HIGHER ORDER NEURAL NETWORKS FOR FINANCIAL TIME SERIES PREDICTION

ROZAIDA GHAZALI

A thesis submitted in accordance with the requirement of Liverpool John Moores University for the degree of Doctor of Philosophy

School of Computing & Mathematical Sciences
Liverpool John Moores University

December 2007

DECLARATION

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

ACKNOWLEDGEMENTS

My thanks go firstly, as it should always be, to The Unique God, the One who blessed me with the ability to undertake and finally complete this work. My most sincere gratitude and appreciation to my supervisor, Dr. Abir Hussain, who pointed me to the exciting subject of Higher Order Neural Network, and for her excellent guidance and encouragement during the period of this research. I also owe a great debt to Dr. Dhiya Al-Jumeily, for his constructive comments on my thesis, and for providing me with additional computers which help to speed up the simulations progress. My gratitude also extends to The Director of School of Computing & Mathematical Sciences, Liverpool John Moores University, Professor Madjid Merabti for his assistance and advice through some of the issues in the PhD. In addition I acknowledge support and guidance by Dr. Wael El-Deredy, especially in the early stages of the PhD. In particular I would like to thank everyone involved in the writing up and giving valuable comments on my journals and proceedings; Dr. Abir Hussain, Dr. Dhiya Al-Jumeily, Professor Madjid Merabti, Dr. Wael El-Deredy, Dr. Panos Liatsis, and Dr. Hissam Taufik.

My special thanks are due to my sponsor, Universiti Tun Hussein Onn Malaysia for granting me the scholarship. My appreciation also goes to School of Computing & Mathematical Sciences, Liverpool John Moores University, which facilitates me with a pleasant working environment, machines, and resources. I would also like to convey my thanks to my colleagues, Ali Al-Fayadh and Adam Knowles for helping me with lots of things.

Finally, my heartiest and warm thanks to my family, who deserves great thanks for their immense support throughout my PhD time. My husband, Lokman Hakim Ismail, deserves my unending gratitude for his help on innumerable tolerance, patience, understanding and assistance. I thank my mother for her continuous advice and guidance, and my brothers and sisters for their encouragement. To my lovely and divine children, Yasmin Kamilia, Amirul Mukminin, and Sufi Dayana for their true understanding and sacrifice in uncountable ways.

ABSTRACT

Neural networks have been shown to be a promising tool for forecasting financial times series. Numerous research and applications of neural networks in business have proven their advantage in relation to classical methods that do not include artificial intelligence. What makes this particular use of neural networks so attractive to financial analysts and traders is the fact that governments and companies benefit from it to make decisions on investment and trading. However, when the number of inputs to the model and the number of training examples becomes extremely large, the training procedure for ordinary neural network architectures becomes tremendously slow and unduly tedious. To overcome such time-consuming operations, this research work focuses on using various Higher Order Neural Networks (HONNs) which have a single layer of learnable weights, therefore reducing the networks' complexity. In order to predict the upcoming trends of univariate financial time series signals, three HONNs models; the Pi-Sigma Neural Network, the Functional Link Neural Network, and the Ridge Polynomial Neural Network were used, as well as the Multilayer Perceptron. Furthermore, a novel neural network architecture which comprises of a feedback connection in addition to the feedforward Ridge Polynomial Neural Network was constructed. The proposed network combines the properties of both higher order and recurrent neural networks, and is called Dynamic Ridge Polynomial Neural Network (DRPNN). Extensive simulations covering ten financial time series were performed. The forecasting performance of various feedforward HONNs models, the Multilayer Perceptron and the novel DRPNN was compared. Simulation results indicate that HONNs, particularly the DRPNN in most cases demonstrated advantages in capturing chaotic movement in the financial signals with an improvement in the profit return over other network models. The relative superiority of DRPNN to other networks is not just its ability to attain high profit return, but rather to model the training set with fast learning and convergence. The network offers fast training and shows considerable promise as a forecasting tool. It is concluded that DRPNN do have the capability to forecast the financial markets, and individual investor could benefit from the use of this forecasting tool.

TABLE OF CONTENTS

		Page
Declaration		i
Acknowledgement		ii
Abstract		iii
Table of Contents		iv-vii
List of Figures		viii-x
List of Tables		xi-xii
List of Symbols		xiii
List of Published Journals/Proceedings		xiv-xv
CHAPTER 1: INTRODUCTION		**1-8**
1.1	Neural Network and its Application to Financial Time Series Prediction	1-3
1.2	Dynamic Ridge Polynomial Neural Network	3-4
1.3	Problem Statements	4-5
1.4	Aims and Research Challenge	6
1.5	Objectives and Contribution of the Thesis	6
1.6	Scope and Limitations	7
1.7	Thesis Structure	7-8
1.8	Chapter Summary	8
CHAPTER 2: NEURAL NETWORKS		**9-26**
2.1	Introduction	9-10
2.2	From Biological to Artificial Neuron	10-11
2.3	Components of Neural Networks	11-12
2.4	Different Structures of Neural Networks	12-25
	2.4.1 Feedforward Neural Networks	13-20
	2.4.1.1 Multilayer Perceptron	14-19
	2.4.1.2 Higher Order Neural Networks	19-20
	2.4.2 Recurrent Neural Networks	20-25
2.5	Chapter Summary	25-26
CHAPTER 3: HIGHER ORDER NEURAL NETWORKS		**27-49**
3.1	Introduction	27
3.2	The Properties of HONNs	27-31

3.3	Product Units in HONNs	31-32
3.4	Types of HONNs	33-49
	3.4.1 Functional Link Neural Network (FLNN)	33-39
	3.4.1.1 Learning Algorithm of FLNN	37
	3.4.1.2 FLNNs' Applications	38-39
	3.4.2 Pi-Sigma Neural Network (PSNN)	39-43
	3.4.2.1 Learning Algorithm of PSNN	41-42
	3.4.2.2 PSNNs' Applications	42-43
	3.4.3 Ridge Polynomial Neural Network (RPNN)	44-49
	3.4.3.1 Learning Algorithm of RPNN	46-47
	3.4.3.2 RPNNs' Applications	47-49
3.5	Chapter Summary	49
CHAPTER 4: DYNAMIC RIDGE POLYNOMIAL NEURAL NETWORK		**50-62**
4.1	Introduction	50
4.2	The Properties and Network Structure of DRPNN	50-53
4.3	Learning Algorithm of DRPNN	53-55
4.4	Issues of stability in DRPNN	55-56
4.5	The Stability Condition for DRPNN	56-62
4.6	Chapter Summary	62
CHAPTER 5: FINANCIAL TIME SERIES FORECASTING		**63-75**
5.1	Introduction	63
5.2	Time Series and their Properties	63-64
5.3	Financial Time Series	64-67
	5.3.1 Technical Data	65
	5.3.2 Fundamental Data	66
	5.3.3 Derived Entities	66-67
5.4	The Prediction of Financial Time Series	67-70
5.5	Conventional Prediction Methods and Neural Networks	70-72
5.6	Application of Neural Networks in Financial Time Series	72-75
5.7	Chapter Summary	75
CHAPTER 6: EXPERIMENTAL DESIGN		**76-98**
6.1	Introduction	76

6.2	Variable Selection	76-77
6.3	Data Selection	77-79
6.4	Data Pre-processing	79-86
6.5	Data Partition	86-87
6.6	Network Models Topology	87-89
6.7	Training of the Networks	89-92
6.8	Model Selection	93
6.9	Performance Metrics.	93-97
6.10	Chapter Summary	98
CHAPTER 7: SIMULATION RESULTS AND ANALYSIS		**99-170**
7.1	Introduction	99
7.2	Prediction of Stationary Signals	99-131
7.2.1	One Step Ahead Prediction using Stationary Signals	99-115
7.2.1.1	Best Average Simulation Results	100-107
7.2.1.2	Best Single Simulation Results	108-115
7.2.2	Five Steps Ahead Prediction using Stationary Signals	116-131
7.2.2.1	Best Average Simulation Results	116-123
7.2.2.2	Best Single Simulation Results	124-131
7.3	Prediction of Non-stationary Signals	132-164
7.3.1	The One Step Ahead Prediction using Non-stationary Signals	132-148
7.3.1.1	Best Average Simulation Results	132-140
7.3.1.2	Best Single Simulation Results	141-148
7.3.2	The Five Steps Ahead Prediction using Non-stationary Signals	149-164
7.3.2.1	Best Average Simulation Results	149-156
7.3.2.2	Best Single Simulation Results	157-164
7.4	Discussions	165-170
7.4.1	Why Neural Networks Make Better Profits with Stationary Signals?	165-166
7.4.2	Why Forecasting of Five Steps Ahead Make Better Profits than Forecasting of One Step Ahead?	166
7.4.3	Why Some Neural Networks with Good AR Produce High NMSE and Low SNR?	166-167
7.4.4	How do DRPNNs and RPNNs Compare?	167-168

7.4.5	How do HONNs and MLPs Compare?	168-169
7.4.6	Why DRPNNs Made the Best Profit than Other Network Models?	169-170
7.5	Chapter Summary	170
CHAPTER 8: CONCLUSIONS AND FUTURE WORK		**171-177**
8.1	Introduction	171
8.2	Main Conclusions Derived from this Research Study	171-173
8.3	Novelty and Research's Contribution	173-174
8.4	Future Research Directions	174-176
8.5	Chapter Summary	177
REFERENCES		**178-186**
APPENDICES		

LIST OF FIGURES

Chapter 2

Figure 2.1: Basic features of biological neurons

Figure 2.2: Network with single perceptron or node

Figure 2.3: MultiLayer Perceptron

Figure 2.4: Basic principle of gradient descent

Figure 2.5: Fully-Connected RNN

Figure 2.6: (a) Elman and (b) Jordan networks

Figure 2.7: A four-layered partially RNN

Chapter 3

Figure 3.1: Groups of HONNs; (a) Input layer with combination of external inputs and their products, (b) Output layer of product unit, (c) Hidden layer of product units

Figure 3.2 (a): Truth table for XOR problem, 3.2 (b): 2^{nd} order HONN with two inputs

Figure 3.3: Linear separation of the input data for the XOR problem using 2^{nd} order HONN

Figure 3.4: (a) 1^{st} order weight correlation, (b) 3^{rd} order weight correlation

Figure 3.5: (a) The FLNN of type functional expansion model, (b) the FLNN of type tensor product model.

Fig. 3.6: Pi Sigma Neural Network of *K*-th order.

Figure 3.7: The Ridge Polynomial Neural Network of *k*-th order

Chapter 4

Figure 4.1: Dynamic Ridge Polynomial Neural Network of *k*-th order.

Chapter 5

Figure 5.1: A general method of performing time series prediction with *n* inputs, where *F(S)* is the activation function, $X_{(t-n)}, \ldots, X_{(t-2)}, X_{(t-1)}, X_{(t)}$ are the input vectors, and *X(t+1)* is the predicted output.

Chapter 6

Figure 6.1 (Part 1): Signal before and after pre-processing

Figure 6.1 (Part 2): Signal before and after pre-processing

Figure 6.2 (Part 1): Histograms of the signal before and after pre-processing

Figure 6.2 (Part 2): Histograms of the signal before and after pre-processing.

Figure 6.3: Learning the non-stationary signal with a neural network

Figure 6.4: Learning the stationary signal with a neural network

Chapter 7

Figure 7.1: Best average annualized return from all network models

Figure 7.2: Networks' performance on the AR with increasing order / number of hidden nodes

Figure 7.3: Networks' performance on the NMSE with increasing order / number of hidden nodes

Figure 7.4: Best ARs (including the transaction cost)

Figure 7.5: Learning curves for the prediction of all signals using DRPNNs

Figure 7.6: Best forecasts made by DRPNN on all data signals

Figure 7.7: Histograms of the signals error on all data sets using DRPNNs

Figure 7.8: Best average annualized return from all network models

Figure 7.9: Networks' performance on the AR with increasing order / number of hidden nodes

Figure 7.10: Networks' performance on the NMSE with increasing order / number of hidden nodes

Figure 7.11: Best AR (including the transaction cost)

Figure 7.12: Learning curves for the prediction of all signals using DRPNNs

Figure 7.13: Best forecasts made by DRPNNs on all data signals

Figure 7.14: Histograms of the signals error on all data sets using DRPNNs

Figure 7.15: Best average annualized return from all network models

Figure 7.16: Networks' performance on the AR with increasing order / number of hidden nodes
Figure 7.17: Networks' performance on the NMSE with increasing order / number of hidden nodes
Figure 7.18: Best AR (including the transaction cost)
Figure 7.19: Learning curves for the prediction of all signals using DRPNNs
Figure 7.20: Best forecasts made by DRPNNs on all data signals
Figure 7.21: Histograms of the signals error on non-stationary data using DRPNNs
Figure 7.22: Best average annualized return from all network models.
Figure 7.23: Networks' performance on the AR with increasing order / number of hidden nodes
Figure 7.24: Networks' performance on the NMSE with increasing order / number of hidden nodes
Figure 7.25: Best AR (including the transaction cost)
Figure 7.26: Learning curves for the prediction of all signals using DRPNNs
Figure 7.27: Best forecasts made by DRPNN on all data signals
Figure 7.28: Histograms of the signals error on non-stationary data using DRPNNs

