u>	Primary key (auto number)	Integer	-
2	<u>PatternId</u>	PatternId	Integer	-
3	<u>DateFrom</u>	Starting training period	Date	-
4	<u>DateTo</u>	Ending training period	Date	-
5	InputNode	Number of input neurons	Integer	-
6	HiddenNode	Number of hidden neurons	Integer	-
7	SampleSize	Sample size of each training patterns, require if the IsStepTraining field is set to "Yes"	Integer	-
8	LearningRate	Learning rate	Integer	-
9	Momentum	Momentum factor	Integer	-
10	MaxLoop	Identify the limitation of maximum epochs	Integer	-
11	IsStepTraining	Type of training methodology	Yes/No	-
12	ErrorType	Type of cost function (1: Traditional, 2: TSCDM)	Integer	-
13	Min	Minimum value of this data source	Integer	-
14	Max	Maximum value of this data source	Integer	-
15	MSE	Training MSE	Double	-
16	TimerCounter	Training time (stored in format: hh:mm:ss)	Text	-

Table Name: tblTesting**Description:** Collect all of testing results of each testing.

No.	Field Name	Description	Type	Size
1	<u>TestingId</u>	Primary key (auto number)	Integer	-
2	PatternId	PatternId	Integer	-
3	TriningId	TrainingId	Integer	-
4	DateFrom	Starting testing period	Date	-
5	DateTo	Ending testing period	Date	-
6	Patterns	Number of all testing patterns	Integer	-
7	MSE(Scale)	MSE value, original value that produced from the testing procedure	Double	-
8	MAE(Scale)	MAE value, original value that produced from the testing procedure	Double	-
9	MAPE(Scale)	MAPE value, original value that produced from the testing procedure	Double	-
10	MSE(Actual)	MSE value, after applied de-normalize function to the prediction value	Double	-
11	MAE(Actual)	MAE value, after applied de-normalize function to the prediction value	Double	-
12	MAPE(Actual)	MAPE value, after applied de-normalize function to the prediction value	Double	-
13	POCID(%)	Percentages of POCID error	Double	-
14	Correct	Number of the correctness fluctuation direction	Integer	-
15	Incorrect	Number of the incorrect fluctuation direction	Integer	-
16	Uncertain	Number of the uncertain fluctuation direction	Integer	-
17	1%Accuracy	Percentage of <i>Tolerance</i> 1% error	Double	-
18	5%Accuracy	Percentage of <i>Tolerance</i> 5% error	Double	-

Table Name: tblWeight_Input_Hidden**Description:** Store the connection weight value between the input and hidden layer.

No.	Field Name	Description	Type	Size
1	<u>TestingId</u>	Primary key	Integer	-
2	InputNodeId	Primary key	Integer	-
3	HiddenNodeId	Primary key	Integer	-
4	IHWeight	Connection weight value	Double	-

Table Name: tblWeight_Hidden_Output**Description:** Store the connection weight value between the hidden and output layer.

No.	Field Name	Description	Type	Size
1	<u>TestingId</u>	Primary key	Integer	-
2	HiddenNodeId	Primary key	Integer	-
3	OutputNodeId	Primary key (fixed to 1)	Integer	-
4	HOWeight	Connection weight value	Double	-

Table Name: tblTraining_Result**Description:** Store the temporary data that produced from the training procedure, the values of the latest training epoch.

No.	Field Name	Description	Type	Size
1	<u>PatternId</u>	Primary key	Integer	-
2	<u>Date</u>	Primary key	Integer	-
3	Actual	Desired value, original scale before perform the normalize function	Double	-
4	Predict	Prediction value, after applied de-normalize function to the <i>Ok</i> value	Double	-
5	Dk	Desired value, applied normalize function to the actual value, <i>Actual</i> field, for feeding into the network for training	Double	-
6	Ok	Prediction value, original value that produce from the training procedure	Double	-

Table Name: tblTesting_Result**Description:** Store the temporary data that produced from the testing procedure, the values of all testing patters.

No.	Field Name	Description	Type	Size
1	<u>PatternId</u>	Primary key	Integer	-
2	<u>Date</u>	Primary key	Integer	-
3	Actual	Desired value, original scale before perform the normalize function	Double	-
4	Predict	Prediction value, after applied de-normalize function to the <i>Ok</i> value	Double	-
5	Dk	Desired value, applied normalize function to the actual value, <i>Actual</i> field, for feeding into the network for training	Double	-
6	Ok	Prediction value, original value that produced from the training procedure	Double	-

4.6 Input-Output Design

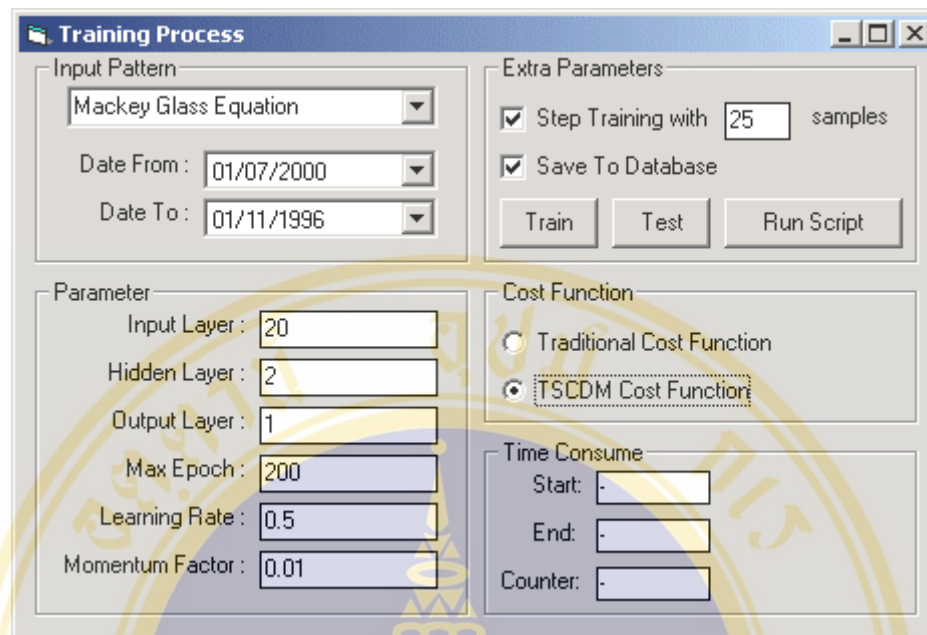
In this section, screens created from the research are described. They are divided into two main screen types as follows:

4.6.1 Training Screen Design

- A screen, Figure 4.13, enables the user to control how the network is trained by specifying the network topology.
- Figure 4.14 shows where to select the training pattern to feed the network.
- Figure 4.15 shows the calendar that is used to specify the starting date and the ending date of the training period.
- Figure 4.16 shows the time consumption report screen that includes the starting time and the ending time when a network is completely trained.

4.6.2 Testing Screen Design

- A screen, Figure 4.17, enables the user to test the trained patterns. All errors of all data points are presented here.
- Figure 4.18 shows the example of the prediction values with an error after performing the test of patterns in the testing process.



Training Process

Input Pattern: Mackey Glass Equation

Date From: 01/07/2000

Date To: 01/11/1996

Extra Parameters:

- ☒ Step Training with 25 samples
- ☒ Save To Database

Train Test Run Script

Parameter:

Input Layer: 20

Hidden Layer: 2

Output Layer: 1

Max Epoch: 200

Learning Rate: 0.5

Momentum Factor: 0.01

Cost Function:

- ☐ Traditional Cost Function
- ☒ TSCDM Cost Function

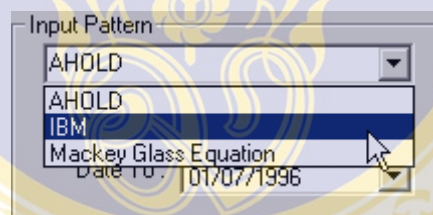
Time Consume:

Start: -

End: -

Counter: -

Figure 4.13 Training Screen

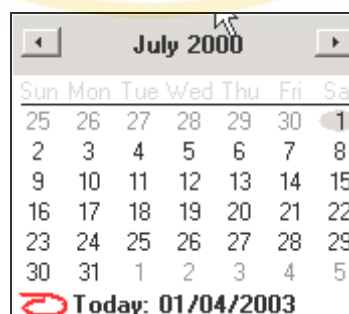


Input Pattern:

- AHOLD
- AHOLD
- IBM
- Mackey Glass Equation

Date To: 01/07/1996

Figure 4.14 Selecting a Pattern Screen



July 2000

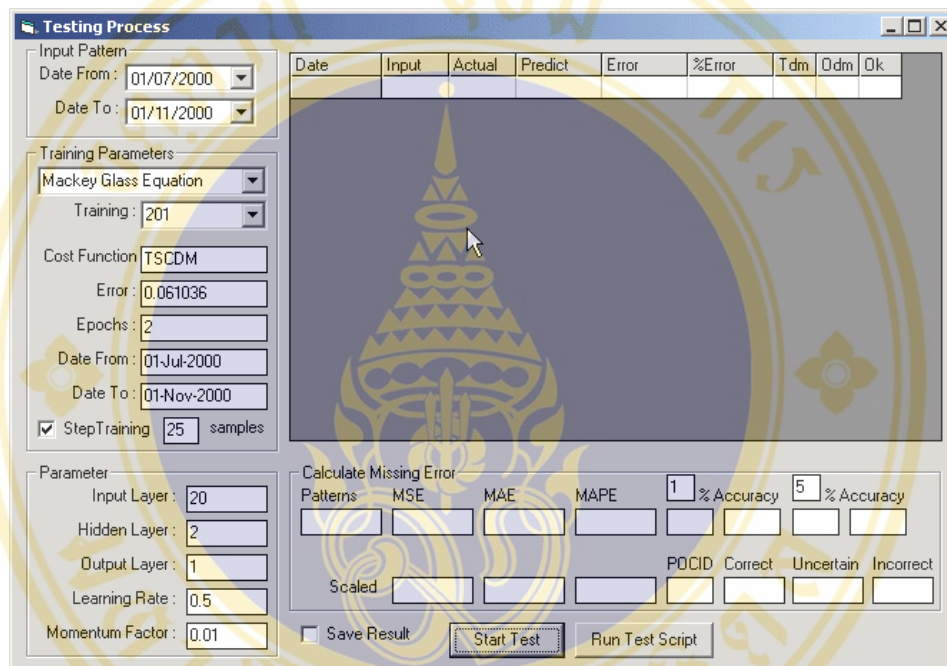
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today: 01/04/2003

Figure 4.15 Calendar of Training Screen

Time Consume	
Start:	18:21:39
End:	18:43:32
Counter:	00:21:53

Figure 4.16 Time Consume Report of Training Screen



Testing Process

Input Pattern
 Date From: 01/07/2000
 Date To: 01/11/2000

Training Parameters
 Mackey Glass Equation
 Training: 201
 Cost Function: TSCDM
 Error: 0.061036
 Epochs: 2
 Date From: 01-Jul-2000
 Date To: 01-Nov-2000
☒ StepTraining 25 samples

Parameter
 Input Layer: 20
 Hidden Layer: 2
 Output Layer: 1
 Learning Rate: 0.5
 Momentum Factor: 0.01

Date	Input	Actual	Predict	Error	%Error	Tdm	Odm	Ok
(Large watermark of Mahidol University logo)								

Calculate Missing Error
 Patterns: MSE MAE MAPE 1 % Accuracy 5 % Accuracy
 Scaled
☐ Save Result **Start Test** Run Test Script

Figure 4.17 Testing Screen

Testing Process

Input Pattern
 Date From: 01/07/2000
 Date To: 01/11/2000

Training Parameters
 Mackey Glass Equation
 Training: 201
 Cost Function: TSCEM
 Error: 0.061036
 Epochs: 2
 Date From: 01-Jul-2000
 Date To: 01-Nov-2000
☒ Step Training 25 samples

Date	Input	Actual	Predict	Error	%Error	Tdm	Odm	Ok
12/07/2000	0.5							
13/07/2000	0.81							
14/07/2000	1.04							
15/07/2000	1.04							
16/07/2000	1.16							
17/07/2000	1.09							
18/07/2000	0.97							
19/07/2000	0.8							
20/07/2000	0.67							
21/07/2000	0.82							
22/07/2000	1.07							
23/07/2000	1.19							
24/07/2000	1.29							
25/07/2000	1.16							
26/07/2000	0.83	0.83	0.8362213	-0.006221	0.749554	0	0	1
27/07/2000	0.56	0.56	0.5599884	0.000012	0.002076	0	0	1

Parameter
 Input Layer: 20
 Hidden Layer: 2
 Output Layer: 1
 Learning Rate: 0.5
 Momentum Factor: 0.01

Calculate Missing Error
 Patterns: 99
 MSE: 0.010813
 MAE: 0.048466
 MAPE: 5.516069
 % Accuracy: 67%
 Scaled: 0.006688
 POClD: 84%
 Correct: 80
 Uncertain: 2
 Incorrect: 16

☐ Save Result **Start Test** Run Test Script

Figure 4.18 Testing Screen after Performing the Prediction

4.7 System Tools and Resources

We develop our model based on the following specifications.

Operating System	: Windows 2000
RAM	: 256 MB
Compiler	: Microsoft Visual Basic 6.0
Database	: Microsoft Access 2000
CPU Speed	: Pentium IV 1.7 GHZ

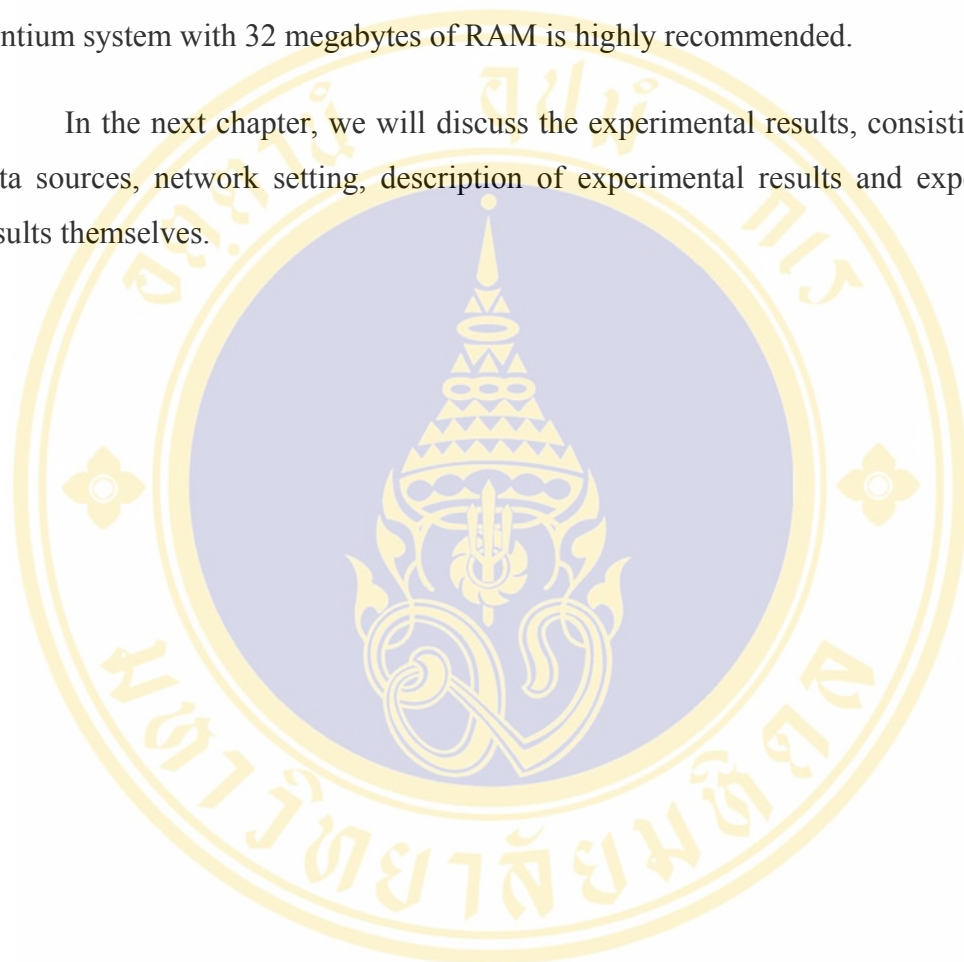
We develop a model using Microsoft Visual Basic 6.0 and use Microsoft Access 2000 to manage and store the data. The neural network for a stock price prediction model is separated into two main modules of training and testing.

The application runs in graphical mode and is able to retrieve the training and testing results from the database directly so that user can view the summary of results or export those results to Microsoft Excel for graph plotting. The screens as shown in Section 4.6 allow an interactive communication between users and the system, as well

as allow users to set the parameter as they want. This system works properly on Microsoft Windows 2000 Professional operating system with Thai Regional setting.

The neural network application can run even on relatively small or old systems; however, due to the computationally intensive nature of many procedures, a Pentium system with 32 megabytes of RAM is highly recommended.

In the next chapter, we will discuss the experimental results, consisting of the data sources, network setting, description of experimental results and experimental results themselves.



CHAPTER V

EXPERIMENTAL RESULTS

In this chapter, we discuss the experimental results starting with the data sources, from which we derive data used in our experiments, next, network setting, including parameters and network topology. The remainder of the chapter discusses some descriptions consisting of a variety of charts and tables summarizing the consequences of the experiments.

We have setup the experiment to evaluate the ability of neural network models in order to develop a set of minimized weights that are trained by our proposed approach. There are two kinds of data sets that are used to evaluate the model. The first two sets are the daily closing stock prices that have been acquired from the Yahoo Financial Data Center. The closing stock prices of the Ahold and the IBM are used in this experiment. These stocks are traded in the New York Stock Exchange (NYSE). The second is the chaotic data set that has been generated from Mackey and Glass differential equation.

According to the designed experiment, the important information for the reliability of results is described in details as follows.

5.1 Data Sources

The goal of the following evaluation is to determine the performance of the neural network that is trained with the step-training method and the TSCFD cost function, which gives higher accuracy (concerning both the value predicted accuracy and value fluctuation direction correctness) than the training method performed by the traditional neural network based on the traditional cost function.

The evaluation is intended to reveal characteristics of any kind of data series that unfavorably affect the forecasting accuracy. The three data sources, two real

world data series and generated data series are selected to provide a reasonable coverage of the possible characteristics likely to be encountered in experiment by evaluating the neural network performance of the training and learning methodology based on our proposed method.

The reason for using the same set of data, univariate time series, as the input and the desired output of a network is to enforce the neural network that has applied our proposed approach to produce the results without being affected by any external factor besides the working of the network engines. The assumption is that the network may yield prominent results. The other reasons that given from many researchers, which we used to support our assumption, are stated below.

- The univariate time series have a few outlying values (make it good theoretical candidate) while the other will not [23].
- The neural network model will find out the relationship between inputs and the fixed targets. The relationship is discovered from the data rather than according to the human expectation. The several trainings are conducted using different variables as inputs to a neural network and the performance of them are compared. If there is not much difference on the performance with or without a variable, this variable is said to be of less significance to the target and thus can be delete from the inputs to the network [12].

The data series are important components for testing the model. The property of the data source, which is used, is given below.

- The *Mackey-Glass* time series is chosen because it is infinite dimensionally, which requires an infinite set of independent real numbers to specify an initial condition, this data set has high non-linearity property [27], and widely studied by many researches [23].
- The *stock price* data series is chosen because it is a popular series used in the forecasting literature, especially in the neural network model for the time series prediction, and because it contains a real world difficulty to

forecast the future value. We decide to use only the daily closing stock price to feed the neural network model as the input value because we intend to use the neural network like a non-parametric model. The aim of a non-parametric model constructing is to make the forecasting with more freedom and give the model an ability to predict dependent variables by deciding the relative importance of the independent variables.

The details of the data sets are introduced below.

1) *Mackey-Glass Time Series*

For stock price prediction, we use the Mackey-Glass equation to generate the data set to evaluate the model. This data set is the example proposed by Mackey and Glass, which describes the production of white blood cells as the following.

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t) \quad (5.1)$$

where $x(t)$ denotes the concentration of blood at time t , and τ is a delay time (for leukemia patient, τ is excessively long).

We have downloaded the Mackey-Glass data set, which is used in the model, from <http://www.ece.ogi.edu/~ericwan/DATA/mg17.dat> and then unique date values are added to reproduce all data to be compatible with the model input format.

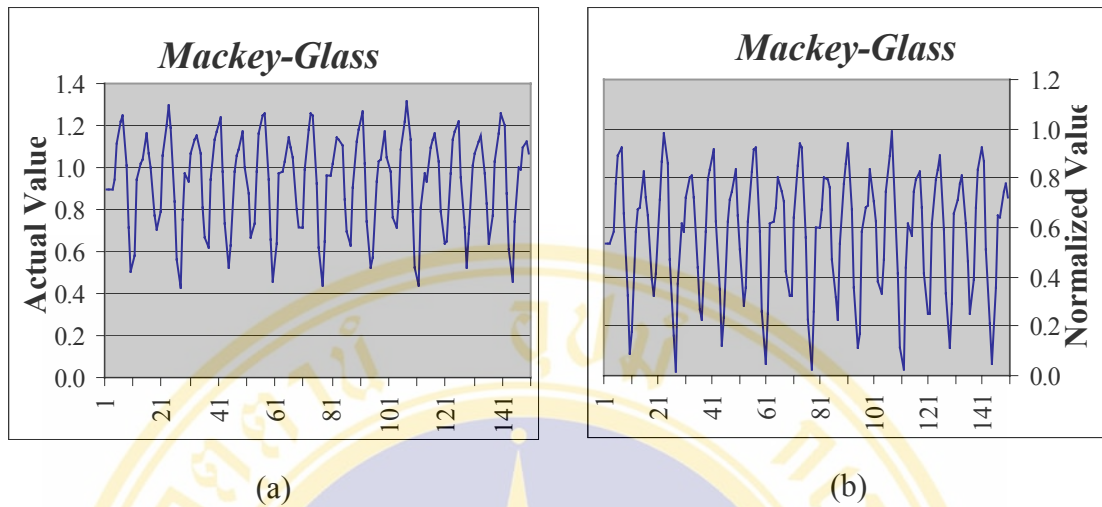


Figure 5.1 Generated Data of Mackey-Glass Time Series
 (a) The Actual Value. (b) The Value after Normalized Return.

The generated data from Jan 1st, 2000 to May 1st, 2001 are used. There are 487 observations with the lowest value of 0.4176 and the highest value of 1.3167. Figure 5.1 (a) depicts the actual value of some generated data set and (b) shows the value after normalized return.

2) *Stock Price Data Set*

In our experiment, the daily closing stock prices of Ahold and IBM are used as time series data to feed the neural network model.