LIST OF TABLES

Chapter 6

Table 6.1: Financial time series signals used

Table 6.2: Calculations for transformation of input and output variables

Table 6.3 (a) Data partition for stationary signals

Table 6.3 (b) Data partition for non-stationary signals

Table 6.4: The learning parameters used in all neural networks

Table 6.5: Performance metrics and their calculations

Chapter 7

Table 7.1: Best average result from the MLPs

Table 7.2: Best average result from the FLNNs

Table 7.3: Best average result from the PSNNs

Table 7.4: Best average result from the RPNNs

Table 7.5: Best average result from the DRPNNs

Table 7.6: The average maximum epoch reached during training

Table 7.7: Number of trainable weights and bias used in all networks

Table 7.8: Best single simulation based on the AR for the MLPs

Table 7.9: Best single simulation based on the AR for the FLNNs

Table 7.10: Best single simulation based on the AR for the PSNNs

Table 7.11: Best single simulation based on the AR for the RPNNs

Table 7.12: Best single simulation based on the AR for the DRPNNs

Table 7.13: CPU time usage for training each neural network

Table 7.14: Best average result from the MLPs

Table 7.15: Best average result from the FLNNs

Table 7.16: Best average result from the PSNNs

Table 7.17: Best average result from the RPNNs

Table 7.18: Best average result from the DRPNNs

Table 7.19: The average maximum epoch reached during training

Table 7.20: Number of trainable weights and bias used in all networks
Table 7.21: Best single simulation based on the AR for the MLPs
Table 7.22: Best single simulation based on the AR for the FLNNs
Table 7.23: Best single simulation based on the AR for the PSNNs
Table 7.24: Best single simulation based on the AR for the RPNNs
Table 7.25: Best single simulation based on the AR for the DRPNNs
Table 7.26: CPU time usage for training each neural network
Table 7.27: Best average result from the MLPs
Table 7.28: Best average result from the FLNNs
Table 7.29: Best average result from the PSNNs
Table 7.30: Best average result from the RPNNs
Table 7.31: Best average result from the DRPNNs
Table 7.32: The average maximum epoch reached during training
Table 7.33: Number of trainable weights and bias used in all networks
Table 7.34: Best single simulation based on the AR for the MLPs
Table 7.35: Best single simulation based on the AR for the FLNNs
Table 7.36: Best single simulation based on the AR for the PSNNs
Table 7.37: Best single simulation based on the AR for the RPNNs
Table 7.38: Best single simulation based on the AR for the DRPNNs
Table 7.39: CPU time usage for training each neural network
Table 7.40: Best average result from the MLPs
Table 7.41: Best average result from the FLNNs
Table 7.42: Best average result from the PSNNs
Table 7.43: Best average result from the RPNNs
Table 7.44: Best average result from the DRPNNs
Table 7.45: The average maximum epoch reached during training
Table 7.46: Number of trainable weights and bias used in all networks
Table 7.47: Best single simulation based on the AR for the MLPs
Table 7.48: Best single simulation based on the AR for the FLNNs
Table 7.49: Best single simulation based on the AR for the PSNNs
Table 7.50: Best single simulation based on the AR for the RPNNs
Table 7.51: Best single simulation based on the AR for the DRPNNs
Table 7.52: CPU time usage for training each neural network

LIST OF SYMBOLS

Y	Network output
X	Input vector
W	Weights matrix
W_{jo}, W_o, W_{oj}, W_{ok}	Bias
Σ	Summation function
Π	Multiplication function
σ	Nonlinear activation function
f'	First derivative of nonlinear activation function
ΔW	Delta weight
η or ε	Learning rate
μ or α	Momentum
E	Mean squared error

LIST OF PUBLICATIONS

International Journals:

1. R. Ghazali, A. J. Hussain, P. Liatsis and H. Tawfik (2007). The application of ridge polynomial neural network to multi-step ahead financial time series prediction. ***Neural Computing & Applications***, DOI: 10.1007/ s00521-007-0132-8
2. Ghazali, R., Hussain, A. J., Al-Jumeily, D., & Merabti, M. (2007). Dynamic Ridge Polynomial Neural Networks in Exchange Rates Time Series Forecasting. ***Lecture Notes in Computer Science.*** (4432), 123-132.
3. A. J. Hussain, D. Al-Jumeily and R. Ghazali (2007). Dynamic Ridge Polynomial Neural Networks for multi-step financial time series prediction. ***Accepted for publication at the International Journal of Intelligent Systems Technologies and Applications.***

International Proceedings:

1. A. J. Hussain, R. Ghazali and D. Al-Jumeily (2006). Dynamic Ridge polynomial neural network for financial time series prediction. ***Accepted for publication at the IEEE International conference on Innovation in Information Technology, IIT06, Dubai.***
2. R. Ghazali, A. Hussain, and M. Merabti (2006). Higher Order Neural Networks for Financial Time Series Prediction. *The **10th IASTED International Conference on Artificial Intelligence and Soft Computing***, Palma de Mallorca, Spain, pp. pages 119-124.
3. R. Ghazali, A. Hussain, and W. El-Deredy, (2006). Application of Ridge Polynomial Neural Networks to Financial Time Series Prediction. ***2006 IEEE World Congress on Computational Intelligence (IJCNN 2006)***, Vancouver, Canada. pp. 1892-1899.

CHAPTER 1: INTRODUCTION

1.1 Neural Network and its Application to Financial Time Series Prediction

Financial forecasting is a difficult task due to the intrinsic complexity of the financial system. While many time series may be approximated with a high degree of confidence, financial time series are found among the most difficult to be analyzed and predicted (Castiglione, 2000). This relates to the fact that stock markets are affected by many highly interrelated economic, political and even psychological factors, and these factors interact with each other in a very complex fashion. It is also very complicated to forecast the movement in the stock market (Yao et al, 1996). Despite the fact that it is difficult in practical applications, predicting financial time series data is still an issue of a much interest to both the economic and academic communities. Decisions regarding investments and trading by large companies and the economic policy of governments rely on computer modeling forecasts (Knowles, 2005). Commercial imperatives have ensured that financial time series prediction has been given a large amount of coverage in research literature and this will no doubt continue to be the case.

Various methods and techniques for the prediction of financial time series have been developed and are still being developed on the ground of these basic principles. From statistical to artificial intelligence, there are a range of techniques which have been used to make a forecast. The traditional methods for financial time series forecasting are based around statistical approaches. However, most of the developed prediction methods have very weak scientific support and were completely unsatisfactory due to the nonlinear nature of most of the financial time series (Dunis and Williams, 2002; Yao and Tan, 2000; Hellstrom and Holmstrom, 1998). Accordingly, throughout the last decade, neural networks have emerged from an esoteric instrument in academic research to a rather common tool assisting auditors, investors, port-folio managers and investment advisors in making critical financial decisions (Chen and Leung,

2005). Neural networks are powerful forecasting tools that draw on the most recent developments in artificial intelligent research. They are nonlinear models that can be trained to map past and future values of time series data thereby extract hidden structures and relationships that govern the data (Shachmurove and Witkowska, 2000). Using neural networks, complex relationships between input and output variables can be learned by machines without requiring a human being to specify the nature of the relationship. Neural networks have appeared as a powerful learning technique to perform complex task in highly nonlinear dynamic environments of financial time series. Financial service companies are becoming more and more dependent on computer technologies to establish and maintain competitiveness in a rapidly expanding global economy (Chen and Leung, 2005). In fact, most of the major investment banks, such as Goldman Sachs and Morgan Staley, have dedicated departments to the implementations of neural networks (Shachmurove and Witkowska, 2000). The fact that major companies in this financial industry are investing resources in neural networks indicates that artificial neural networks may serve as an important method for forecasting. The application of neural networks in time series prediction has shown better performance in comparison to statistical methods because of their nonlinear nature and training capability (Yumlu et al, 2005; Ho et al, 2002; Dunis and Huang, 2002). In addition, it has been shown that neural networks are universal approximators and have the ability to produce complex nonlinear mappings.

This research work examines the ability of High Order Neural Networks (HONNs) as a forecasting tool to predict the upcoming trends of financial time series data. The utilization of higher order terms allows the neural networks to expand the input space into a higher dimensional space where linear separability is possible, thus reducing the complexity of the network. The use of HONNs is circumvented by the fact that the higher the order of the network, the more complex the network becomes and learning is significantly slower. Functional Link Neural Network (FLNN) (Giles and Maxwell, 1987) is a type of HONN, which can use higher order correlations of the input components to perform nonlinear mappings using only a single layer of units. However, the network suffers from the combinatorial explosion in the number of weights, when the order of the network becomes excessively high. A simple yet

efficient alternative to FLNN is the Pi-Sigma Neural Network (PSNN) which was proposed by Ghosh and Shin (Ghosh and Shin, 1991-b). PSNN was introduced to overcome the problem of weight explosion in FLNN. The network has a regular structure and requires a smaller number of free parameters, when compared to other single layer HONNs. However, the Pi-Sigma Neural Network is not a universal approximator (Shin and Ghosh, 1995). A generalisation of PSNN is the Ridge Polynomial Neural Network (RPNN) (Shin and Ghosh, 1995). The network has a well regulated structure which is constructed by the addition of PSNNs of varying orders. Contrary to the FLNN, which utilizes multivariate polynomials, thus leading to an explosion in the number of free parameters, RPNN uses univariate polynomials which are easy to handle. RPNN is a universal approximator (Shin and Ghosh, 1995), and the network maintains the fast learning and powerful mapping properties of single layer HONNs and avoids the explosion of weights, as the number of inputs increases.

1.2 Dynamic Ridge Polynomial Neural Network

Applications in forecasting and signal processing require explicit treatment of dynamics. The behaviour of the financial signal itself related to some past inputs on which the present inputs depends. The inherent nonlinearity of financial time series can prevent a single neural network from being able to accurately forecast an extended trading period even if it could forecast changes in the testing data. To overcome the problems associated with neural networks when used for financial time series forecasting; in this research work, a new dynamically sized higher order recurrent neural network architecture is proposed. The network will start with small a basic structure, which will grow as the learning proceeds until the desired mapping task is carried out with the required degree of accuracy. The network is called the Dynamic Ridge Polynomial Neural Network (DRPNN). This becomes the novel aspect of this research in which the proposed Dynamic Ridge Polynomial Neural Network incorporates both higher order terms and a recurrent structure. In particular, this research work systematically investigates a method of pre-processing the financial signals in order to reduce the influence of their trends. The networks are

tested for the prediction of one and five steps ahead predictions of financial time series in which two methods are utilized; in the first method the data are passed directly to the neural network as non-stationary signals while in the second method the financial data are transformed into stationary signals. Ten financial time series are used in the simulation process. The performance of each network is evaluated using financial criteria for trading performance and standard statistical measures for forecasting accuracy.

1.3 Problem Statements

Although there have been a number of research advancements taken place in the area of neural networks applications, not all of which can be used in real time commercial applications. In practice, it appears that although many organizations have expressed interest in applying neural networks technology, few have actually implemented them successfully. This relates to the fact that the size of the neural networks can be potentially so large as to prevent the problem solution from being commercialized in the real world (Leerink et al., 1995). Furthermore, the large network size can slow down the training speed and its convergence. For these reasons, selecting the optimum network structure is very important. A neural network of size below the optimum will usually fail to approximate the underlying function. On the other hand, a network with size above the optimum will have a large number of weights and tend to memorize the training data, this can lead to over-fitting of the problem, which can result in poor generalization (Lawrence and Giles, 2000).

The highly popularized Multilayer Perceptron (MLP) has been successfully applied in a broad class of financial markets prediction tasks (Hellstrom and Holmstrom, 1998; Dunis and Williams, 2002; Yao and Tan, 2002; Shachmurove and Witkowska, 2002). However, MLP adopts computationally intensive training algorithms such as the error backpropagation and can get trapped in local minima (Lawrence and Giles, 2000). In addition, the network has problems when dealing with large amounts of training data, and demonstrates poor interpolation properties, when using reduced training sets.

In many cases, the slow speed of neural networks is due to its large size, which can slow down the feedforward process. In feedforward process, each weight in a network results in multiplication with the nodes, and each node results in the evaluation of the transfer function. Thus, it is important to consider the number of nodes and weights employed in the network since they require a large space for mathematical implementation. Since the MLP network has multilayered structure, the network requires excessive training time for learning. Furthermore, the number of weights and the training time increases as the number of layers and the nodes in a layer increases (Patra and Pal, 1995; Chen and Leung, 2004).

Higher Order Neural Networks (HONNs) which have a single layer of trainable weights can help speeding up the training process. HONNs are a type of feedforward neural networks, which have certain advantages over MLP. They are simple in their architecture and this potentially reduces the number of required training parameters. As a result, they can learn faster, since each iteration of the training procedure takes less time (Cass and Radl, 1996).

HONNs have applications in wide range areas of human interests. They are not just scientific curiosities as they have already been applied in many and various real commercial applications such as pattern recognition (Artyomov and Pecht, 2004; Voutriaridis et al, 2003; Kaita et al, 2002; Shin et al, 1992), function approximation (Voutriaridis, 2003; Shin and Ghosh, 1995; Shin and Ghosh, 1992; Ghosh and Shin, 1992), process optimization (Cass and Radl, 1996),system identification (Mirea and Marcu, 2002), signal processing (Patra and Pal, 1995), image processing(Hussain and Liatsis, 2002), classification (Shin and Ghosh, 1995; Ghosh and Shin, 1992), time series prediction (Tawfik and Liatsis, 1997), and intelligent control (Karnavas and Papadopoulos, 2004; Pau and Phillips, 1995). Most applications are related to pattern recognition and function approximation problems but other applications are steadily growing. Nevertheless, literatures on the use of HONNs for financial time series prediction are limited. The questions of how good HONNs on pattern recognition and function approximation have been widely researched, but the corresponding question of how good they are on financial time series has not been adequately addressed.