For Ahold, the closing prices from Jan 1st, 1994 to Jan 1st, 1997 are used. This period consists of approximately 504 trading days, with the lowest price shown at 29.63 and the highest price shown at 64. Figure 5.2 (a) shows the daily Ahold stock prices and (b) shows the values after normalized return based on the calculation defined in Section 4.2.2

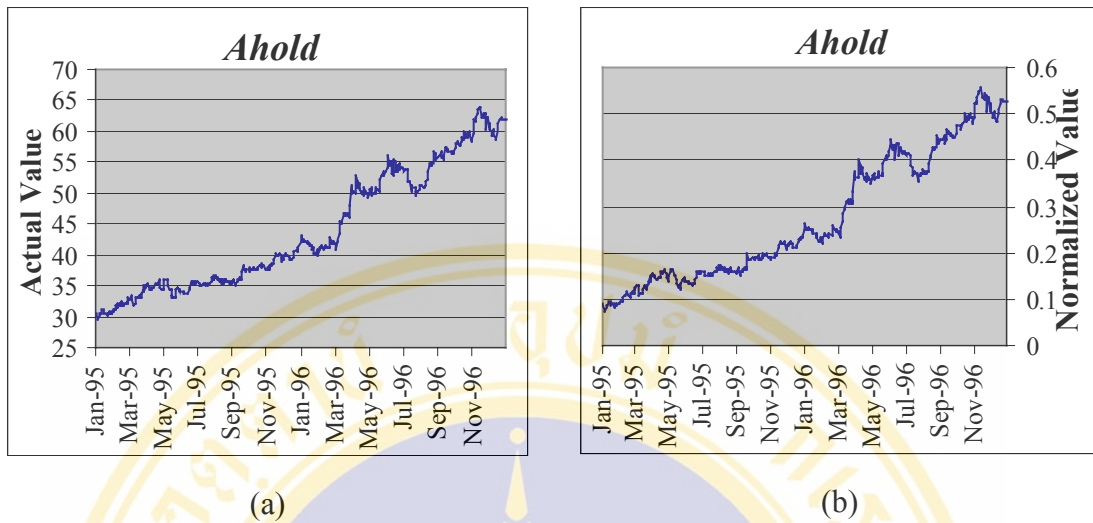


Figure 5.2 Daily Price of Ahold

(a) The Actual Value. (b) The Value after Normalized Return.

For IBM, the closing prices from Jan 1st, 2001 to Jan 1st, 2003 are used. This period consists of 500 trading days, with the lowest price shown at 54.86 and the highest price shown at 124.85. Figure 5.1 (a) shows the daily closing prices of IBM stock market and (b) shows the values after normalized return.

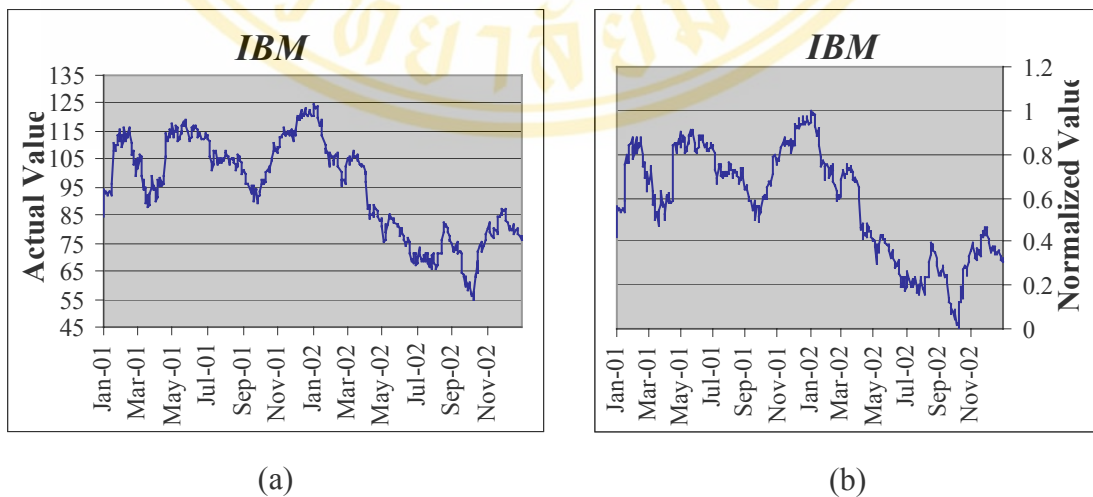


Figure 5.3 Daily Price of IBM

(a) The Actual Value. (b) The Value after Normalized Return.

The output objective is to forecast the future daily closing prices of the trained stock.

5.2 Network Setting

In order to proceed a certain test and evaluate our idea, there are many parameters to be set and network topology decisions to be made, which affect the results of a network. In our experiments, we consider a three-layer feedforward backpropagation architecture for time series modeling. The activation function is chosen as the sigmoid function in the hidden neurons, and an output neuron is set as the linear function.

A number of input neurons are chosen to perform the experiment, as the input vectors are 5, 6, 10, 15 and 20 neurons.

A numbers of hidden neurons of each topology are chosen determined by the previewed researches [16]. There are two formulas as stated below.

$$No. \text{ hidden neurons} = \sqrt{input \cdot output}$$

and

$$No. \text{ hidden neurons} = \ln(No. \text{ of neural in previous layer})$$

In this model, the weight initialization function is used as the randomized calculation for (0,1). Learning rate and momentum are set to 0.5 and 0.01, respectively. We set the maximum epochs to 200 and MSE error measure is chosen to measure the actual error on each training iteration of all network configurations for stopping the training.

5.3 Description of Experimental Results

This section provides a guideline to comprehend each part of the experiments.

5.3.1 Report Classification

We design the experiments for a collection and comparison of the results to find out whether our goal will be fulfilled or not. We therefore classify the experimental results into four sets. Each set of the results is derived from the experiment using various network topologies, which are based on the same network parameters. Each network configuration will be initialized and trained for five times. After the training is done, the testing data sets are evaluated and the best results are chosen for a presentation. Each data source is constructed to support the same purpose. The experimental results are presented in the order of data sources.

Four categories of the report of our experiments are described below.

- 1) Firstly, we mention the network training by using the traditional training method with traditional cost function. To train the network by applying this approach requires a lot of data to be fed as the network input. When the training is done, the set of generalized weights is given. We will use this set of weights to predict the future value of the testing set in the short-term period.

To experiment, the data set is divided into two parts: training and testing data sets. The training data set has approximately three times as many data series as the testing set.

For the generated data set, the Mackey-Glass time series; a test period is extended over 180 observations. For Ahold and IBM data sets, the test period is extended for approximately six months.

The evaluation and conclusion of the later experiment will be compared with this experiment based on our objectives.

- 2) Secondly, we mention the network training using the step-training method, which belongs to our approach. The concept the step-training method focuses on minimizing the traditional cost function.

In this experiment, we train the network with more than one training data set at the same time and use the given set of weights to forecast only the future value of each testing data set.

An example of this training is given. If the 20-2-1 network is trained by fixing the training size to 40 samples and stipulating 100 input values to feed the network as its input, it means this training will give 60 sets of weights (60 training data sets were trained at the same time). The time-window will be moved one day ahead and the network is retrained on the updated training data set. The testing data set is the same as the stipulated set but the desired output of the testing data set is overlapped by the next training data set. See more detail in Section 3.2.1.

- 3) Thirdly, we mention the network learning using the step-training method by maximizing the fluctuation direction correctness of the target based on the traditional cost function. We expect that this section will give the best experimental result when compared with those of the first and second experiments.
- 4) Fourthly and finally, we have experimented with various training sizes, which affect the network performance, by picking up only the configuration that gives the best result in the third experiment display.

For each experiment, we present a graph of the specific network configuration based on the experimental results that are displayed in the table format. These configurations are chosen to plot the diagram because we consider the network configuration in the third experiment, which gives the best result. These figures will be used to draw general conclusions on the practicalities of the neural network for time series prediction using our approach. The actual, predicted and error values are presented in Section 5.4.

5.3.2 Phraseology Expression

The experimental reports to be displayed in Section 5.4 consist of the following columns:

- *Topology*: the notion such as " n_1 - n_2 - n_3 " is used to denote a network with n_1 input units, one hidden layer of n_2 units and n_3 output units. Thus n_1 consecutive values of data set are fed into the network to forecast the next value.
- *MSE* refers to a mean squared error value of the testing data set. A lowest value is preferable. It is possible to scale the MSE down to a scale in which a comparison can be made.
- *MAE* refers to a mean absolute error value of the testing data set. A lowest value is preferable. See more details in Section 2.3.1.
- *MAPE* refers to a mean absolute percentage error value of the testing data set. A lowest value is preferable. See more details in Section 2.3.1.
- *Tolerance 1%*: if this column being 40% means that the neural network is able to predict the future value with an accuracy of 40% with tolerance of 1%. See more details in Section 2.3.1.
- *Tolerance 5%*: if this column being 85% means that the neural network is able to predict the future value with an accuracy of 85% with tolerance of 5%. See more details in Section 2.3.1.
- *POCFD* represent the percentage of prediction of correct fluctuation direction. The POCFD close to 100% is preferable. See more details in Section 2.3.1.
- *Patterns* are the number of test patterns.
- *Training Sizes* are the number of inputs in the training data set.

Section 5.4 presents a few experimental results and we will discuss the results in the next chapter.

5.4 Results

In this section, we describe the results of testing the total of three data sets, in which they are divided into four subsections for further classification of the performance comparison. The results reveal the different error measures as described in chapter 3.

5.4.1 Source: Mackey-Glass Time Series

The first set of networks is trained by using the traditional training method with traditional cost function, and the second one by the step-training method with the traditional and the TSCFD cost functions, respectively.

It can be seen from Table 5.1 that the best performance outcomes are obtained from those five configurations over the Mackey-Glass time series after performing with the traditional training method and a traditional cost function. The Mackey-Glass data series during the period of 1st January 2000 to 1st November 2000 are given to the network as inputs. The result of the 6-3-1 network reveals that it can reach only 8% accuracy by the Tolerance 1% and 8% accuracy by the Tolerance 5%. The results of all networks produce much worse outputs if we consider the prediction values of the Tolerance 1% and the Tolerance 5%.

Table 5.1 Configurations and Error Summary Applying Traditional Training Method with a Traditional Cost Function for the Mackey-Glass Time Series

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	0.091	0.228	32.938	2%	19%	57%	181	306
5-4-1	0.151	0.300	42.961	6%	21%	57%	181	306
6-3-1	0.113	0.256	36.842	8%	25%	60%	181	306
10-3-1	0.124	0.277	39.463	4%	19%	61%	181	306
15-3-1	0.173	0.347	47.868	3%	9%	57%	181	306
20-2-1	0.095	0.237	34.138	4%	20%	59%	181	306

In Figure 5.4 illustrates graphs showing the prediction accuracy of the 20-2-1 network for the values of the Mackey-Glass time series over the test period. Figure 5.4 (a) reveals both the actual and predicted values, (b) displays the different (error) values in (a). The 181 output data were produced based on the 20 input values.

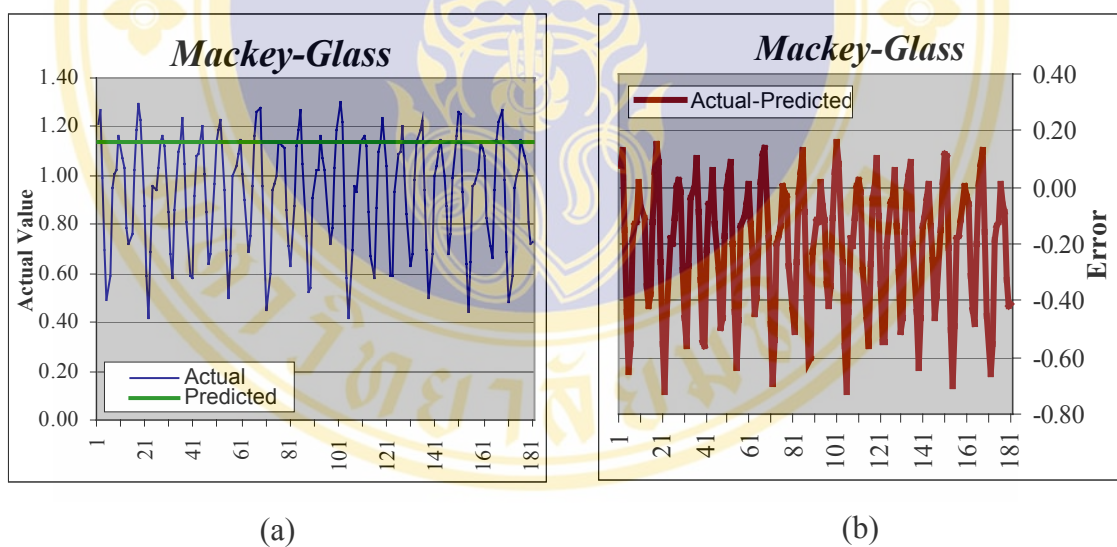


Figure 5.4 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.1
(a) Both the Actual and Predicted Values, (b) The Different (error) Values in (a)

Table 5.2 reveals the consequences of the neural network trained to forecast Mackey-Glass time series by minimizing a traditional cost function, applying on the step-training method. The Mackey-Glass data series during the period of 1st July 2000 to 1st November 2000 are given to the network as inputs. These obviously demonstrate that the MSE, MAE, MAPE, Tolerance 1% and Tolerance 5% produce

slightly better results than those displayed in Table 5.1, using the traditional training method based on a traditional cost function.

Table 5.2 Configurations and Error Summary Applying Step-Training Method with a Traditional Cost Function for the Mackey-Glass Time Series

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	0.030	0.149	18.433	5%	20%	59%	109	15
5-4-1	0.091	0.218	28.016	5%	20%	48%	109	15
6-3-1	0.035	0.149	18.102	6%	21%	69%	99	25
10-3-1	0.065	0.192	23.418	8%	21%	51%	99	25
15-3-1	0.107	0.244	30.116	2%	14%	53%	94	30
20-2-1	0.034	0.153	18.702	2%	24%	44%	89	35

Figure 5.5 illustrates graphs showing the prediction accuracy of the 20-2-1 network for the values of the Mackey-Glass time series over the test period. Figure 5.5 (a) reveals both the actual and predicted values, (b) displays the different (error) values in (a). The 89 outputs data were produced based on the 20 input values.

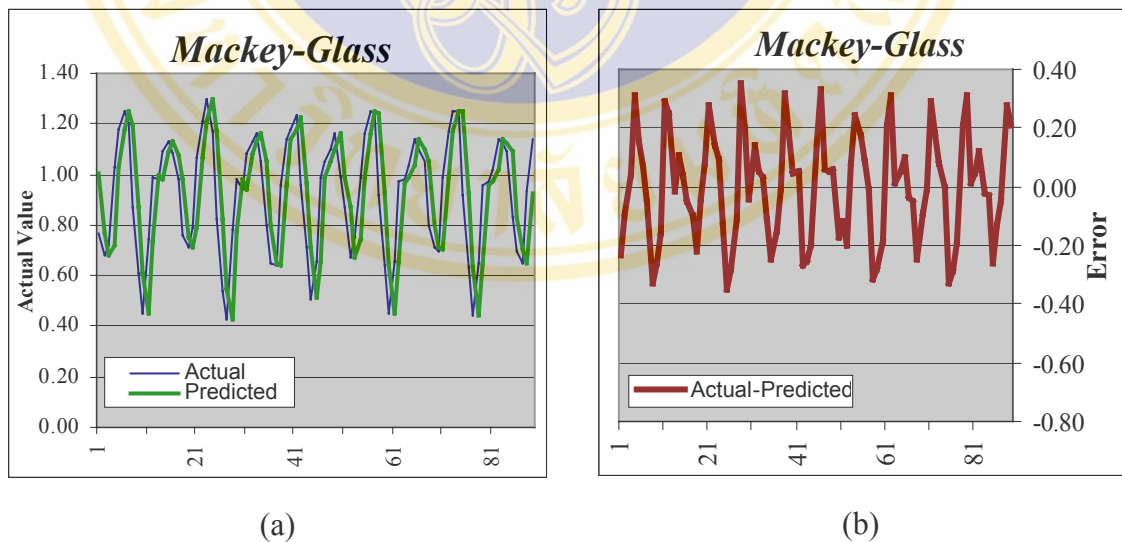


Figure 5.5 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.2
(a) Both the Actual and Predicted Values, (b) The Different (error) Values in (a)

Table 5.3 displays the results of the testing data sets using a TSCFD cost function and the step-training method. Descriptive performance measures of MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and percentages of POCFD are calculated. The Mackey-Glass data series during the period of 1st July 2000 to 1st November 2000 are given to the network as inputs.

All predicted values of the Tolerance 5% and percentages of the POCFD outperform those received from using the step-training method with a traditional cost function. The percentages of a POCFD of the 20-2-1 network reaches 86%, and this network provides 70% accuracy by Tolerance 1% and 79% accuracy by Tolerance 5%.

Table 5.3 Configurations and Error Summary Applying Step-Training Method with a TSCFD Cost Function for the Mackey-Glass Time Series

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	0.010	0.077	8.844	7%	44%	81%	109	15
5-4-1	0.088	0.178	25.198	3%	28%	75%	109	15
6-3-1	0.018	0.083	11.190	12%	58%	81%	99	25
10-3-1	0.056	0.156	22.094	7%	30%	76%	99	25
15-3-1	0.094	0.220	28.868	6%	18%	73%	94	30
20-2-1	0.010	0.046	5.303	70%	79%	86%	89	35

Figure 5.6 illustrates graphs showing the neural network trained to minimize a TSCFD cost function of the 20-2-1 network for the values of the Mackey-Glass time series over the test period. Figure 5.6 (a) reveals both the actual and predicted values, (b) displays the different (error) values in (a).

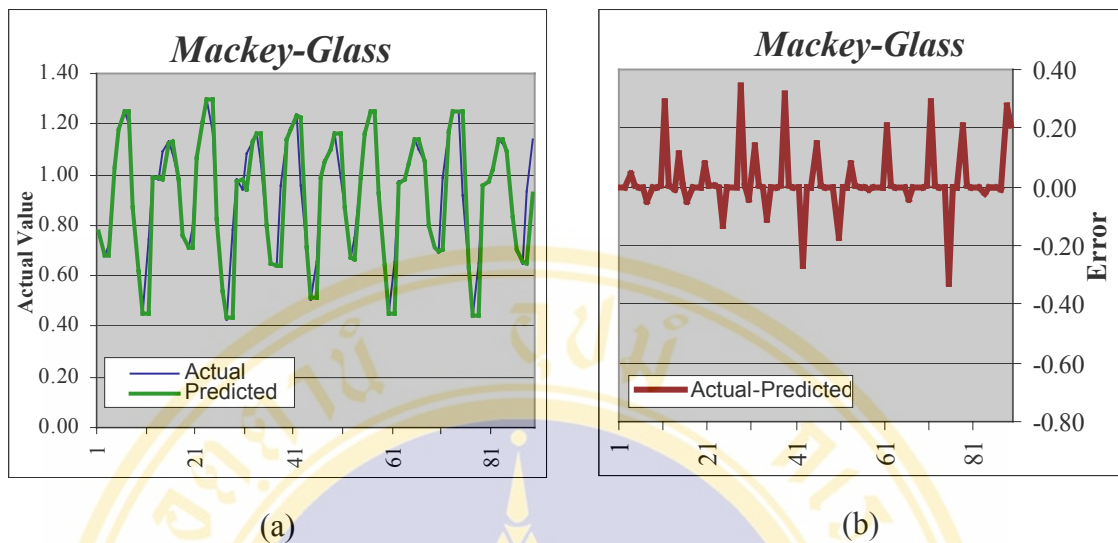


Figure 5.6 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.3
 (a) Both the Actual and Predicted Values, (b) The Different (error) Values in (a)

Table 5.4 displays the results in which the training sizes vary with the testing data sets using a TSCFD cost function and the step-training method. These results display an output based on 20 input values. The Mackey-Glass data series shown during the period of 1st July 2000 to 1st November 2000 is given to the network as inputs. The networks trained on a training size of 50 samples turn out to perform best. This conclusion is based on the quality of the prediction in terms of Tolerance 1% and accuracy percentages of POCFD for each of the 74 outputs.

Table 5.4 The Mackey-Glass Time Series Performances with Varying Training Size, Applying Step-Training Method with a TSCFD Cost Function

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
20-2-1	0.011	0.048	5.516	68%	78%	84%	99	25
20-2-1	0.010	0.047	5.321	69%	78%	85%	94	30
20-2-1	0.010	0.046	5.303	70%	79%	86%	89	35
20-2-1	0.011	0.049	5.563	69%	79%	89%	84	40
20-2-1	0.012	0.051	5.836	71%	76%	83%	79	45
20-2-1	0.011	0.048	5.455	73%	78%	89%	74	50

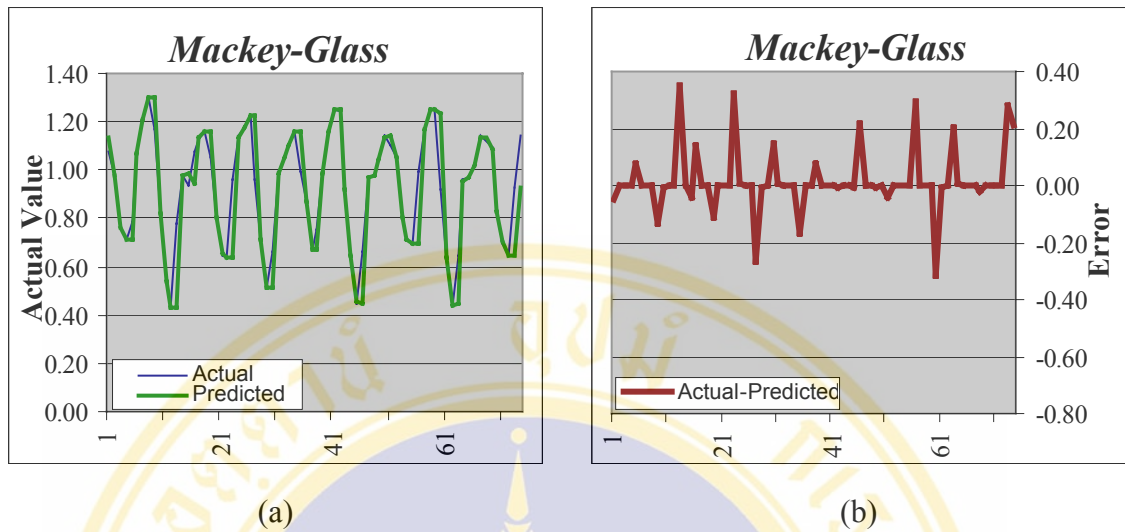


Figure 5.7 The Prediction Accuracy of the 20-2-1 Network Selected from Table 5.4, Trained on a Training Size of 50 Samples

(a) Both the Actual and Predicted Values, (b) The Different (error) Values in (a)

5.4.2 Source: Ahold Stock Price

Table 5.5 shows the best performance of all seven configurations over the Ahold data sets after using the traditional training method with the traditional cost function. The stock values during the period of 1st January 1995 to 1st July 1996 are given to the network as inputs.

The MSE, MAE and MAPE yield the highest accuracy with the 20-2-1 network. The result of 20-3-1 network reveals that it can reach only 15% accuracy by Tolerance 1% and 56% accuracy by Tolerance 5%. The results of overall networks reveals much worse outputs if we consider the predicted values of the Tolerance 1% and the Tolerance 5%.

Table 5.5 Configurations and Error Summary Applying Traditional Training Method with the Traditional Cost Function for an Ahold Data Set

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	15.198	3.351	5.702	6%	48%	56%	125	377
5-4-1	13.056	3.108	5.329	6%	43%	56%	125	377
6-3-1	15.678	3.412	5.810	6%	47%	56%	125	377
10-3-1	11.336	2.826	4.907	15%	50%	58%	123	377
15-3-1	12.749	2.951	5.230	14%	54%	58%	123	377
20-2-1	25.180	4.393	7.477	2%	29%	55%	121	377
20-3-1	18.294	3.367	6.177	15%	56%	63%	121	377

In Figure 5.8, the graph shows the prediction of 20-2-1 network for the closing stock prices that belong to the Ahold over the test period. The 121 outputs of data were produced based on the 20 input values.