1.4 Aims and Research Challenge

The aim of this research study is to make an investigation and analysis on HONNs models, with an application to financial time series prediction. More specifically, this research work investigates the theory of HONNs, their architectures and their learning algorithms. Furthermore, this research work seeks to find a network architecture which maintains a good performance, while at the same time reducing all the problems associated with network complexity. Besides, the research study also points to observe the use of HONNs as financial time series predictor with parsimony structure that can maintain good generalization capability. This research work emphasizes at designing a network architecture which can simplify the training and significantly can reduce the convergence time. Hence, the major challenge ahead is the development of a valid, precise, and reliable network predictor which can be tested against actual trading performance and gain profits based on the prediction results.

1.5 Objectives and Contribution of the Thesis

In order to investigate the research aims, a few specific objectives are set as follows:

- To design, implement and simulate HONNs models; such as the FLNN, PSNN, and RPNN for the prediction of the future trend of financial time series.
- To construct a novel Dynamic Ridge Polynomial Neural Network, which comprises of a feedback connection in addition to the feedforward Ridge Polynomial Neural Network.
- To address the problem of stability in the proposed Dynamic Ridge Polynomial Neural Network and finding a mathematical solution for its stability.
- To compare the out-of-sample performance of various HONNs models as well as the MLP.
- To evaluate the performance of all network models with financial metrics and statistical metrics.

1.6 Scope and Limitations

The potential combinations of neural networks type and financial time series prediction are virtually limitless. In order to place boundaries around the vast topic of time series forecasting using neural networks, this research work is limited to the analysis, construction, implementation, and testing of the Multilayer Perceptron, Functional Link Neural Network, Pi-Sigma Neural Network, Ridge Polynomial Neural Network, and Dynamic Ridge Polynomial Neural Network. The construction of the networks is based on the Standard Incremental Backpropagation (for MLP, FLNN and PSNN), and Constructive Learning Algorithm (for RPNN and DRPNN). All the networks will be tested on ten financial time series signals. They are the IBM common stock closing price, the Standard & Poor 500 stock index futures, the United States 10-year government bond, the United States 30-year government bond, the UK pound to EURO exchange rate, the UK pound to US dollar exchange rate, the US dollar to EURO exchange rate, the Japanese yen to EURO exchange rate, the Japanese Yen to US dollar exchange rate, and the Japanese Yen to UK pound exchange rate. In this research work, the performance of the network is evaluated using five financial criteria (annualized return, maximum drawdown, annualized volatility, sharpe ratio, and transaction cost) and four statistical criteria (normalized mean squared error, mean squared error, correct directional change, and signal to noise ratio).

Detailed Gantt chart for the research framework is presented in Appendix 1.

1.7 Thesis Structure

The remaining part of this thesis is broken up into the following chapters. Chapter 2 is concerned with the literature review on neural networks and their types. This includes the architectures of feedforward neural networks and recurrent neural networks. Chapter 3 describes various types of Higher Order Neural Networks, their

learning algorithms and applications. This covers the Functional Link Neural Network, the Pi-Sigma Neural Network, and the Ridge polynomial Neural Network. Chapter 4 introduces the proposed Dynamic Ridge Polynomial Neural Network; presented as an extension of the ordinary feedforward Ridge Polynomial Neural Network. Subsequently, the stability and convergence of the network is shown. Chapter 5 reviews the fundamentals of financial time series prediction, addressing their difficulties, and their practical applications using neural networks and traditional forecasting approaches. Chapter 6 assesses the extensive modeling and design methodology in all network models, as well as the generation of input-output pattern, the specification of parameters, and performance measures. Chapter 7 presents the simulation results for the prediction of all data signals using all neural network models. Analysis on the results is presented follows with a statistical and graphical review of the information acquired. Some issues raised by the results are discussed. Chapter 8 is dedicated for the final conclusions, contribution of research, and further works.

1.8 Chapter Summary

The challenge in financial time series forecasting is to discover the network model that would provide the best forecast and yield the best profit. However, there cannot be a universal model that can produce good prediction for all data signals, and indeed, there is probably no single best forecasting method for all situations. The design of neural network model does require knowledge, such as a strategy to acquire the necessary data to train the network, the selection of learning rules, data pre-processing methods and mainly how to connect the neurons within the network. With proficiency design, careful selection of learning parameters and pre-processing of the data, it is anticipated that HONNs used in this research work might be able to produce superior performance in the prediction of financial time series.

CHAPTER 2: NEURAL NETWORKS

2.1 Introduction

Neural networks provide a general class of nonlinear models which have been successfully applied in many engineering and scientific problems. These includes real world problems such as time series prediction (Dunis and Williams, 2002; Chen and Leung, 2004; Ho et al., 2002; Plummer, 2000; Leung et al., 2002), image processing (Hussain and Liatsis, 2002), speech/character/pattern recognition (Pao, 1989; Kaita et al., 2002), system identification (Mirea and Marcu, 2002), medical image analysis (Shieh, et al., 2004), system optimization (Yu and Morales, 2005), function approximation (Ghosh and Shin, 1992; Shin and Ghosh, 1991) and more. Their numerous application domains fall into categories: for example regression and generalization, classifications, association, clustering, pattern completion, and optimization.

The idea of Artificial Neural Networks (ANNs) is to model a neuron by building interconnected networks, and devise learning algorithms to work out the ANNs. Often the term '*Neural networks*' is used as a broad sense which group together different families of algorithms and methods. A formal definition of ANNs according to Haykin (1999) is:

"A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects: 1) knowledge is required by the network through a learning process, 2) Interneuron connection strengths known as synaptic weights are used to store knowledge."

Biological wise, the term '*Neural networks*' is used to describe models of computation in single neurons or whole areas of brain. Neural networks which posses learning abilities have attracted much attention due to the way they use data to learn

patterns and underlying relationship instead of totally rely on people to specify them. According to Zaknich (2003), neural networks can often provide suitable solutions for problems that generally are characterised by nonlinearities, high dimensionality, noisy, complex, imprecise, imperfect and/or error prone sensor data, poorly understood by physical and statistical models, and lack of clearly stated mathematical solution or algorithm.

2.2 From Biological to Artificial Neuron

Neural networks are information processing paradigms that are inspired by the way in which human brain processes information. Researchers and even computer scientists are excited about the enormous power of the human brain. The capability of the brain to solve complex nontrivial problems is even impossible to solve using the newest computer technology. Recognition of the brain's impressive power has lead to interest in the development of ANNs (Zaknich, 2003).

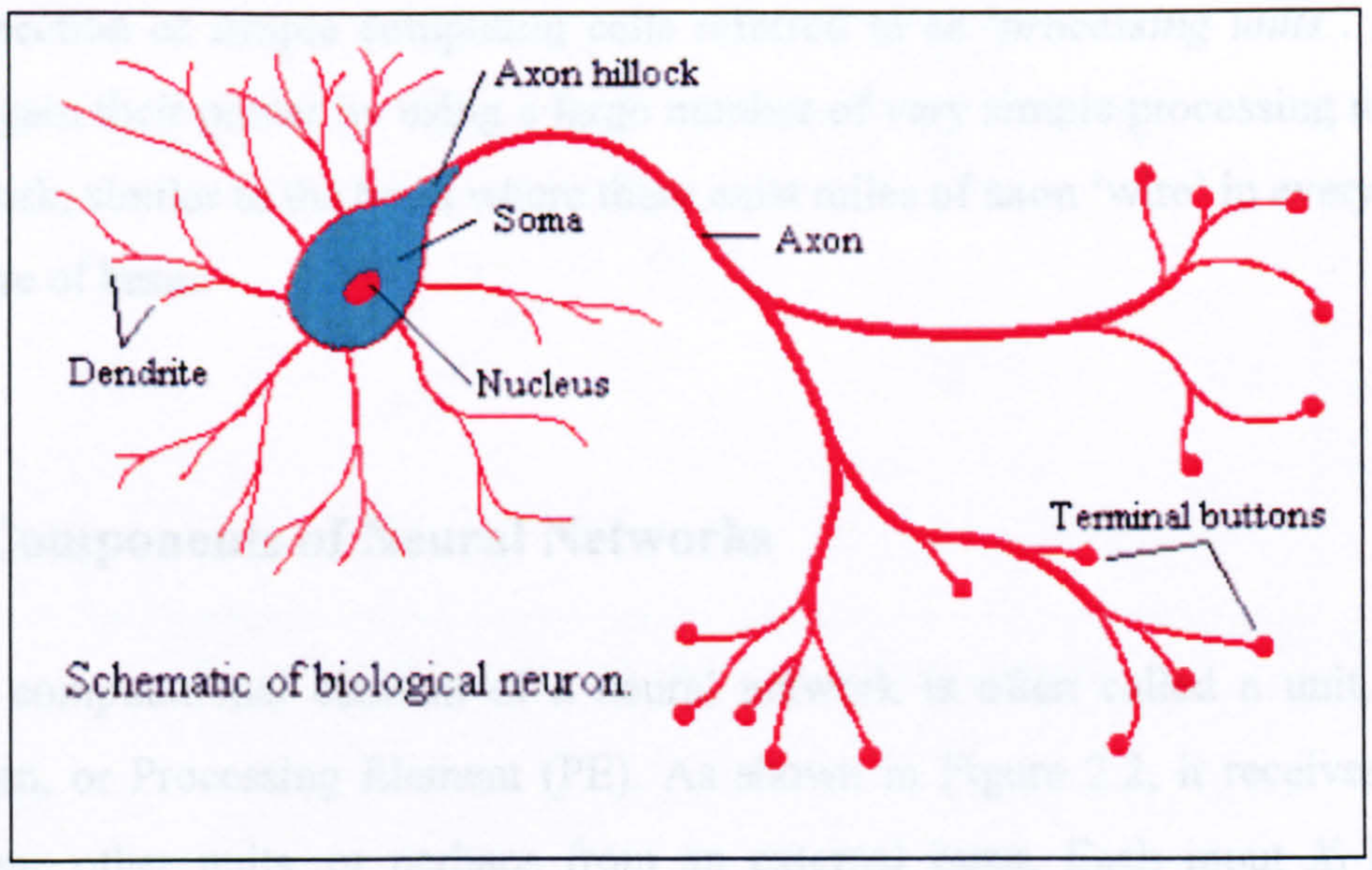


Figure 2.1: Basic features of biological neurons (Fraser, 1998)

Neural networks are based on a rather simple model of brain neuron as shown in Figure 2.1. Most neurons have three parts: a dendrite which acts as receptive zones and collects inputs from other neurons, or from external stimulus; a soma (cell body)

which performs an important nonlinear processing step; and finally an axon, a cable-like wire along which the output signal is transmitted to other neurons further down the processing chain (University of North Carolina at Charlotte, 2002). The connection site between two neurons is called a synapse. Synapses are elementary structural and functional units that mediate the interconnections between neurons. The signal of most real neurons is chemical and it consists of spikes, short pulses of electrical activity. In Artificial Neural Networks, these spikes are replaced by a continuous variable X_j which we may think of as a temporally average pulse. The majority of neurons encode their outputs as a series of brief voltage pulses. A biological neuron may have as many as 10,000 different inputs, and may send its output to many other neurons (up to 200,000).

The same mechanism and function exist in ANNs. They have many very simple processors, each possibly having a local memory, which are organized in layers and are connected by weighted links. It is an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neurons. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as '*processing units*'. ANNs systems gain their power by using a large number of very simple processing units in the network, similar to the brain where there exist miles of axon 'wire' in every cubic centimetre of brain.

2.3 Components of Neural Networks

A basic computational element of a neural network is often called a unit, node, perceptron, or Processing Element (PE). As shown in Figure 2.2, it receives input from some other units, or perhaps from an external input. Each input X_i has an associated weight W_{ij}, which have different synaptic strength. These weighted inputs are summed to give the net input, S. Most units in neural networks transform their net input by using a scalar-to-scalar function called an '*activation function*', yielding a value called the '*unit's activation*' or neuron's output. This neuron's output, Y, is produced at the output layer. Hidden and output units usually use a '*bias*' or

'*threshold*' term in computing the net input to the unit. A bias term can be treated as a connection weight from a special unit with a constant, nonzero activation value. The single bias unit is connected to every hidden or output unit that needs a bias term. Hence the bias terms can be learned just like other weights.

Neural networks behave, react, self organize, learn, and generalize rather than just execute programs (Schwaerzel, 1996). Neural networks derive their computing power through their massive parallel distributed structure, and their ability to learn and therefore generalize (Haykin, 1999). Generalization refers to the fact that neural networks can produce reasonable outputs for inputs not encountered during learning. Given a training set of data, neural networks can learn the data with a learning algorithm; the most common learning algorithm is the backpropagation. Through the learning algorithm, neural networks form a mapping between inputs and the desired outputs from the training sets by altering weighted connections within the networks.