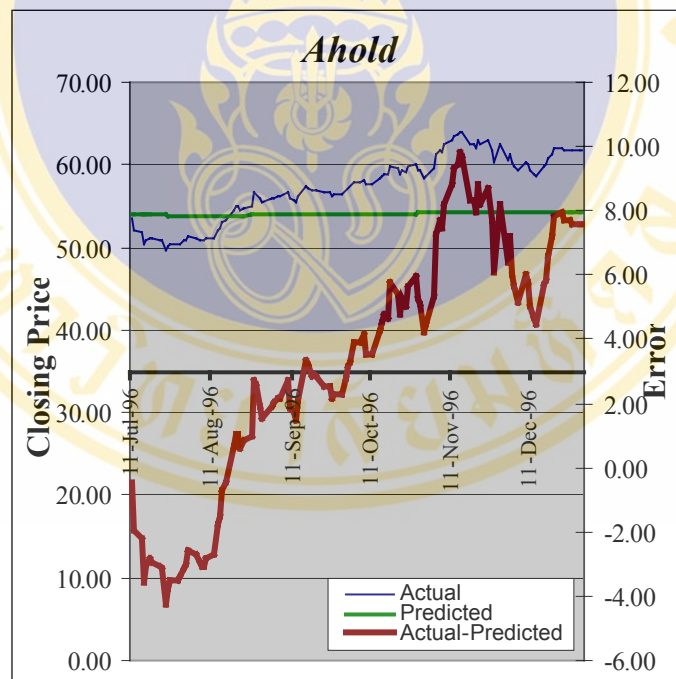


Figure 5.8 The Error Value of the 20-2-1 Network Selected from Table 5.5

In Table 5.6, the neural network is trained to forecast the Ahold closing stock prices by minimizing the traditional cost function, applying to the step-training method. The stock values during the period of 1st January 1996 to 1st July 1996 are given to the network as inputs.

These results give only a slight difference in percentages of a POCFD. If we consider the percentages of a POCFD, the results that we obtain are slightly worse than we have expected. But all these obviously reveal that the MSE, MAE, MAPE, Tolerance 1% and Tolerance 5% yield much better results than those displayed in Table 5.5, which are derived from using the traditional training method based on an MSE cost function. The Tolerance 1% produces the highest accuracy with the 20-2-1 network.

Table 5.6 Configurations and Error Summary Applying Step-Training Method with a Traditional Cost Function for the Ahold Data Set

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	0.932	0.705	1.448	47%	97%	55%	112	15
5-4-1	0.959	0.679	1.386	50%	96%	51%	112	15
6-3-1	0.824	0.644	1.319	51%	96%	55%	112	15
10-3-1	1.350	0.834	1.679	40%	95%	51%	102	25
15-3-1	1.411	0.844	1.680	39%	95%	56%	97	30
20-2-1	0.879	0.664	1.332	53%	97%	55%	92	35
20-3-1	2.174	1.043	2.050	39%	92%	54%	92	35

Figure 5.9 shows the predicted and the desired values derived from the 20-2-1 network versus the actual error. The data of 92 outputs were produced based on 20 input values

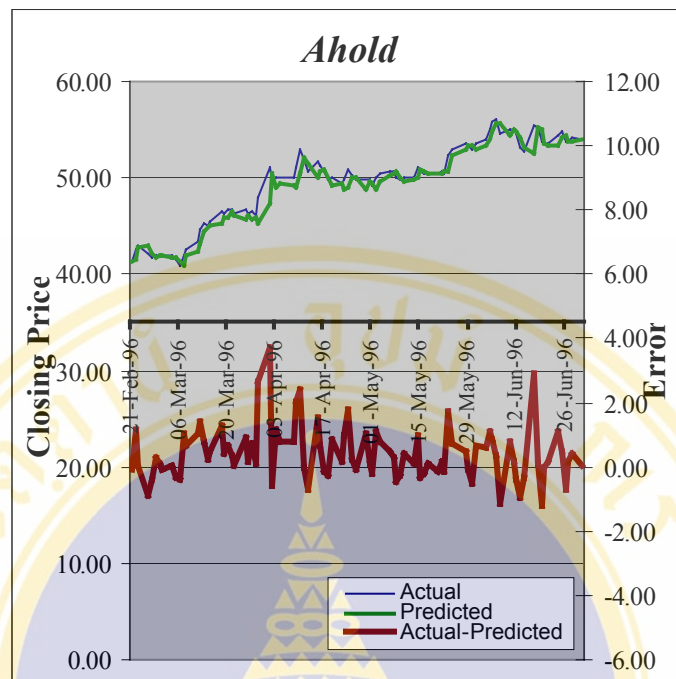


Figure 5.9 The Error Value of the 20-2-1 Network Selected from Table 5.6

Table 5.7 displays the results of the testing data sets using a TSCFD cost function and the step-training method. Descriptive performance measures of the MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and accuracy percentages of POCFD are calculated. The stock values during the period of 1st January 1996 to 1st July 1996 are given to the network as inputs.

All predicted values of the MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and percentages of POCFD outperform the predicted values derived from using the step-training method with a traditional cost function. The Tolerance 5% of the 5-2-1 network achieves 100% accuracy and the 20-2-1 network gives 68% accuracy by the Tolerance 1%, 98% accuracy by the Tolerance 5% and 73% accuracy by a POCFD.

Table 5.7 Configurations and Error Summary Applying Step-Training Method with a TSCFD Cost Function for the Ahold Data Set

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	0.600	0.571	1.176	54%	100%	61%	112	15
5-4-1	0.569	0.524	1.070	60%	99%	70%	112	15
6-3-1	0.504	0.483	0.987	64%	98%	69%	102	25
10-3-1	1.032	0.717	1.438	47%	97%	68%	102	25
15-3-1	0.944	0.705	1.400	45%	97%	69%	97	30
20-2-1	0.491	0.461	0.920	68%	98%	73%	92	35
20-3-1	1.916	0.936	1.824	43%	92%	64%	92	35

In Figure 5.10, the neural network is trained to minimize a TSCFD cost function. The difference between actual stock prices and the forecast values is illustrated. The 92 output data were produced based on the 20 input values.

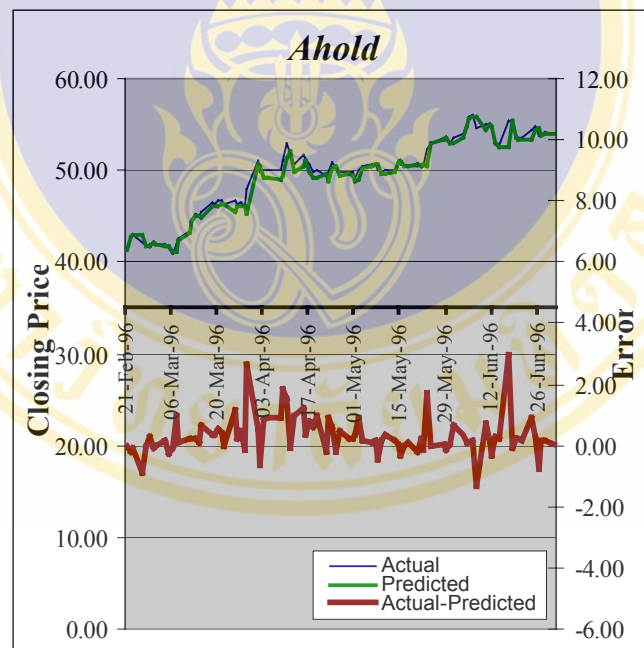


Figure 5.10 The Error Value of the 20-2-1 Network Selected from Table 5.7

Table 5.8 displays the results in which the training sizes are varied with the testing data sets using a TSCFD cost function and the step-training method. These results reveal an output based on the 20 input values. The stock values during the period of 1st January 1996 to 1st July 1996 are given to the network as inputs. The

networks trained on a training size of 25 samples turn out to perform best. This conclusion is based on the quality of the prediction in terms of MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and accuracy percentages of POCFD for each of the 102 outputs.

When we increase the number of samples of the training data set, the networks turn out to decrease in the predicted value accuracy of the Tolerance 1%, the Tolerance 5% and percentages of a POCFD while the predicted value accuracy of MSE, MAE and MAPE increase.

Table 5.8 The Ahold Data Series Performances with Varying Training Size, Applying Step-Training Method with a TSCFD Cost Function

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
20-2-1	0.294	0.305	0.613	81%	100%	76%	102	25
20-2-1	0.387	0.371	0.744	73%	99%	73%	97	30
20-2-1	0.491	0.461	0.920	68%	98%	73%	92	35
20-2-1	0.682	0.563	1.115	60%	98%	67%	87	40
20-2-1	0.808	0.663	1.309	55%	98%	67%	82	45
20-2-1	1.075	0.793	1.557	40%	97%	67%	77	50

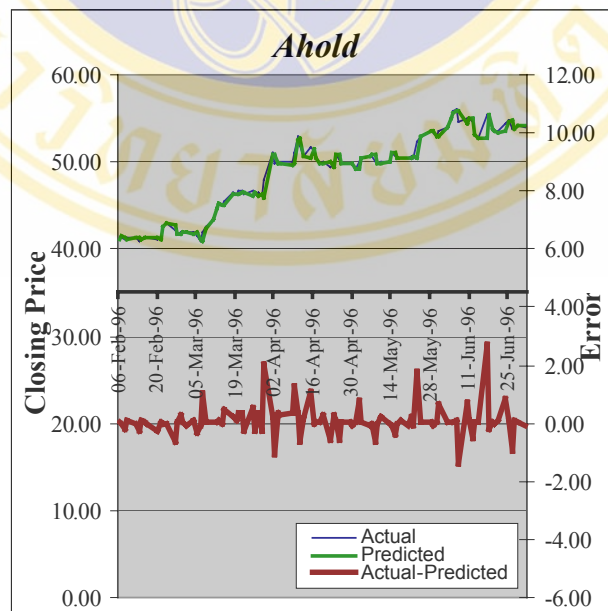


Figure 5.11 The Error Value of the 20-2-1 Network Selected from Table 5.8, Trained on a Training Size of 25 Samples

5.4.3 Source: IBM Stock Price

All experiments with the IBM data sets use the same set of configurations as the Ahold's for the training and the testing. The details of the experiments are as follows:

The best results of those five configurations over IBM data sets that we have obtained after performing with the traditional training method and a traditional cost function can be seen in Table 5.9. The stock values during the period of 1st January 2001 to 1st July 2002 are given to the network as inputs.

The MSE, MAE and MAPE yield the highest accuracy for the 15-3-1 network. The MSE reaches to 346.887. The result of the 6-3-1 network reveals that it can reach only 6% accuracy by the Tolerance 1% and 39% accuracy by the Tolerance 5%. The results of all networks reveal much worse outputs if we consider the predicted values of the Tolerance 1% and the Tolerance 5%.

Table 5.9 Configurations and Error Summary Applying Traditional Training Method with the Traditional Cost Function for the IBM Data Set

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	73.772	7.397	9.739	2%	20%	56%	124	373
5-4-1	54.612	6.268	8.325	6%	33%	55%	124	373
6-3-1	41.840	5.296	7.446	6%	39%	52%	124	373
10-3-1	43.926	5.528	7.547	8%	35%	55%	122	373
15-3-1	346.887	17.079	22.358	1%	5%	55%	122	373
20-2-1	66.941	6.973	9.330	5%	32%	53%	120	373
20-3-1	261.269	14.539	18.914	2%	9%	57%	120	373

In Figure 5.12, the graph shows the prediction accuracy of the 20-2-1 network for the closing stock prices that belong to the IBM over a test period. The 120 output data were produced based on the 20 input values.

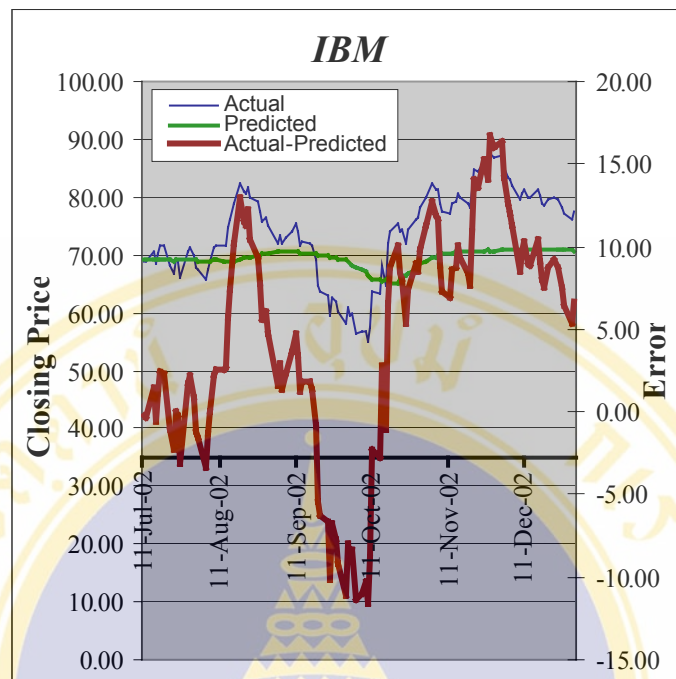


Figure 5.12 The Error Value of the 20-2-1 Network Selected from Table 5.9

Table 5.10 reveals the consequences of the neural network trained to forecast the IBM closing stock prices by minimizing a traditional cost function applying on the step-training method. The stock values during the period of 1st January 2002 to 1st July 2002 are given to the network as inputs.

These results reveal only a slight difference in percentages of the POCFD. If we consider the accuracy percentage of the POCFD, the results that we obtain are slightly worse than we have expected. But these obviously show that the MSE, MAE, MAPE, Tolerance 1% and Tolerance 5% yield results than those revealed in Table 5.9, using the traditional training method based on a traditional cost function. The Tolerance 1% and Tolerance 5% produce the highest accuracy for the 20-1-1 network.

Table 5.10 Configurations and Error Summary Applying Step-Training Method with a Traditional Cost Function for the IBM Data Set

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	9.310	2.394	2.754	23%	87%	50%	110	15
5-4-1	16.244	2.622	2.842	29%	86%	50%	110	15
6-3-1	11.183	2.588	3.032	28%	80%	48%	100	25
10-3-1	11.034	2.591	2.945	21%	81%	49%	100	25
15-3-1	43.472	4.106	4.643	14%	74%	51%	95	30
20-2-1	4.635	1.506	1.768	39%	94%	49%	90	35
20-3-1	35.360	4.297	4.885	16%	64%	55%	90	35

Figure 5.13 shows the predicted and desired values derived from the 20-2-1 network versus the actual error. The 90 output data were produced based on the 20 input values

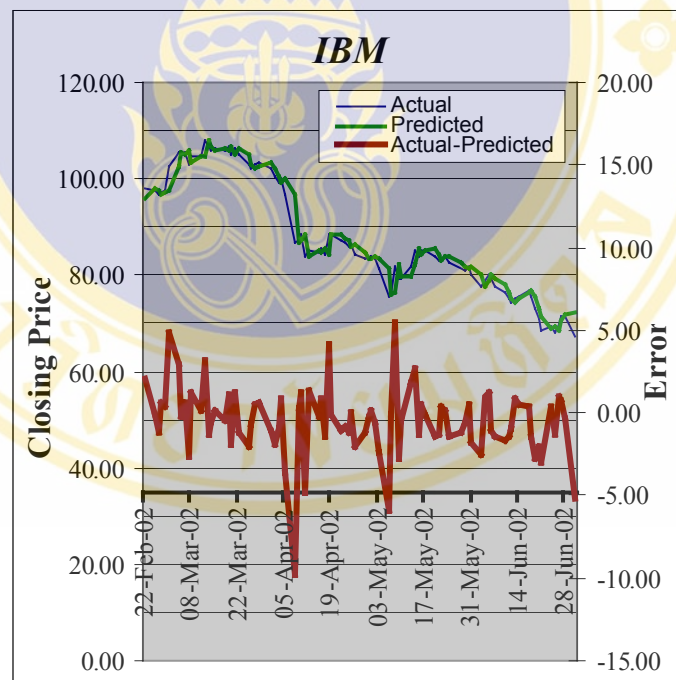


Figure 5.13 The Error Value of the 20-2-1 Network Selected from Table 5.10

Table 5.11 displays the results of the testing data sets using a TSCFD cost function and the step-training method. Descriptive performance measures of MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and accuracy percentages of POCFD are

calculated. The stock values during the period of 1st January 2002 to 1st July 2002 are given to the network as inputs.

All predicted values of MSE, MAE, MAPE, Tolerance 1%, Tolerance 5% and percentages of POCFD outperform those derived from using the step-training method with a traditional cost function. The accuracy percentages of a POCFD that belong to the 20-2-1 network reach 75% and this network provides 64% accuracy by the Tolerance 1% and 98% accuracy by the Tolerance 5%.

Table 5.11 Configurations and Error Summary Applying Step-Training Method with a TSCFD Cost Function for the IBM Data Set

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
5-2-1	6.173	1.926	2.207	33%	89%	64%	110	15
5-4-1	14.941	2.439	2.580	32%	87%	62%	110	15
6-3-1	7.828	2.274	2.644	25%	86%	59%	100	25
10-3-1	13.016	2.592	2.868	23%	85%	62%	100	25
15-3-1	32.996	3.888	4.366	17%	73%	61%	95	30
20-2-1	1.703	0.799	0.943	64%	98%	75%	90	35
20-3-1	43.671	4.869	5.611	11%	61%	61%	90	35

Figure 5.14 reveals an outcome of the neural network trained to minimize a TSCFD cost function. The difference between actual stock prices and the forecast values is illustrated.

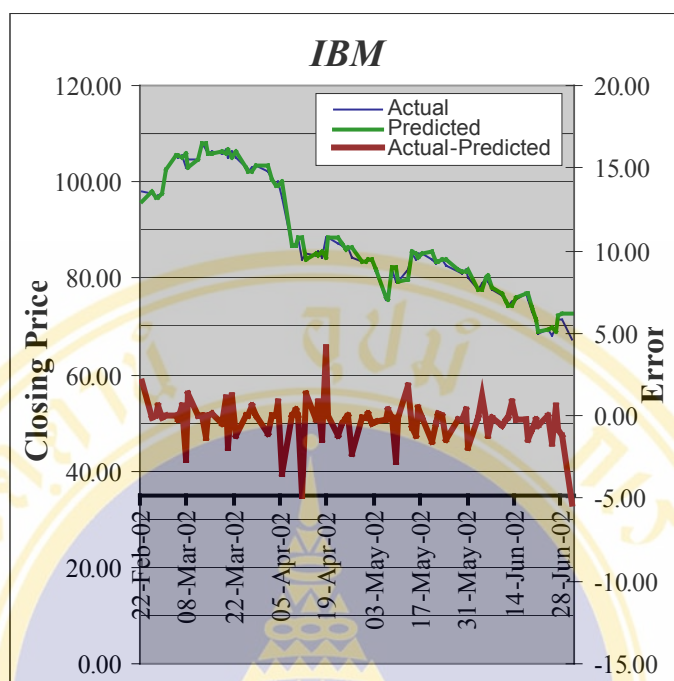


Figure 5.14 The Error Value of the 20-2-1 Network Selected from Table 5.11

Table 5.12 displays the results in which the training sizes are varied with the testing data sets using a TSCFD cost function and the step-training method. These results demonstrate an output based on 20 input values. The stock values during the period of 1st January 2002 to 1st July 2002 are given to the network as inputs. The networks trained on a training size of 30 samples turn out to perform best. This conclusion is based on the quality of the prediction in terms of MSE, MAE, MAPE, Tolerance 1% and Tolerance 5% for each of the 95 outputs.

Table 5.12 The IBM Data Series Performances with Varying Training Size, Applying Step-Training Method with a TSCFD Cost Function

Topology	MSE	MAE	MAPE	Tolerance 1%	Tolerance 5%	POCFD	Patterns	Training Sizes
20-2-1	1.571	0.747	0.864	66%	98%	77%	100	25
20-2-1	1.547	0.735	0.862	71%	98%	74%	95	30
20-2-1	1.703	0.799	0.943	64%	98%	75%	90	35
20-2-1	1.937	0.869	1.034	67%	98%	69%	85	40
20-2-1	2.004	0.899	1.081	65%	98%	71%	80	45
20-2-1	1.963	0.907	1.098	63%	97%	68%	75	50

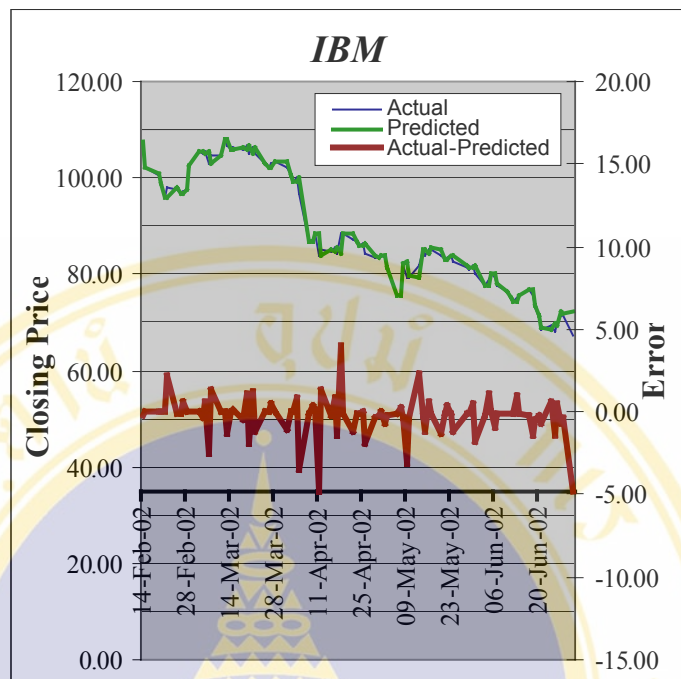


Figure 5.15 The Error Value of the 20-2-1 Network Selected from Table 5.12, Trained on a Training Size of 25 Samples

CHAPTER VI

DISCUSSION AND CONCLUSION

As mentioned in Chapter 2, the research objective is to determine the performance of different kinds of training methodologies, and the learning methods based on the adaptation of traditional cost function (TSCFD cost function), which are evaluated by using a large number of samples in the training data sets (training sizes) derived from three kinds of data sources. Based on the experimental results presented in the previous chapter, several points of discussion and conclusion are made in the following sections.

6.1 Discussion

This section discusses results of the neural network for a stock price prediction. Problems and difficulties, including prediction performance and limitations of the model development are explained. Details of each part are described below.

6.1.1 Problems and Difficulties

The problems and difficulties are found throughout the developmental phase. The performance of the training process may depend on the appropriateness of the network configuration setting, parameter selection and other controlled options. So, it is not clear where to set the optimum values for any network parameters.

It appears from the training time spent or error arising, which make it difficult to determine the network's forecasting ability. The only way to determine forecasting ability is to make a forecast and compare the results with known values as well as compute the performance by various performance measures.

6.1.2 Performance

The performance of the system is measured in terms of prediction accuracy, response time, storage usage and program complexity.

6.1.2.1 Prediction Accuracy

The aim of this research is to adjust the training and learning methodology of feedforward backpropagation neural network for time series prediction, which specially concerns both the value prediction accuracy and value fluctuation direction prediction correctness. The results presented in Chapter 5 indicate that better predictions could be obtained through the use of the step-training method and the TSCFD cost function approach in combination. In addition, to train the prediction model by using the TSCFD cost function is the best choice for our experiment.