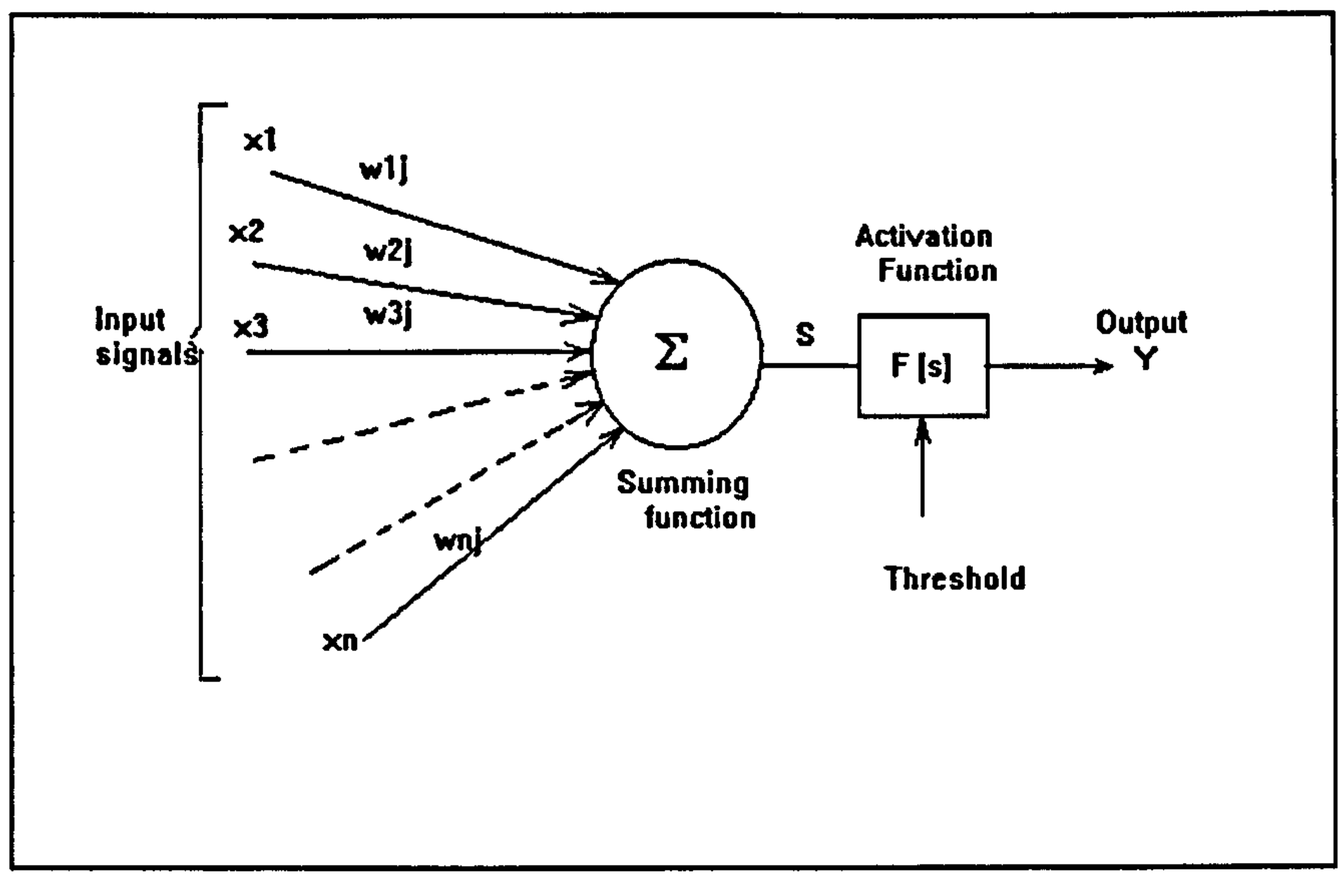


Figure 2.2: Network with single perceptron or node

2.4 Different Structures of Neural Networks

In a great variety of neural networks, the interconnection architecture can be very different for different types of networks (Zaknich, 2003). The neurons in ANNs can be interconnected in many different possible topological ways. These topologies

include single-layer and multi-layer networks. The layer where the input features are presented is referred to as the input layer and the output layer is where the network outputs are formed. A single-layer network actually has both the input and the output layers, whereas a multi-layer network can also have one or more hidden layers in between. The hidden layers are so called because their inputs and outputs are only used for internal connections. The number of inputs to the network is constrained by the problem to be solved, and the number of neurons in the output layer is constrained by the number of outputs required by a particular problem.

Different types of neural networks have different strength and abilities particular to their application and can be related to their structure and learning method. It is widely acknowledged that there is no single method, statistical or neural network that gives the best result for all kinds of problems. Generally, there are two types of neural networks architectures; feedforward network and recurrent network (Sarle, 2002). In feedforward network, the signal can only travel in one direction, whereas in recurrent network, the signal can travel in both directions by introducing loops or cycle in the network itself.

2.4.1 Feedforward Neural Networks

Often neural networks are arranged in layers such that the connections are only between consecutive layers, all in the same direction. Such neural networks are called feedforward neural networks. These networks can have any number of layers, units per layer, network inputs, and network outputs. The input signals propagate through the network in a forward direction, on a layer-by-layer basis, hence the term feedforward. The output is only a function of the current input, not of the past or future inputs or outputs, therefore the node equations are memoryless.

2.4.1.1 Multilayer Perceptron (MLP)

MultiLayer Perceptron (MLP) is a feedforward network which is formed by a collection of summing units that are connected by their associated weights. Due to its capability of learning a rich variety of nonlinear decision surfaces, MLP has been successfully tested in many applications; among those are financial time series prediction (Dunis and Williams, 2002; Plummer, 2000; Yao and Tan, 2000), signal processing (Richmond, 2002), and function approximation (Lawrence and Giles, 2000).

The network has a hierarchical structure of several perceptrons, and has the ability to overcome the shortcomings of single-layer networks (Nikolaev, 2006). It has one or more hidden layers in between the input and output layers, which transmit the data from the input nodes to the output nodes. The function of the hidden nodes is to intervene between the external inputs and the network output in some useful manner. Figure 2.3 illustrates the layout of MLP with single hidden layer. The network figure is said to be fully connected in the sense that every node in each layer is connected to every other node in the adjacent forward layer.

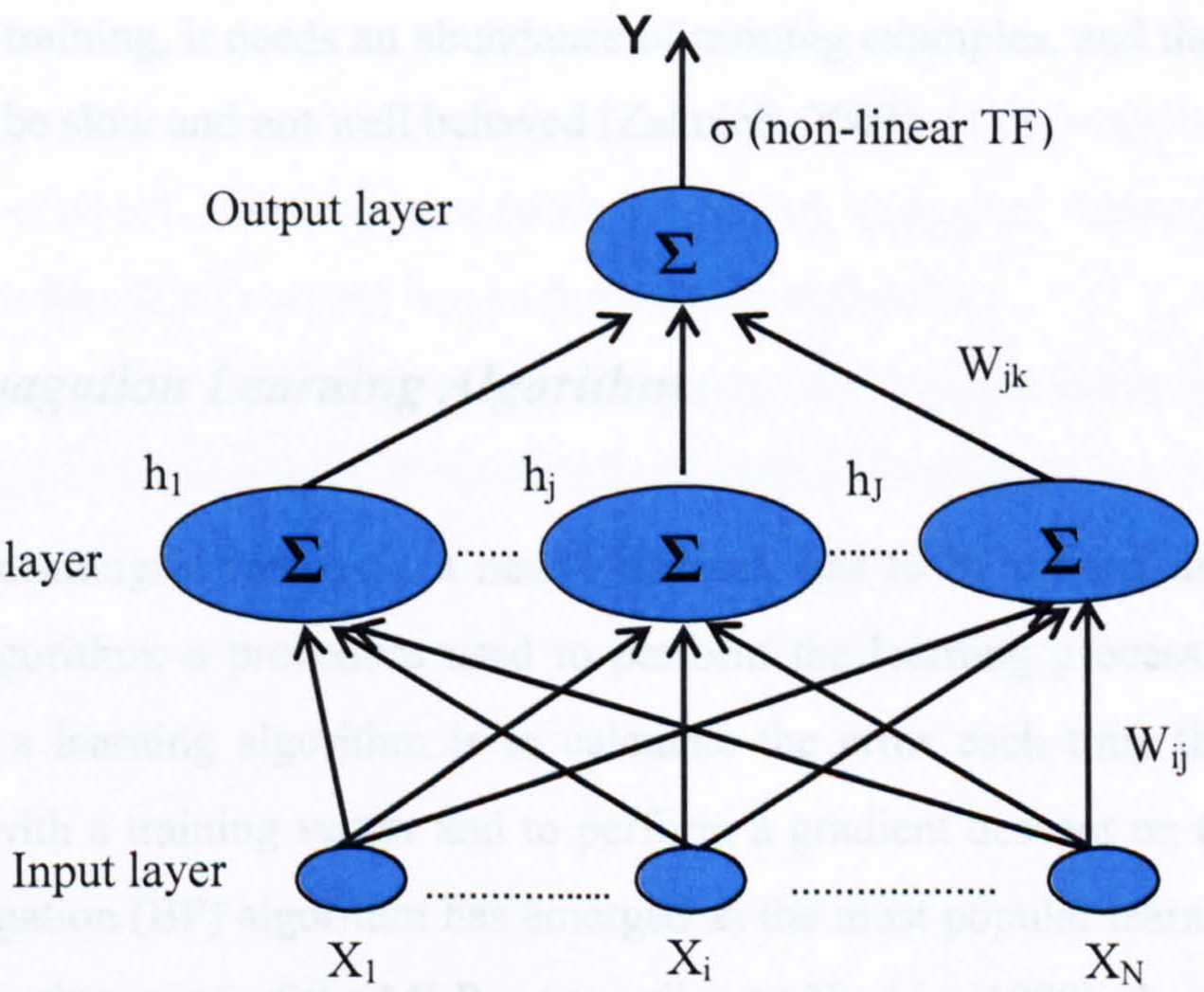


Figure 2.3: MultiLayer Perceptron
Bias nodes are not shown here for reason of simplicity.

MLP computes the network output according to following equation:

$$Y = \sigma(\sum_{j=1}^{J} W_{jk}\sigma(\sum_{i=1}^{N} W_{ij}X_i + W_{oj}) + W_{ok}) \qquad (2.1)$$

where x_i denotes the input value, W_{ij} is the weights from the input layer to the hidden layer, W_{jk} is the weights from the hidden layer to the output layer, W_{oj} is a bias for hidden node, σ is a sigmoid transfer function, and Y is the network output. MLP has a highly connected topology since every input is connected to all nodes in the first hidden layer, and every unit in the hidden layers is connected to all nodes in the next layer, and so on. The input nodes pass values to the first hidden layer's nodes. The forward propagation is continued to the second hidden layer and so on until the output of the network is produced at the output layer.

MLP can approximate reasonable functions to any desired degree of accuracy using only one hidden layer, provided that sufficiently many hidden nodes are available, and having sigmoid function as the nonlinear activation function (Cybenko, 1989; Hornik et al, 1989). Due to their multiple layer structure, they utilised computationally expensive training algorithms and thus can get stuck in local minima. One of the network's disadvantages is that it can only be used with supervised training, it needs an abundance of training examples, and the training can sometimes be slow and not well behaved (Zaknich, 2003).

Backpropagation Learning Algorithm

To make meaningful forecasts, a neural network has to be trained using a certain learning algorithm; a procedure used to perform the learning process. The idea of employing a learning algorithm is to calculate the error each time the network is presented with a training vector and to perform a gradient descent on the error. The Backpropagation (BP) algorithm has emerged as the most popular learning algorithm for supervised training of the MLPs. According to Haykin (1999), the algorithm has two distinct properties; it is simple to compute locally, and it performs stochastic

(updating weights by pattern learning) gradient descent in weight space. It however has some downside properties:

1. As the algorithm uses an 'instantaneous estimate' for the gradient of the error surface in weight space, it is therefore stochastic in nature and has a tendency to zigzag its way about the true direction to a minimum in the error surface.
2. As a result, it converges fairly slow, which in turn make it computationally agonizing.
3. The algorithm runs a risk of being trapped in local minima in which every small change in the synaptic weights affects the cost function. It is unfavourable to have the learning process cease at a local minimum instead of global minimum.

Backpropagation algorithm is a supervised learning algorithm based on a suitable error or cost function, with values determined by the actual and desired outputs of the networks, which is to be minimized via a gradient descent method. The principle idea of this algorithm is to compute the influence of each weight in the network by performing an iterative training process. The aim is to minimize the error by performing simple gradient descent where the weight is adjusted in the steepest descent direction (negative of the gradient). This is the direction in which the error rapidly decreased.

The learning of the networks is performed in such a way that the weights are adjusted after the presentation of each or a batch of training examples. During the iterative process, two sets of signals are passed through the networks:

- Function signals: the input examples propagated through the hidden units and processed by their activation functions and emerge as output.
- Error signals: the errors at the output nodes are propagated backward layer-by-layer through the networks so that each node returns its error back to the nodes in the previous layer.