We have found both many good and poor results while training the network. The training data set sizes and parameter value setting are important factors in training the networks. However, it can sometimes reduce the network's error, but never fulfils our goal.

6.1.2.2 Response Time

For the system implementation, we develop the program by absolutely separating the screen into two individual ones. The testing process is performed on the testing screen is called when a user would like to evaluate the prediction model of each training data set which completely processes through the training screen. The main focus is the response time the training process has taken. The response time of this process depends on various parameters such as number of input neurons, number of hidden neurons, learning rate and momentum factor. Approximately, we find out the response time per 1 training set does not exceed 1 minute.

However, the response time also depends on hardware performance and software operation used. Furthermore, the size of database has an effect on the response time. If the used database has a large size, we find that the modifying and searching time to access the database is quite slow.

6.1.2.3 Storage Usage

We present the summary of storage use of all components of the system in two groups as follows:

- **Program Coding**

The total size of this project (program coding) is approximately 90 Kbyte, which is quite small.

- **Database**

The size of database, Microsoft Access 2000, for storing all data of the prediction model depends on the amount of transactions submitted to the system. Once we have trained the network by using traditional training method, it means that we train the network with one training set of data at a time. But when we train the network by using the step training method that normally means we trained the network with more than one training set of data at the one time. From the experiment, we sometimes use to train the network by controlling the training process with batch file controller. This batch file is written for training a network with multiple training sets. This causes the data size to increase to 2 Gbyte. Nevertheless Microsoft Access has one engine to reduce the size of database that occurs, which occurs with many transactions (insert or update transaction). This is called by “Compact and Repair Database” menu. This point is our problem also.

6.1.2.4 Program Complexity

The program is written with Microsoft Visual Basic 6.0 and consists of two forms of related procedures. The constant and global variables are stored in the core module so that they can be referred to throughout the program.

The data on changes of database are stored in Relational Database Management System (Microsoft Access 2000) instead of text files. Consequently, when we want to add or modify the data structure of the system storage, we can almost do it without having to modify the program. So, it helps make the program not too complicate to be developed, while it is rather easy to maintain the system. Moreover, because this program is developed under Microsoft Window platform, it enables users to familiarize themselves with the program interfaces in a short time.

6.1.3 Limitations

The limitations of the model development are described below.

- The number of output neurons has been fixed to one because we intend to design the network with an ability to predict only the value for the next day.
- All records of the input data have the unique date that suites the program's structural requirements for feeding the model.
- The sigmoid activation function has been assigned to the hidden layer, which corresponds to the Min-Max normalization function that is used to normalize the input data.

6.2 Conclusion

As we understand, no model can achieve a perfect prediction in reality. In this research, we can only try to optimize the prediction strategy for improving the

accuracy of the prediction model by focusing on the training and learning methodology of the feedforward neural network model. The following conclusions can be drawn from this research work.

- Throughout the experiment, at the lowest level of MSE error in the training process, it doesn't mean that this network will give satisfactorily correct fluctuation direction of the target.
- A feedforward backpropagation neural network has been successfully applied to the time series of a one-day-ahead prediction using univariate input data.
- A longer training period does not always give a better result in a forecasting capability.
- A feedforward backpropagation neural network that is trained with TSCFD cost function shows a significantly high POCFD level.

After reviewing those problem statements and the proposed solution, we can conclude that all objectives are fully accomplished. We expect this research to be a reference for future work, may be in terms of an improvement of the financial time series prediction or a modification of the neural network algorithm.

CHAPTER VII

SUGGESTIONS FOR FUTURE WORK

In this section, we make some suggestions concerning the areas in which our work (model) can be developed or improved its predictive ability. The suggestions for the future extension of our work are as follows.

- Based on our experiment, it is found that a verification of multiple data sources, other network topologies and parameters are required, in order to be used to evaluate the results of current researches.
- The main focus of our research is rather on univariate prediction. Using the only closing prices of an individual stock without any additional indicator is a hard restriction though it is interesting by itself. Another reason is that it is easy to evaluate and compare the given results of the training and learning methodology with those derived from our approach. But if we are absolutely concerned about the problem of a stock price prediction, other stock price indicators may be required for improving the prediction accuracy. So, the training input screens of our model should be adjusted to better perform this task.
- Our proposed learning method by adjusting the traditional cost function (TSCFD) can be a key (factor) to future improvement of backpropagation neural network's forecasting efficiency. It is flexible and can be adapted to many existing learning algorithms. We hope that it may be useful to a solution to problems of time series prediction, such as future exchange rate prediction, weather forecasting or even time series prediction in general. We also hope that these directions will also provide a fruitful encouragement and a strong interest to the field of neural networks for time series prediction.

REFERENCES

1. Xiru Z, Kurt T. Non-Linear Time Series Prediction by Systematic Data Exploration on a Massively Parallel Computer. Santa Fe Institute. Technical Report: 94-07-045.
2. John P. A Neural Network Approach to Predicting Stock Performance [Final Project]. 2000.
3. Georg D. Neural Networks for Time Series Processing. Vienna: Dept. of Medical Cybernetics and Artificial Intelligence.
4. Op't Landt FW. Stock Price Prediction using Neural Networks [Master Thesis]. Leiden University; 1997Aug.
5. Eric AP. Time Series Forecasting with Feed-Forward Neural Networks: Guidelines and Limitations [Master Thesis in Computer Science]. Laramie: The Graduate School of The University of Wyoming; 2000 Jul.
6. Davey N, Hunt SP, Frank RJ. Time Series Prediction and Neural Networks. U.K.: University Hertfordshire, Hatfield; 1999.
7. Rothermich J. Financial Time-Series Prediction Using Negatively Correlated Neural Network Ensembles [Mini-project, MSc in Natural Computation]. University of Birmingham: School of Computer Science. 2002Jan.
8. Yao JT, Tan CL. A Study on Training Criteria for Financial Time Series Forecasting. New Zealand: Department of Information Systems, Massey University.
9. Yu T, Fujun X, Xuhui W, Yan-Qing Z. Web-based Fuzzy Neural Networks for Stock Prediction. USA: Department of computer Science, Georgia State University. Proc. of the 2nd International Workshop on Intelligent Systems Design and Applications, pp.169-174, Aug. 2002.
10. Clarence NW. Tan and Gerhard E. Witting. An Empirical Study of parametric Effect on an Experimental Backpropagation Stock Price Prediction Mode. 1993Apr. Hossein AT, Hossein P. Stock Price Prediction by Artificial

Neural Networks: A Study of Tehran's Stock Exchange (T.S.E).
Department of Accounting, Azad University of Kashan.

11. Yao JT, Tan CL, Hean-Lee P. Neural Networks For Technical Analysis: A Study On KLCI. Singapore: School of Computing, National University of Singapore; 1998Oct.
12. Yochanan S, Dorota W. Utilizing Artificial Neural Network Model to Predict Stock Markets [Working Paper]. New York: Department of Economics, The University of Pennsylvania; 2000Sep.
13. Efstahios K. Using Neural Network and Genetic Algorithms to Predict Stock Market Returns [Master Thesis in Computer Science]. 2001Oct.
14. Thomas H, Kenneth H. Predicting the Stock Market. Swedem: Department of Mathematics and Physics, Center of Mathematical Modeling, Malardalen University; 1998Aug. Technical Report Series: IMA-TOM-1997-07.
15. Marek D. Applications of Modular Neural Networks in Financial Management [Master's Thesis in Cybernetics and Artificial Intelligence]. Faculty of Electrical Engineering and Information, Technical University of Kosice. 1997Apr.
16. Hossein AT, Hossein P. Stock Price Prediction by Artificial Neural Networks: A Study of Tehran's Stock Exchange (T.S.E). Department of Accounting, Azad University of Kashan.
17. Masri A, Mohammad FN, Khairuddin O, Miswan S. The Effect of Returns Function on Individual Stock Price (KLSE) Prediction Model Using Neural Networks. Malaysia: Faculty of Technology & Information Sciences, University Kebangsaan Malaysia.
18. Nicholas AHG. Development of Neural Network Systems for Financial Modeling [Report Summit: Bachelor of Science, Computer Science]. 2001May
19. Guido JD, Trading on The Edge: Neural, Genetic, And Fuzzy Systems For Chaotic Financial Markets. John Wiley & Sons. Canada; 1994
20. Dave A, Geprge M. Artificial Neural Networks Technology. New York: Data & Analysis Center for Software, Rome Laboratory RL/C3C; 1995Aug.
21. Nicholas HG. Development of Neural Network Systems for Financial Modeling [Beachelor Report in Computer Science]; 2001May.

22. Fieldsend JE. Training Neural Network Beyond the Euclidean Distance, Multi-Objective Neural Network using Evolutionary Strategy. Department of Computer Science, University of Exeter; 1999.
23. Alan MP. Financial Time Series Prediction using Neural Networks: Approach to Data Pre-Processing.
24. Thomas K, Gottfried R. Time Series Forecasting Using Neural Network. Austria: Department of Applied Computer Science, Vienna University of Economics and Business Administration.
25. Yao JT, Tan CL. Guidelines for Financial Forecasting with Neural Networks. New Zealand: Department of Information Systems, Massey University.
26. Janet XG, Goran R, Lam KC. Forecasting Hong Kong House Prices: An Artificial Neural Network Vs Log-linear Regression Approach. Hong Kong: Department of Building and Construction, City University of Hong Kong.
27. Radu D, Zoran O. Efficient Design of Neural Networks for Time Series Prediction. Washington: School of Electrical Engineering and Computer Science, Washington State University. 99164-2752.
28. David M. Backpropagation. 2000Aug. Available from: www.csse.monash.edu.au
29. Sven FC. Training Artificial Neural Networks For Time Series Prediction Using Asymmetric Cost Functions [Technical Working Paper IWI-0202]. Germany: Institute of Business Information Systems, University of Hamburg. Proceedings (forthcoming) V1.02, Accepted at: ICONIP 2002.



APPENDIX

BACKPROPAGATION DERIVATION

This part presents an explanation of the mathematics, which underpins the backpropagation algorithm [29]. The key mathematical concepts required are *partial derivatives*, which are explained in Section A.4 , and the *chain rule*, which is explained in Section A.5 .

A.1 Definition of The Network

Consider the standard feedforward neural network shown in Figure A.1, also known as a multilayer perceptron (MLP). Notice that the neurons of the input layer are shown as simple black circles. This is to indicate that no processing occurs in these nodes: they serve only to introduce the inputs to the network.

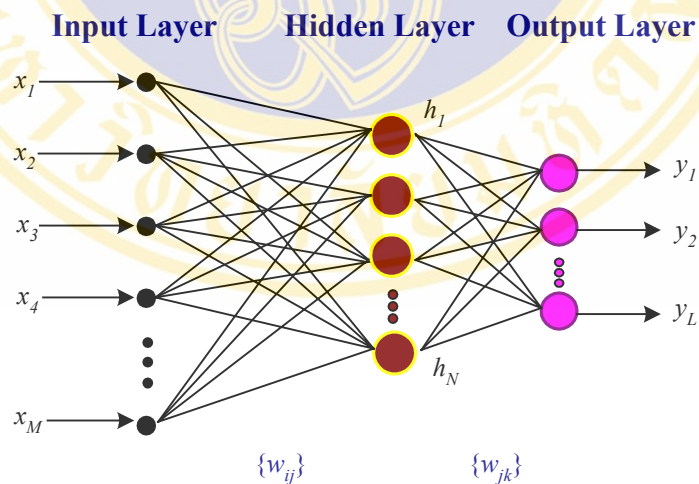


Figure A.1 Multilayer Perceptron: A Standard Feedforward Neural Network

Each node computes the weighted sum of its inputs, net , and uses this as the input to its transfer function $f()$. Consider a neuron k in the output layer. Its output, y_k , is given by

$$y_k = f(net_k) \quad (A.1)$$

where net_k , its net activation, is the weighted sum of the outputs of the nodes h_j in the hidden layer;

$$net_k = \sum_{j=1}^N w_{jk} h_j \quad (A.2)$$

Likewise, the output of each node j in the hidden layer is

$$h_j = f(net_j) \quad (A.3)$$

where

$$net_j = \sum_{i=1}^M w_{ij} x_i \quad (A.4)$$

and the x_i are the inputs to the network.