The weights are adjusted in accordance to the Delta Rule. It suggests that the actual network output is subtracted from the desired output in the example. The weights are adjusted so as to make the network output much closer to the desired output. The error function to be minimized is:

$$E = \frac{1}{2}\sum_{k}(t_k - y_k)^2 \tag{2.2}$$

where t_k is the desired output and y_k is the network output. Each component of the gradient provides the slope of the error function with respect to that weight:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial W_0}, \frac{\partial E}{\partial W_1}, \ldots, \frac{\partial E}{\partial W_n} \tag{2.3}$$

The partial derivative of the error function with respect to the weights and biases in the backpropagation algorithm is determined as follows:

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial S_i}\frac{\partial S_i}{\partial net_i}\frac{\partial net_i}{\partial W_{ij}} \tag{2.4}$$

where W_{ij} is the weight from neuron j to neuron i, S_i is the neuron output, and net_i is the weighted sum of the inputs of neuron i. Given the gradient, each weight is adjusted by the negative of the gradient to reduce the error. The value of the derivative is then used to minimize the error function by performing a gradient descent as below:

$$W_{ij}(t+1) = W_{ij}(t) - \varepsilon \frac{\partial E}{\partial W_{ij}}(t) \tag{2.5}$$

The learning rate, ε, is used to control the learning step, and has a very important effect on convergence time. A very large learning rate can lead to oscillation in the weight space and could end up with reaching only local minima instead of global optima. Meanwhile setting a small value of the learning rate can lead to a slow training since many weight steps are required. To prevent the above problems, momentum term usually added to scale the influence of the previous step/derivative to the current and to make the learning process more stable:

$$\Delta W_{ij}(t) = -\varepsilon \frac{\partial E}{\partial W_{ij}}(t) + \mu \Delta w(t-1) \tag{2.6}$$

where μ is the momentum term. Another reason for introducing the momentum term is to avoid oscillation when using high learning rate. For each iteration, the

change in the weight keeps a little bit of the direction of the previous weight change. Thus the weights behave as if they had some inertia or 'momentum'. The use of momentum in the BP algorithm can be helpful in speeding the convergence and avoiding local minima (Nikolaev, 2006).

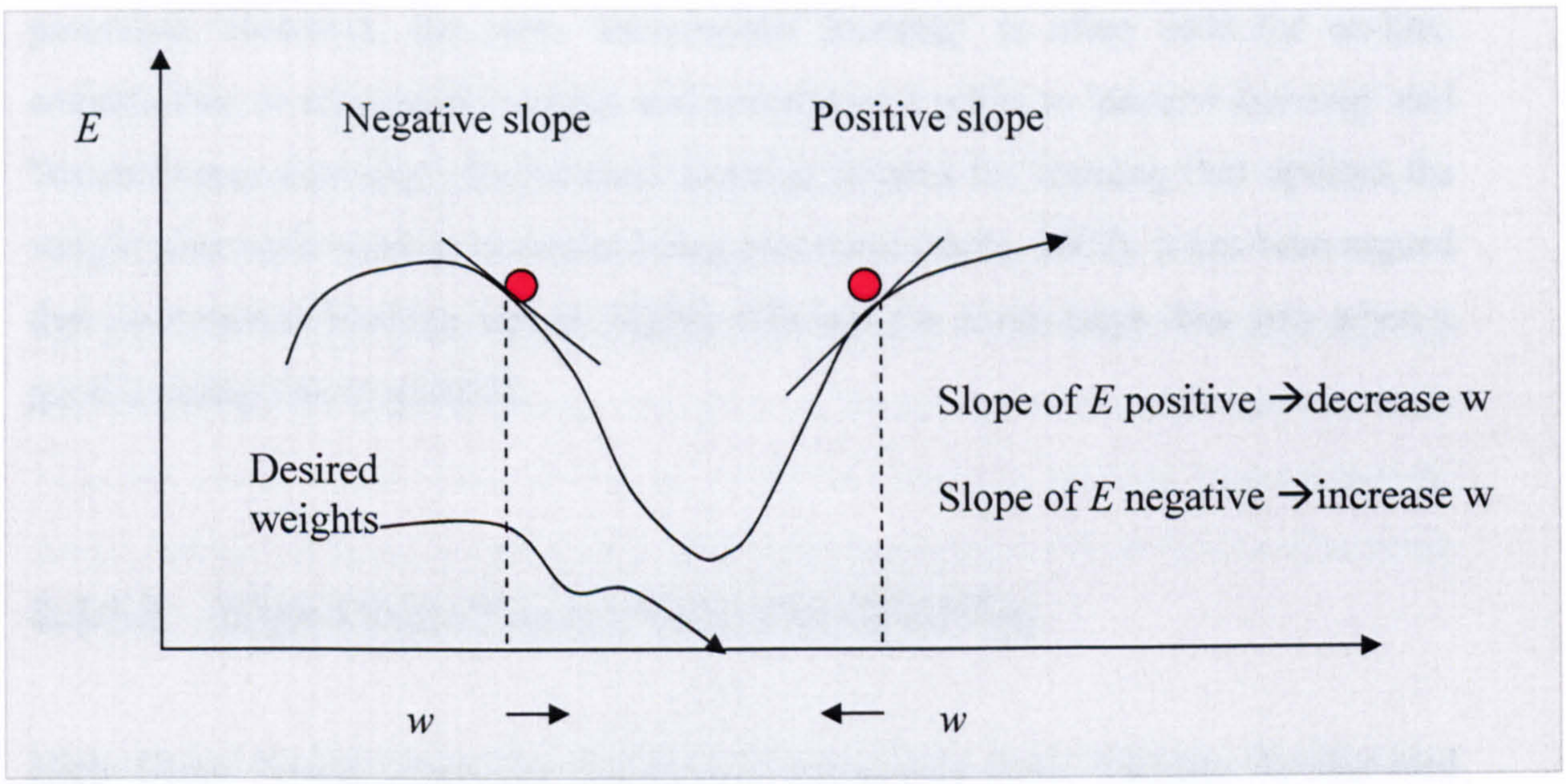


Figure 2.4: Basic principle of gradient descent

Figure 2.4 shows the behaviour of error E with respect to one weight w. In order to decrease the value of the error function E, the Backpropagation algorithm does gradient descent in the reverse direction of the error gradient (slope). If the gradient of E is negative, the value of w must be increased to move forward towards the minimum. If E is positive, the value of w must be decreased to move backward to the minimum. There will be a gradient of slope for each weight. By repeating this process, E is moved 'downhill'until a minimum is reached, where at this time, no further progress is possible.

There are two methods to update the weights of the network; batch gradient descent and incremental gradient descent. In batch gradient descent, the true gradient is usually the sum of the gradients caused by each individual training example. Therefore, batch gradient descent requires one sweep through the training set before any parameters can be changed. In incremental gradient descent, the true gradient is

approximated by the gradient of the cost function, which is evaluated on a single training example.

The term '*batch learning*' is used quite consistently in neural networks literature, which means that the weights are updated after the whole training examples being processed. However, the term '*incremental learning*' is often used for on-line, constructive, or sequential learning, and sometimes it refers to '*pattern learning*' and '*instantaneous learning*'. Incremental learning is used for learning that updates the weight after each training examples being processed (Sarle, 2002). It has been argued that incremental learning can be highly efficient for some large data sets when a good learning rate is selected.

2.4.1.2 High Order Neural Networks (HONNs)

High Order Neural Networks distinguish themselves from ordinary feedforward networks by the presence of high order terms in the network. In a great variety of neural networks models, neural inputs are combined using the summing operation. HONNs contain summing unit and product units that multiply their inputs. These high order terms or product units can increase the information capacity of higher order network in comparison to standard neural networks with summation units only. The larger capacity means that the same function or problem can be solved using higher order network that has fewer units. HONNs also make use of non-linear interactions between the inputs. The networks therefore expand the input space into another space where linear separability is possible (Pao, 1989).

Although most neural networks models share a common goal in performing functional mapping, different network architectures may vary significantly in their ability to handle different types of problems. For some tasks, higher order combinations of some of the inputs or activations may be appropriate to help form good representation for solving the problems. HONNs are needed because ordinary feedforward network like MLP cannot elude the problem of slow learning, especially when involving highly complex nonlinear problems (Chen and Leung, 2004). The

representational power of high order terms can help solving complex problems with construction of significantly smaller network whilst maintaining the fast learning (Leerink et al, 1995).

A comprehensive discussion on HONNs will be included in Chapter 3.

2.4.2 Recurrent Neural Networks (RNNs)

Feedforward Neural Networks have been successfully used to solve problems that require the computation of a static function; i.e. function whose output depends only upon the current input, and not on any previous inputs. In the real world however, many problems cannot be solved by using static functions because the function being computed changes with each input received.

Feedforward networks however have no way of influencing the processing of future inputs. This situation can be rectified by the introduction of feedback connections in the network. Network activation produced by past inputs can be cycled back and can affect the processing of future inputs. This allows the network to have knowledge of the past behaviour. To enable the architecture to learn a representation of time in data, Recurrent Neural Network (RNN) is used.

Recurrent Neural Networks have been proposed to overcome these deficiencies in ordinary feedforward networks. A neural network is said to be recurrent if it possesses at least one feedback connection. RNNs are neural networks where the connections between the units form a directed cycle or looping. They must be approached differently than feedforward networks, both when analysing their behaviour and during training. RNNs behave chaotically where dynamical systems theory is used to model and analyse them. As stated by Kuan and Liu (1994), RNNs have a richer dynamic structure and they are similar to nonlinear time series models with moving average terms; the nonlinear Auto Regressive Moving Average models (ARMA).

RNNs have some notion on how the past inputs can affect the processing of current input, as well as a way of storing the past inputs. In other words, they have a memory of the past input and a way to use that memory to process the current input. Recurrence is achieved by feeding the network with a delayed version of the past observations, commonly referred to as a delay vector or tapped delay line. These recurrences enable the network to find out an appropriate internal state representation which allows the time series behaviour to be captured. With those internal dynamics, RNNs can learn sequences as time evolves and can response to the same input pattern differently at different times, depending on the previous input patterns as well.

According to Kim (1998), there are multiple methods to present temporal information in the neural networks. These include: (1) creating a spatial representation of temporal pattern, (2) putting time delays into the neurons or their connections, (3) employing recurrent connections, (4) using neurons with activations summing inputs over time, and (5) using combination of the above. Above all, Kim in his work (1998) suggested that employing time delayed recurrences in the layered network is more efficient for temporal correlations and prediction than putting multiple time delays into the neurons or their connections.

The RNN can be fully or partially connected. In a fully connected RNN all the units are connected recurrently, whereas in partially RNN the recurrent connections are omitted partially. Fully connected RNN uses unconstrained fully interconnected architectures and learning algorithms that can deal with time-varying input and/or output in non-trivial ways (Omlin and Giles, 1996). Fully RNN, as shown in Figure 2.5, has feedforward and feedback connections in any order, all of which are trainable. The network can take on any arbitrary topology as any node in the network may be linked with any other nodes including the node itself. The only requirement to be made is that the network should has clearly defined input and output nodes. Omlin and Giles in their work (1996) used partially and fully RNNs to classify strings with arbitrary length. Meanwhile, research done by Moody et al (1998), used fully RNN model to perform a spatial delayed matching to sample task.

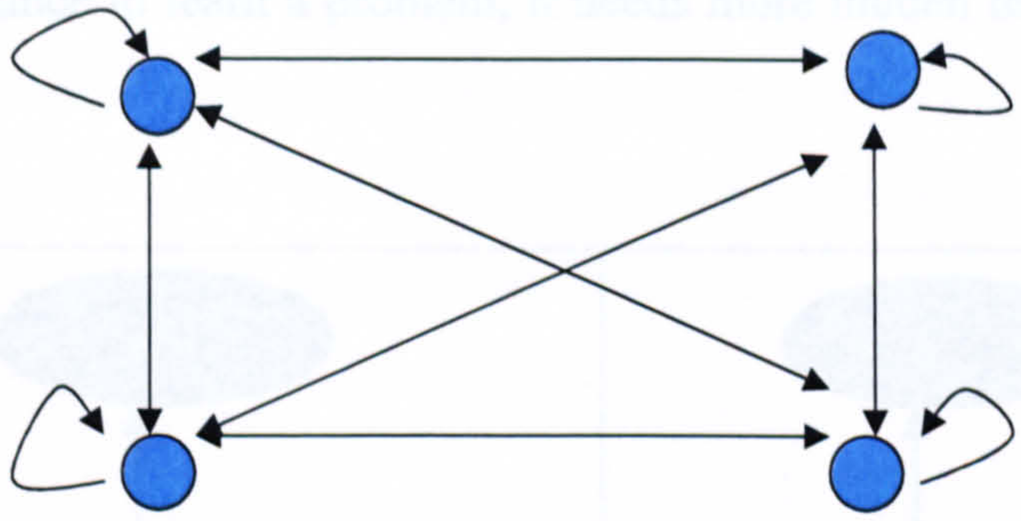


Figure 2.5: Fully-Connected RNN

Partially RNNs are a type of feedforward networks with the incorporation of a unit called 'context unit', which stores the output from the hidden or output layers. The connections in the partially RNNs are mainly feedforward but include a carefully chosen set of feedback connections. This network has all its feedforward connection trainable, whereas the feedback connections are fixed. The recurrence in the partially RNNs allows the networks to remember cues from the recent past but does not appreciably complicate the structure and training of the whole network. There are several different models of partially RNNs. Examples of partially RNNs include the Jordan network (Jordan, 1986), and the Elman network (Elman, 1990). There are two kinds of local feedbacks in partially RNNs; the activation feedback (mainly applied in Elman network), and the output feedback (used in Jordan network). Figures 2.6 (a) and (b) shows the architectural of the Elman and the Jordan networks, respectively.

The Elman network is a two-layer network with feedback from the hidden layer to the input layer, as depicted in Figure 2.6 (a). This recurrent connection allows the Elman network to both detect and generate time-varying patterns. The input layer is divided into two parts; actual input units, and context units. The context units are connected to the forward direction with weights fixed to unity and they are not trainable. The presence of this simple loop implies that the activations of the hidden units at time t can influence the activations of the hidden units at time $t+1$. The recurrent connections allow the network's hidden units to see its own previous output, so that the subsequent behaviour can be shaped by previous responses. These recurrent connections give the network memory. In order for the Elman network to

have the best chance to learn a problem, it needs more hidden neurons in its hidden layer.

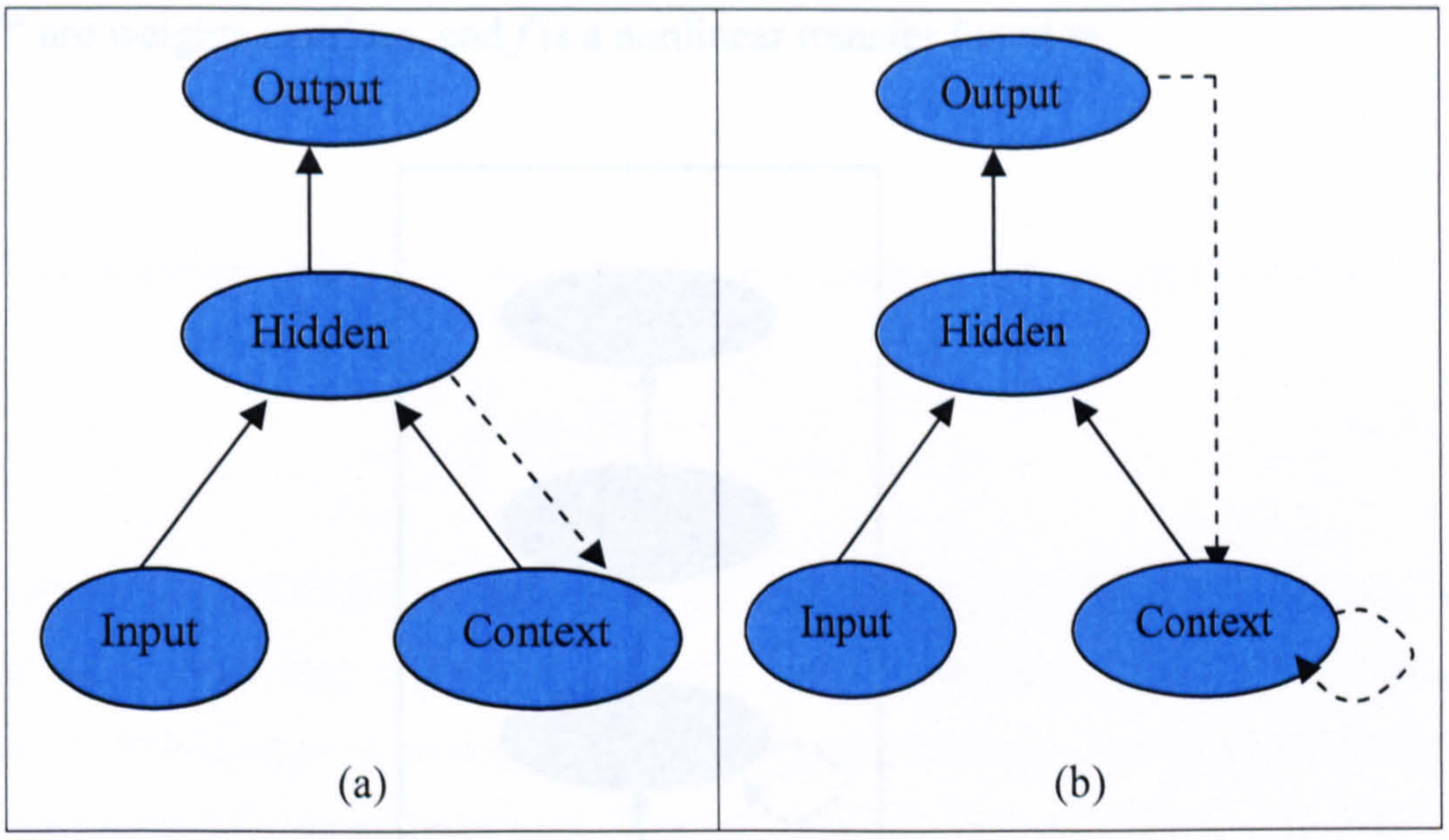


Figure 2.6: (a) Elman and (b) Jordan networks

In the case of Jordan network, the architecture is realized by adding recurrent links from the network's output layer to a set of context units which form a context layer. Additionally, the context units are connected with each other and with themselves. This allows them to calculate their next state as a function of the current net output, their current state, and the current state of the other state units. The self connections in the context layer give the context units some individual memories or inertias. Hence, Jordan network can be trained to recognize and distinguishes different input sequences. Based on Figures 2.6(a) and 2.6(b), the following Equation (2.7) and Equation (2.8) hold for Elman and Jordan network respectively:

$$Y(t) = W^{yx} X(t)$$
$$X(t) = f\left(W^{xc} X^{c}(t) + W^{xu} U(t)\right) \tag{2.7}$$
$$X^{c}(t) = X(t-1)$$

$$Y(t) = W^{yx} X(t)$$
$$X(t) = f\left(W^{xc} X^{c}(t) + W^{xu} U(t)\right) \tag{2.8}$$
$$X^{c}(t) = Y(t-1) + \alpha(X^{c}(t-1)$$

where $Y(t)$ represents the network output at time t, $X(t)$ is the activations of hidden units at time t, $X^c(t)$ is the output of the context units at time t, $U(t)$ is the external input to the network at time t, α is the feedback gain of the self connection, W^{yx}, W^{xc}, and W^{xu} are weights matrices, and f is a nonlinear transfer function.

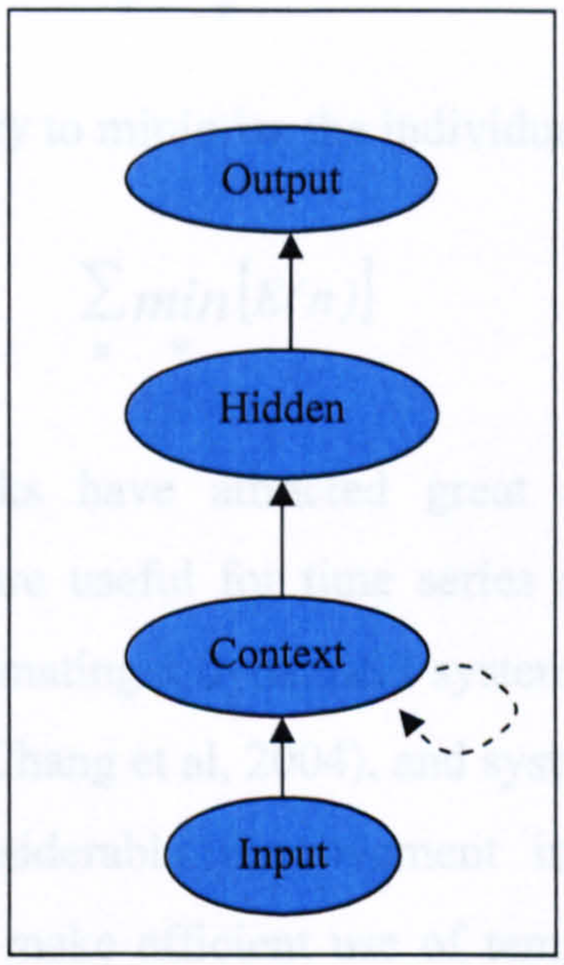


Figure 2.7: A four-layered partially RNN

Variations of simple RNNs can be defined by adding additional hidden layers and by changing the connectivity of the context layers. Another different type of partially recurrent network was developed by (Mozer, 1989) and is illustrated in Figure 2.7, has been tested for pattern recognition. The feedback connection shown in this figure originated from context unit to itself. The network's input layer consists of a small temporal buffer holding several elements of the input sequence. The connectivity in the context layer is restricted to one-to-one recurrent connection and the integration over time in the context layer is linear. The presence of the feedback loops in Figures 2.5, 2.6 and 2.7 have a profound impact on the learning capability of the networks and on its performance.

A number of training algorithms for training RNNs have been proposed. Some of them are the Dynamic Backpropagation (Kuan, 1989), Real Time Recurrent Learning (Williams and Zipser, 1989), and Backpropagation Through Time (Patterson, 1996). The fundamental difference between the Backpropagation

Through Time (BPTT) and the Real Time Recurrent Learning (RTRL) algorithms can be expressed in the following way. While the BPTT algorithm will try to minimise the error over a sequence:

$$\min_{w} \sum_{n} E(n) \tag{2.7}$$

The RTRL algorithm will try to minimise the individual error terms of a sequence:

$$\sum_{n} \min_{w} [E(n)] \tag{2.8}$$

Recurrent Neural Networks have attracted great attention from the scientific community because they are useful for time series forecasting (Zhang and Chan, 2000; Steil, 2006), approximating a dynamical system (Kimura and Nakano, 2000), forecasting a stream flow (Chang et al, 2004), and system control (Reyes et al, 2000). They have shown a considerable improvement in performance over ordinary feedforward networks and make efficient use of temporal information in the input sequence, both for classification (Jordan, 1986; Husken and Stagge, 2003) as well as for prediction (Kuan and Liu, 1994; Ho et al, 2002; Kim, 1998). As argued in (Gilde, 1996), RNNs are more suitable for the analysis and prediction of time series since they can represent time dependencies in the data better than feedforward networks. They are believed to be able to represent complex dynamic system better than feedforward networks, and they could increase the precision of predictions and improve the possibilities to analyse time series.

2.5 Chapter Summary

Neural Networks have been shown to have a high parallel computational ability. They have the ability to learn and find optimal solution based on actual and desired output. The capability of neural networks to implement solutions without complete knowledge of the algorithms or data transformations makes them suitable to solve many real world problems. In practical applications, Neural Network is expected to be an interconnected network of many (possibly thousand) simple processing units.

The effectiveness of the network is expected to come about because of the complexity of the interconnections rather than through any particular clever behaviour of the individual neurons.

CHAPTER 3: HIGHER ORDER NEURAL NETWORKS

3.1 Introduction

This chapter discusses various types of Higher Order Neural Networks (HONNs), their learning algorithms, and their applications. Each type of networks has its own strengths and capabilities in input-output mappings, on various kinds of problems ranging from signal prediction, pattern recognition, time series forecasting, data classification, and etc. Three Higher Order Neural Networks models will be investigated in this research work; the Functional Link Neural Network (FLNN), the Pi-Sigma Neural Network (PSNN), and the Ridge polynomial Neural Network (RPNN).

3.2 The Properties of HONNs.

Neurons in an ordinary feedforward network is just a first order neuron, also called a 'linear neuron' since it only uses a linear sum of its inputs for decision. This linearity provides a hyperplane for decision that limits the capability of the neuron to solve only linear discriminant problems (Guler and Sahin, 1994).

It is well known that using single layer feedforward neural networks with first-order units can only provide linearly separable mappings (Minsky and Papert, 1969). One possibility to drop this limitation is by using multilayer networks with hidden units which can combine the outputs of previous units and give rise to nonlinear mappings (Hornik et al., 1989). The other way to overcome the restriction to linear maps is to introduce higher order units to model nonlinear dependences (Giles and Maxwell,1987; Giles et al., 1998).