In this research, we used the $f(net_k)$ and $f(net_j)$ as below

$$f(net_k) = net_k$$

$$f(net_j) = \frac{1}{1 + e^{-net_j}}$$

A.2 Total Squared Error And Gradient Descent

For convenience, we can consider the inputs to the network as an *input vector* \mathbf{x} , where

$$\mathbf{x} = [x_1, x_2, \dots, x_M]^T$$

Likewise, the output of the network can be considered to be an *output vector* \mathbf{y} , where

$$\mathbf{y} = [y_1, y_2, \dots, y_L]^T$$

The *training set* for the network can then be considered to be a set of pairs of K input vectors \mathbf{x}_l and desired output vectors \mathbf{d}_l :

$$\text{training set} = \{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \dots, (\mathbf{x}_l, \mathbf{d}_l), (\mathbf{x}_K, \mathbf{d}_K)\}$$

Whenever an input vector from the training set \mathbf{x}_l is applied to the network, the network will produce an actual output y_l according to the equations in Section A.1. We may thus define the squared error for that input vector by summing over the *squared errors* at each output neuron:

$$[\text{squarederror}]_l = \frac{1}{2} \sum_{k=1}^L (d_k - y_k)^2$$

The factor of $\frac{1}{2}$ in front of the sum is introduced for computational convenience only, the aim will be to minimize the error, multiplying by a constant makes no difference to this goal. We can thus define the *mean of total squared error* E by summing over the all input/output pairs in the training set:

$$E = \frac{1}{2N} \sum_{l=1}^K \sum_{k=1}^L (d_{kl} - y_{kl})^2 \quad (\text{A.5})$$

The direction is determined by the derivative of E with respect to each component of w .

$$\text{Gradient of } E: \nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_o}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Our aim in training the network is to minimize E , by finding an appropriate set of weights $\{\{w_{jk}\}, \{w_{jk}\}\}$. We will do this by using a gradient descent algorithm: we will find out which direction is “downhill” on the error surface and modify each weight w so that we take a step in that direction. Mathematically, this means that each weight w will be modified by a small amount Δw in the direction of decreasing E :

$$w(t+1) = w(t) + \Delta w(t) \quad (\text{A.6})$$

where

$$\begin{aligned} \Delta w(t) &= -\eta \nabla E(w) \\ &= -\eta \left. \frac{\partial E}{\partial w} \right|_t \end{aligned}$$

Here $w(t)$ is the weight at time t and $w(t+1)$ is the updated weights. Equation A.6 is called the *generalized delta rule*. We see that the crucial thing we need in order to be able to perform gradient descent is the *partial derivative* $\frac{\partial E}{\partial w}$ of the error with respect to each weight.

The generalized delta rule is often augmented with a “momentum” term, which can increase the learning rate and help to avoid oscillations:

$$\Delta w(t) = -\eta \left. \frac{\partial E}{\partial w} \right|_t + \alpha \Delta w(t-1) \quad (\text{A.7})$$

The magnitude of α determines the effect past weight changes have on the current direction of change in weight space.

A.3 Finding The Partial Derivatives with Respect To The Weights

A.3.1 Weights Between The Hidden And Output Layers

Let us begin by considering a weight w_{jk} between a hidden neuron j and output neuron k . We wish to find $\frac{\partial E}{\partial w_{jk}}$. Using the *chain rule*, we may write

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}} \quad (\text{A.8})$$

Equation A.5 tells us that

$$\frac{\partial E}{\partial y_k} = d_k - y_k \quad (\text{A.9})$$

We see that

$$\frac{\partial y_k}{\partial net_k} = 1 \quad (\text{A.10})$$

From Equation A.2, we obtain (again taking the partial derivative has selected a particular term from the sum).

$$\frac{\partial net_k}{\partial w_{jk}} = h_j \quad (\text{A.11})$$

Substituting Equation A.9, A.10 and A.11 into Equation A.8, we see that

$$\frac{\partial E}{\partial w_{jk}} = (d_k - y_k) \cdot 1 \cdot h_j \quad (\text{A.12})$$

We have thus found the partial derivative of the error E with respect to weight w_{jk} in terms of known quantities and can employ this result in Equation A.6 in order to perform gradient descent for the weight $\{w_{kl}\}$ between the hidden and output layers.

A.3.2 Weights Between The Input And Hidden Layers

We now consider the weight w_{ij} between the input layer and the hidden layer. Again we start from Equation A.5 and apply the chain rule to find an expression for $\frac{\partial E}{\partial w_{ij}}$:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^L \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial net_k} \frac{\partial net_k}{\partial h_j} \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \quad (\text{A.13})$$

Note that on this occasion taking the partial derivative does not select a particular k from the sum, since all the outputs y_k depend on w_{ij} , as shown in Figure A.2.

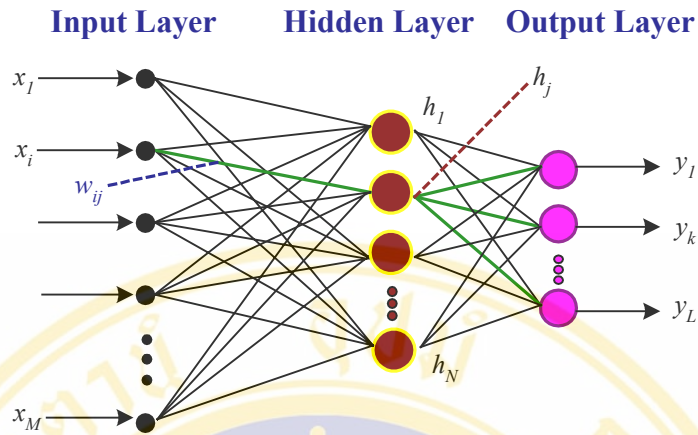


Figure A.2 All Output y_k Depend on each Weight w_{ij} Between the Input and the Hidden Layer

We have already found the first two terms of this sum in Section A.3.1. From Equation A.2 we obtain:

$$\frac{\partial net_k}{\partial h_j} = w_{jk} \quad (\text{A.14})$$

Using Equation A.25, derived in Section A.6 on the derivative of the sigmoid, we see that

$$\frac{\partial h_j}{\partial net_j} = h_j (1 - h_j) \quad (\text{A.15})$$

Finally, using Equation A.4, we find that

$$\frac{\partial net_j}{\partial w_{ij}} = x_i \quad (\text{A.16})$$

Substituting all these results into Equation A.13, we obtain the desired result:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^L (d_k - y_k) w_{jk} h_j (1 - h_j) x_i \quad (\text{A.17})$$

We have thus found the partial derivative of the error E with respect to weight w_{ij} in terms of known quantities (many of which we had already calculated in obtaining $\frac{\partial E}{\partial w_{jk}}$). Together with Equation A.12 this gives us all the $\frac{\partial E}{\partial w}$ needed so that Equation A.6 can be used to perform gradient descent for all the weights in the network.

A.4 Partial Derivatives

Partial derivatives are defined as derivatives of a function of multiple variables when all but the variable of interest are held fixed during the differentiation.

$$\frac{\partial f}{\partial x_m} = \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_m + h, \dots, x_n) - f(x_1, \dots, x_m, \dots, x_n)}{h} \quad (\text{A.18})$$

This is probably easiest to understand via examples. Here we find partial derivatives with respect to x , so the other variables (here y and z) are treated as constants:

$$\begin{array}{lll} f(x, y) = x + y & g(x, y) = x^2 + y^2 & h(x, y, z) = x^2 y + xy^2 + x^3 z^2 \\ \frac{\partial f}{\partial x} = 1 & \frac{\partial g}{\partial x} = 2x & \frac{\partial h}{\partial x} = 2xy + y^2 + 3x^2 z^2 \end{array}$$

A.5 The Chain Rule

If $g(x)$ is differentiable at the point x and $f(x)$ is differentiable at the point $g(x)$, then $f \circ g$ is differentiable at x . Furthermore, let $y = f(g(x))$ and $u = g(x)$, then

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \quad (\text{A.19})$$

There are a number of related results, which also go under the name of “chain rules.” For example, if $z = f(x, y)$, $x = g(t)$, and $y = h(t)$, then

$$\frac{dz}{dt} = \frac{\partial z}{\partial x} \frac{dx}{dt} + \frac{\partial z}{\partial y} \frac{dy}{dt} \quad (\text{A.20})$$

The “general” chain rule applies to two sets of functions

$$\begin{aligned} y_1 &= f_1(u_1, \dots, u_p) \\ &\vdots \\ y_m &= f_m(u_1, \dots, u_p) \end{aligned} \quad (\text{A.21})$$

and

$$\begin{aligned} u_1 &= g_1(x_1, \dots, x_n) \\ &\vdots \\ u_p &= g_p(x_1, \dots, x_n) \end{aligned} \quad (\text{A.22})$$

Defining the $m \times n$ Jacobi matrix by

$$\left(\frac{\partial y_i}{\partial x_j} \right) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \quad (\text{A.23})$$

and similarly for $\frac{\partial y_i}{\partial u_j}$ and $\frac{\partial u_i}{\partial x_j}$ then gives

$$\left(\frac{\partial y_i}{\partial x_j} \right) = \left(\frac{\partial y_i}{\partial u_j} \right) \left(\frac{\partial u_i}{\partial x_j} \right) \quad (\text{A.24})$$

Equation A.19 also applies for partial derivatives, as is indicated in matrix form by Equation A.24.

Again, simple examples should help to make this clear. Consider the function

$$y = \sin(x^2)$$

Using the notation of Equation A.19, we may write

$$y = f(g(x))$$

where

$$f(u) = \sin(u)$$

and

$$g(x) = u = x^2$$

We then have

$$\frac{dy}{du} = \cos(u)$$

$$\frac{du}{dx} = 2x$$

Bringing these together and using Equation A.19, we obtain

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy}{du} \cdot \frac{du}{dx} \\ &= \cos(u) \cdot 2x \\ &= 2x \cos(x^2) \end{aligned}$$

Here is a second example:

$$y = (x - d)^2$$

We write

$$y = f(u)$$

where

$$f(u) = u^2$$

and

$$u = x - d$$

Differentiating;

$$\frac{dy}{du} = 2u \qquad \frac{du}{dx} = 1$$

We thus obtain

$$\begin{aligned} \frac{dy}{dx} &= 2u \cdot 1 \\ &= 2(x - d) \end{aligned}$$

A.6 The Chain Rule Derivative of The Sigmoid

Examining Equation A.3 we see that

$$f(x) = \frac{1}{1 + e^{-x}}$$

therefore

$$\begin{aligned} \frac{df}{dx} &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\ &= f(x) \left(\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right) \\ &= f(x) (1 - f(x)) \end{aligned} \tag{A.25}$$

It is particularly convenient computationally that the derivative of $f(x)$ can be expressed solely in terms of $f(x)$ itself, since this is usually already known.

BIOGRAPHY

NAME	Miss Prapaphan Pan-O
DATE OF BIRTH	28 June 1977
PLACE OF BIRTH	Suphanburi, Thailand
INSTITUTIONS ATTENDED	Rajamangala Institute of Technology, 1994-1998: Bachelor of Science (Computer Science) Mahidol University, 1998-2003: Master of Science (Computer Science)
POSITION&OFFICE	1999-Present, Inetasia Holding Ltd. Chidlom, Pleonchit Bangkok, Thailand. Position: System Analyst 1998-1999, Mahidol University Computing Center Prayathai Bangkok, Thailand Position: IT Coordinator