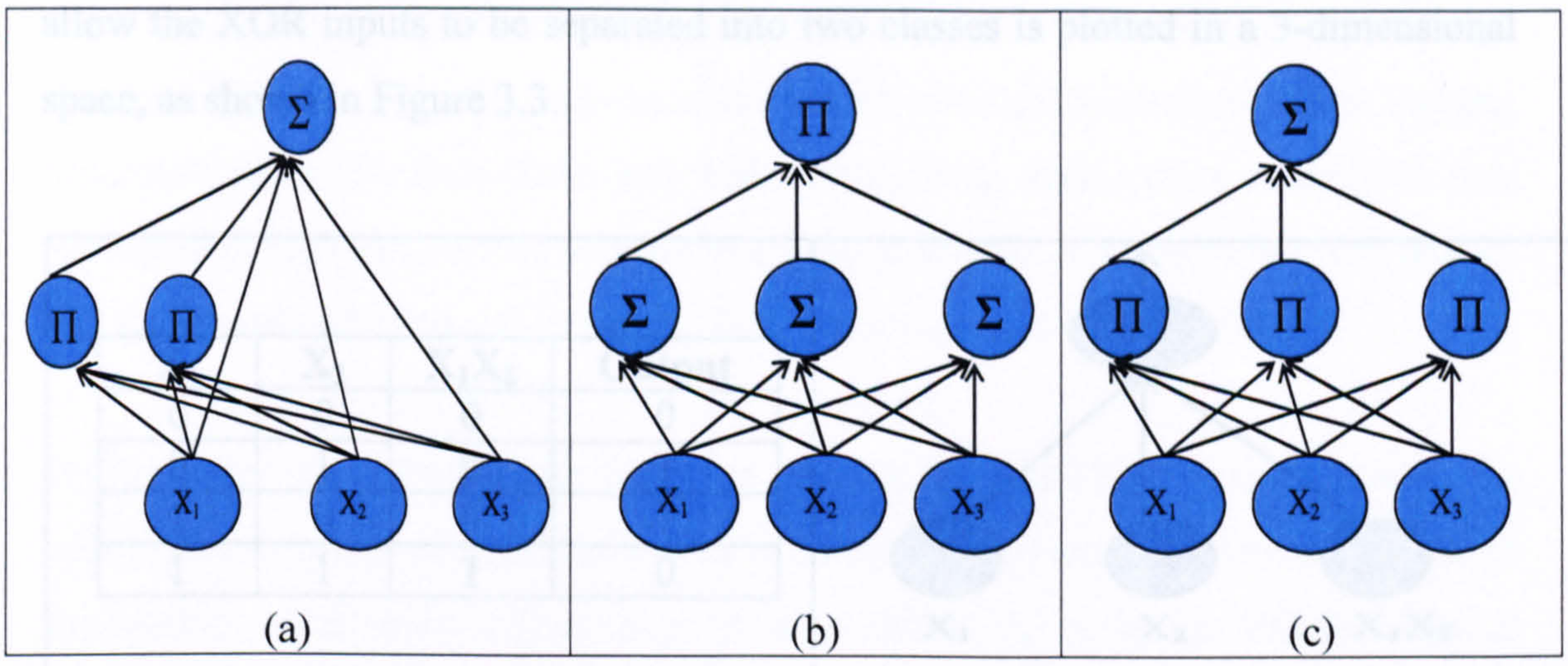


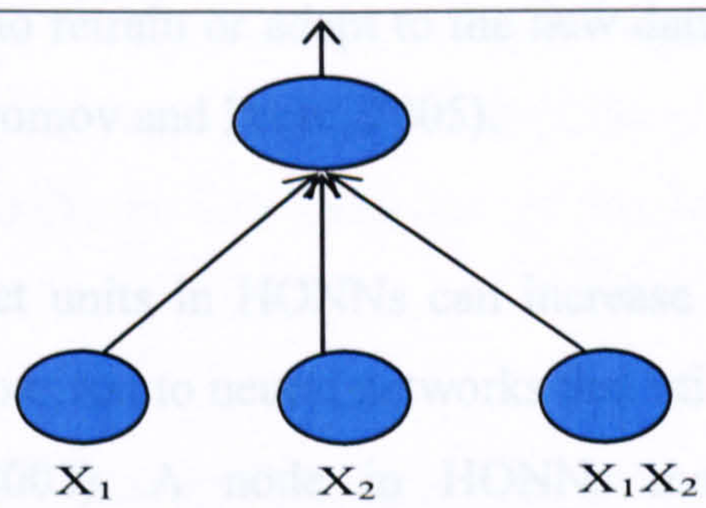
Figure 3.1: Groups of HONNs;
(a) Input layer with combination of external inputs and their products
(b) Output layer of product unit
(c) Hidden layer of product units

High Order Neural Networks (HONNs) are type of feedforward neural networks which have the combination of summing units and multiplicative units (product units) in the networks. In this research work, the HONNs' architecture is classified into three different groups, as shown in Figure 3.1. The networks provide nonlinear decision boundaries offering a better classification capability than the linear neuron (Guler and Sahin, 1994). A major advantage of HONNs is that only one layer of trainable weights is needed to achieve nonlinear separable, unlike the typical Multilayer Perceptron (MLP) or feedforward networks (Park et al., 2000). This results in faster training. The nonlinearity is introduced into the HONNs by having multi-linear interactions between their inputs or neurons which enable them to expand the input space into higher dimensional space. This lead to an easy separation of nonlinear separable classes where linear separability is possible or a reduction in the dimension of the nonlinearity is achieved. For example, the XOR problem could not be solved with a network without a hidden layer or by a single layer of first-order units, as it is not linearly separable. To demonstrate this, Figure 3.2 shows the XOR problem which has two inputs, and the 2^{nd} order HONN architecture. The same problem, however, is easily solved if the patterns are represented in three dimensions in terms of an enhanced representation (Pao, 1989), by just using a single layer HONN with second-order terms. The resulting linearly separable hyperplane which

allow the XOR inputs to be separated into two classes is plotted in a 3-dimensional space, as shown in Figure 3.3.

X_1	X_2	X_1X_2	Output
0	0	**0**	0
0	1	**0**	1
1	0	**0**	1
1	1	**1**	0

(a)



(b)

Figure 3.2: (a) Truth table for XOR problem, (b) 2nd order HONN with two inputs

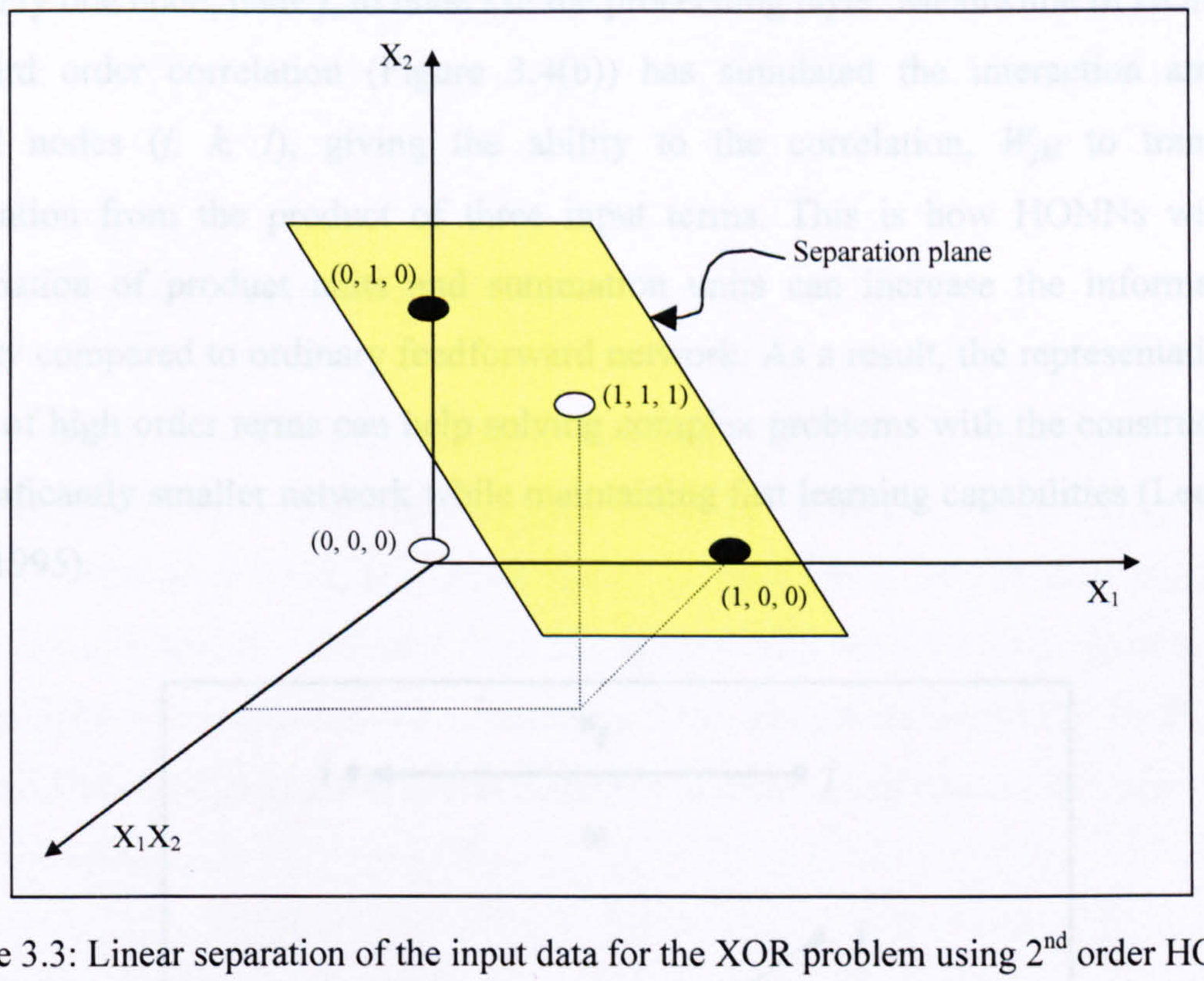


Figure 3.3: Linear separation of the input data for the XOR problem using 2nd order HONN

HONNs can achieve similar performance to that of standard multilayer neural network using a single layer of trainable weights (Park et al., 2000). They are simple in their architecture and require fewer numbers of weights to learn the underlying equation when compared to ordinary feedforward networks, in order to deliver the

same input output mapping (Leerink et al., 1995; Giles and Maxwell, 1987; Shin and Ghosh, 1995). As a result, they can learn faster since each iteration of the training procedure takes less time (Cass and Radl, 1996). This makes them suitable models for complex problem solving where the ability to retrain or adapt to the new data in real time is critical (Pau and Phillips, 1995; Artyomov and Pecht, 2005).

On the other hand, high order terms or product units in HONNs can increase the information capacity of neural networks in comparison to neural networks that utilise summation units only (Yonghong et al., 2003). A node in HONNs model (particularly of type (a) and (c) from Figure 3.1), could receive information from more than one nodes only via one weight connection, as this special criteria is never exist in MLP network. Figure 3.4(a) is an example of signal flow in original feedforward network where the first order correlation can transmit the information from only one node, node j, to node i in the proceeding layer. Meanwhile in HONNs, the third order correlation (Figure 3.4(b)) has simulated the interaction among several nodes (j, k, l), giving the ability to the correlation, W_{jkl} to transmit information from the product of three input terms. This is how HONNs with a combination of product units and summation units can increase the information capacity compared to ordinary feedforward network. As a result, the representational power of high order terms can help solving complex problems with the construction of significantly smaller network while maintaining fast learning capabilities (Leerink et al., 1995).

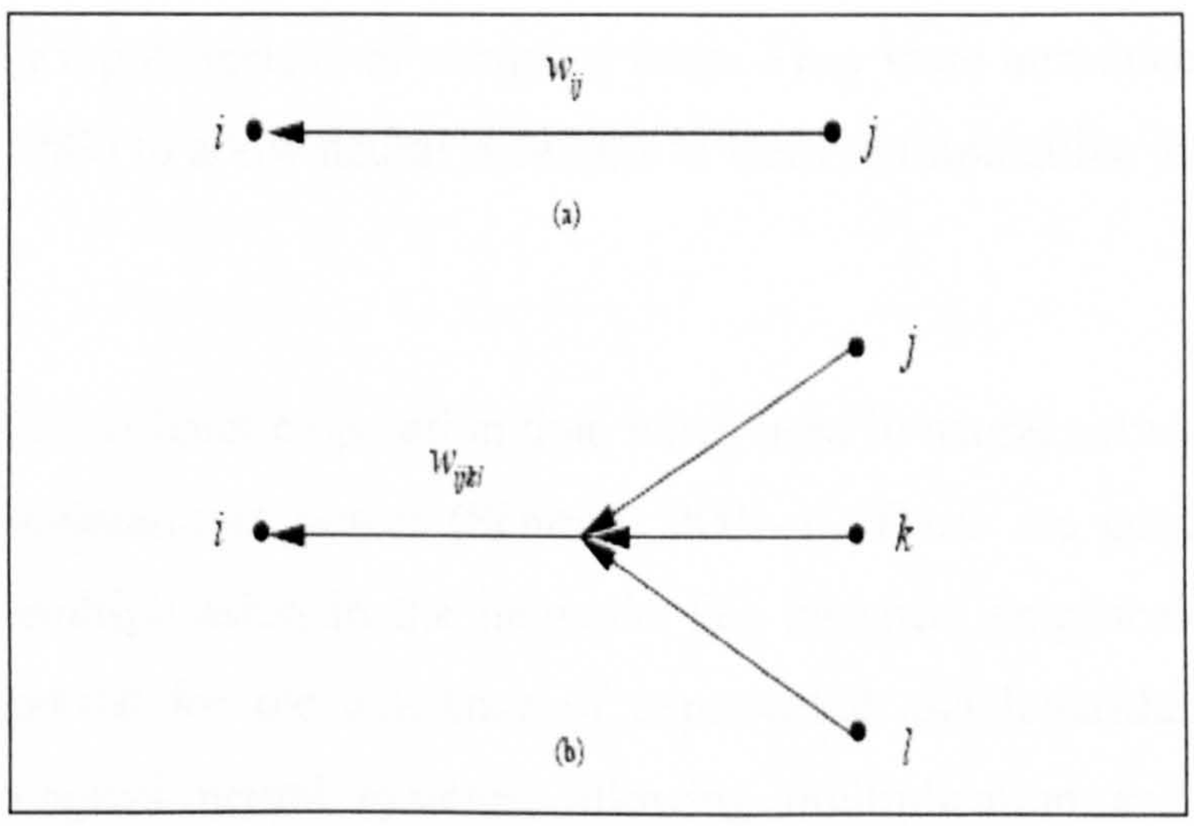


Figure 3.4: (a) 1st order weight correlation, (b) 3rd order weight correlation

According to Patra and Pal (1995), since MLP structure is multilayered and the Backpropagation (BP) algorithm involves high computational complexity, this structure requires excessive training time for learning. Furthermore, the number of weights and in turns the training time increases as the number of layers and the nodes in a layer increases. In contrast, the HONNs structures are single layered of learnable weights and thus the training time will potentially be less than that of the MLP structure.

HONNs are endowed with certain unique characteristics; stronger approximation property, faster convergence rate, greater storage capacity, and higher fault tolerance than lower-order neural networks (Wang et al., 2006). The networks have been considered as good candidate for invariance geometric transformation, due to their design flexibility for given geometric transforms, robustness to noisy and/or occluded inputs, inherent fast training ability, and nonlinear separability (Park et al., 2000)

3.3 Product Units in HONNs

Standard neural networks models use a summation function 'Σ' which performs a linear weighted sum of the inputs. Apart from the utilization of summing units, HONNs on the other hand, also make use of product terms 'Π'. Product units are normal neurons that are different from the most widely used neurons types in that they multiply their inputs instead of summing them. They were introduced by Durbin and Rumelhart (1989) to allow neural networks to learn multiplicative interactions of arbitrary degree.

Multiplication is an arithmetic operation that, when used in neural networks, helps to increase their computational power (Schmitt, 2001-a). There are good reasons to explicitly apply multiplication in the network. For instance, empirical evidence is available and reported for the existence of exponential and logarithmic dendritic processes in biological neural systems, allowing multiplication and polynomial processing (Schmitt, 2001-a). Consequently, as argued in (Durbin and Rumelhart,

1990), in order to model biological neural networks, one should extend the standard MLP model with multiplicative or product units. Further, biological nets make use of nonlinear activation components in the form of axo-axonic synapses performing pre-synaptic inhibition (Neville et al., 2000). The simplest way of modelling such synapses and introducing increased node complexity is to use multi-linear activation, which is the node's activation is in 'higher order' nodes form (Rumelhart et al., 1986), resulting the use of nonlinear activation components.

According to Durbin and Rumelhart in their work (1989), there are various ways in which product units could be used in a network. One way is for a few of them to be made available as inputs to the network in addition to the original raw inputs (refer to Figure 3.1 (a)). Alternatively, they can be used as the output of the network itself (Figure 3.1 (b)). The other way of utilizing them is a whole hidden layer of product units, feeding into a subsequent layer of summing units (Figure 3.1 (c)). The attraction is rather in mixing both types of units; product unit and summing unit, so that product units are mainly used in a network where they occur together with summing units.

Product units have been proven computationally more powerful than summing units in many learning applications. Networks with product units have increased information capacity and the ability to form higher-order combinations of inputs. Durbin and Rumelhart (1989) determined empirically that the information capacity of the product units (measured by their capacity for learning random Boolean patterns) is approximately $3N$, compared to $2N$ of a network with additive units for a single threshold logic function, where N denotes the number of inputs to the network. There are many researches in the literature which show that a network with combination of summing unit and product units could possibly enhance the network performance (Leerink et al., 1995; Ismail and Engelbrecht, 2002; Schmitt, 2001-a; Sanzogni et al., 2000; Schmitt, 2001-b; Durbin and Rumelhart, 1989; Durbin and Rumelhart, 1990; Estudillo et al., 2006).

3.4 Types of HONNs

This section introduces a few types of HONNs. These include the Functional Link Neural Network, Pi-Sigma Neural Network, and Ridge Polynomial Neural Network. Each one of them employs the powerful capabilities of product units with some combinations of summing units. With different strength and capabilities, a structure and characteristic of these networks is elaborated and discussed below, as well as their training algorithms and applications in use.

3.4.1 Functional Link Neural Network (FLNN)

FLNN was first introduced by Giles and Maxwell (1987) who referred to the network as 'Higher Order Neural Network'. Pao (1989) further analyzed the network, referred to them as Functional Link Neural Network. The network naturally extends the family of theoretical feedforward network structure by introducing nonlinearities in inputs patterns enhancements (Durbin and Rumelhart, 1989). These enhancement nodes act as supplementary inputs to the network, and effectively increase the dimensionality of the input vector. Hence the hyperplane generated by the FLNN provides greater discrimination capability in the input pattern space (Pao, 1989). In FLNN, the input information is increased without adding extra input patterns, nevertheless the representation has apparently been enhanced. The network can use higher order correlations of the input components to perform nonlinear mappings using only a single layer of units.

Pao (1989) proposed two FLNNs models; the functional expansion model and the tensor (outerproduct) model. In the functional expansion model, the functional link acts on each node singly, in which it simply applies one or more univariate functions to each input. The input space is expanded by passing the inputs of the network to the univariate functions and including the output of those functions into the network input units. As illustrated in Figure 3.5 (a), each component of the input vector is enhanced by the functional link to yield the quantities $f_1(x)$, $f_2(x)$... $f_n(x)$. The functions $f(x)$ might simply be included (but are not limited to) logical (*AND, OR,*

XOR), trigonometric (*sin, cos*) and joint activations (x_1x_2, etc), such as x, x_2, x_3, ..., or x, $sin\pi x$, $cos\pi x$, $sin2\pi x$, $cos\pi x$, and so on, depending on the set of functions that one want to use.

In the tensor or outerproduct model (refer to Figure 3.5 (b)), each component of the input pattern multiplies the entire input pattern vector. The functional link in this case generates an entire vector from each of the individual components. The effect of the nonlinear functional transform is to change the representation of the input pattern so that, instead of being described in terms of a set of components $\{x_i\}$, it is described as $\{x_i , x_i x_j\}$, where $j \geq i$, or as $\{x_i , x_i x_j, x_i x_j x_k\}$, where $k \geq j \geq i$, and so on. Therefore no new information has been added, but joint activations have been made available to the network. Such functional transforms greatly increase the number of components in terms of which the input pattern is described.

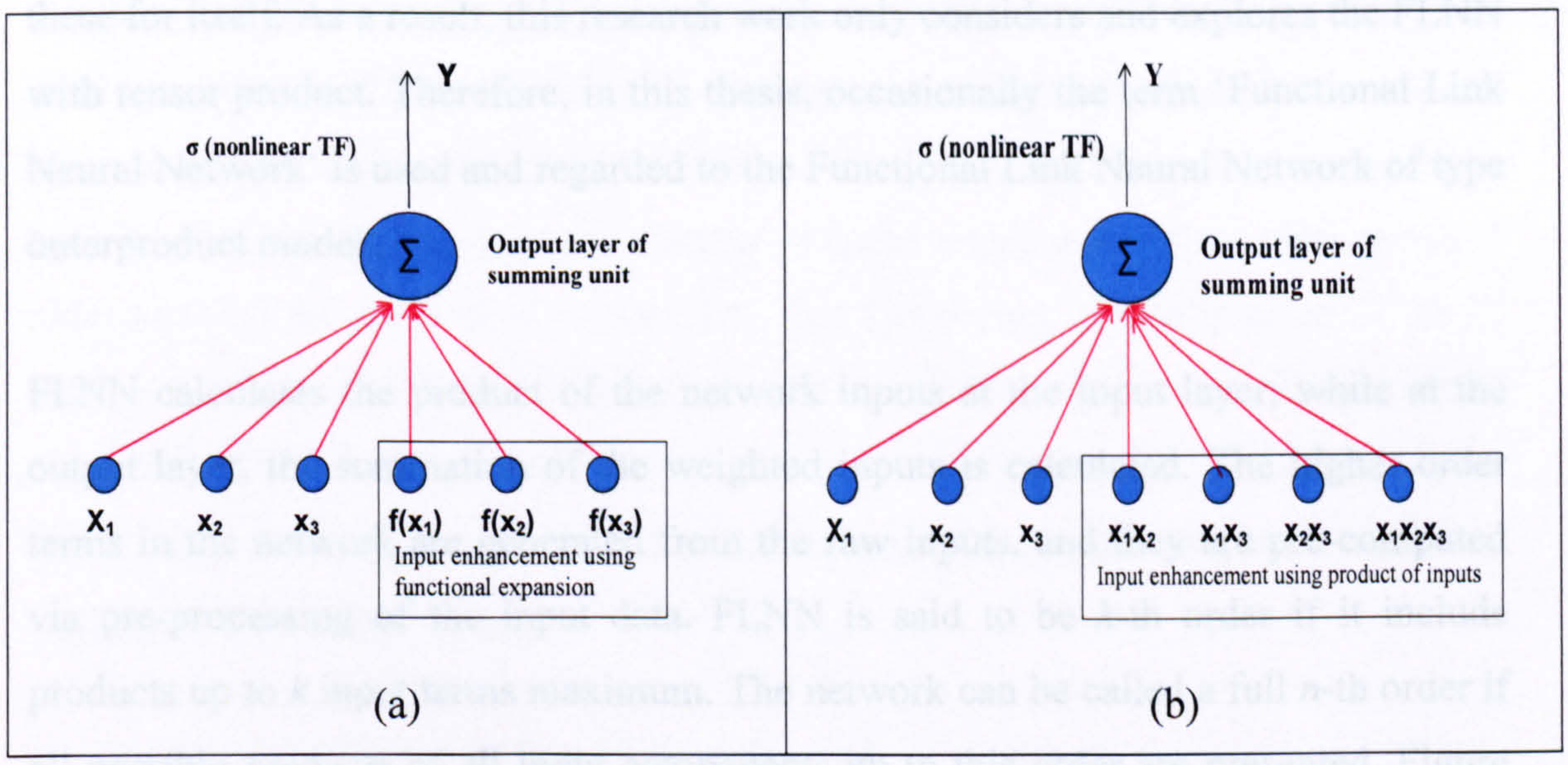


Figure 3.5: (a) The FLNN of type functional expansion model, (b) The FLNN of type tensor product model.
Bias nodes are not shown here for reason of simplicity.

Both models may be used simultaneously and in combination, as appropriate (Pao, 1989). The outerproduct model truly introduced higher-order terms in the enhanced representation in the sense that some of these terms represent joint activations. In contrast, the functional expansion model merely expands the dimension of the representation space without introducing joint activations, and without involving any

interactions between inputs. However, with the functional expansion model, the responsibility lies to the user to choose an appropriate set of functions to deal with the problem at hand. Most nonlinear problems are complex and there is little information about them. Thus the decision of choosing a good set of functions from a near infinite set of possibilities is difficult (Patra and Bos, 2000). One possible answer currently being explored is the use of evolutionary computation to select the function set (Sierra, 2001).

On the other hand, the outerproduct model uses only joint activations between the inputs to expand the input space. Limiting the higher-order terms to joint activations avoids the problem of selecting which functions to use to expand the input space. Moreover, an interesting class of nonlinear transformations which transform the input pattern vectors into higher order tensors allows a direct representation of higher order correlations between input features, rather than forcing the network to discover these for itself. As a result, this research work only considers and explores the FLNN with tensor product. Therefore, in this thesis, occasionally the term 'Functional Link Neural Network' is used and regarded to the Functional Link Neural Network of type outerproduct model.

FLNN calculates the product of the network inputs at the input layer, while at the output layer, the summation of the weighted inputs is calculated. The higher order terms in the network are generated from the raw inputs, and they are pre-computed via pre-processing of the input data. FLNN is said to be k-th order if it include products up to k input terms maximum. The network can be called a full n-th order if all possible products of all input components up to this order are presented. Figure 3.1 (a) shows an example of third order FLNN with 3 external (raw) inputs x_1, x_2, and x_3 and four high order inputs which are x_1x_2, x_1x_3, x_2x_3, and $x_1x_2x_3$ act as supplementary inputs to the network, making the total inputs of the network seven.

Below is the basic equation for calculating FLNN's output of 3rd degree:

$$Y = \sigma\left(W_o + \sum_i W_i X_i + \sum_i \sum_j W_{ij} X_i X_j + \sum_i \sum_j \sum_k W_{ijk} X_i X_j X_k \right) \tag{3.1}$$

where 'σ' is a nonlinear transfer function, w_0 is the adjusted threshold, and w_i, w_{ij}, w_{ijk} are adjustable weights that link the external inputs x_i, x_j, x_k and also the high order inputs, x_ix_j, x_ix_k, x_jx_k, and $x_ix_jx_k$ to the output node.

FLNN unfortunately suffers from the explosion of weights where the number of weights required to accommodate all high order correlations increase with the number of input dimension *d*, and the desired order of the network, *k*. A *k-th* degree of FLNN needs a total of:

$$\sum_{i=0}^{k} \left(\frac{d+i-1}{i} \right) = \left(\frac{d+k}{k} \right) \tag{3.2}$$

weights if all products of up to *k* components are to be employed (Shin and Ghosh, 1995). A large number of free weights eliminate the advantages of quick training and low complexity that is the basis for using networks with higher-order terms. It is therefore necessary to restrict the number of input nodes and higher order terms in order to avoid the curse of dimensionality. For that reason, normally up to 2nd or 3rd order networks are considered in practice (Thimm, 1995; Kaita et al., 2002; Park et al., 2000; Pao, 1989). Furthermore, since the networks do not allow terms like X_I^K $(k > 1)$, a single layer FLNN is not capable of approximating some functions well (Shin and Ghosh, 1995).

Despite FLNN has one major drawback, the network is likely to solve problem in an elegant and simple manner. Learning in FLNN is often quicker, although this is highly dependent on the specific problem and implementation design. Since there is no hidden layer in the FLNN, the computational requirement is drastically reduced compared to that of MLP (Patra and Bos, 2000). As a consequence of simpler architecture, it has the ability to reduce computational cost in the training stage, whilst maintaining good performance of approximation (Mirea and Marcu, 2002). The reduced number of free weights compared with MLP means that the problems of over-fitting and local minima can be migrated to large degree.

3.4.1.1 Learning Algorithm of the FLNN

Learning algorithm for Functional Link Neural Network used in this research work is based on the incremental backpropagation algorithm (Haykin, 1999). The Mean Squared Error function (MSE) is as follows:

$$E = \frac{1}{N}\sum_{p=1}^{N}\left(d^p - y^p\right)^2 \tag{3.3}$$

where superscript p denotes the p-th training example, d^p is the target output, whereas $y^p = \sigma\left(\sum_j h_j^p\right)$ is the network predicted output.

The learning algorithm for FLNN can be divided into the following:

For each training example,

- Calculate the output

  Functional Link Neural Network computes the output:

$$y = \sigma(w_0 + \sum_j w_j x_j + \sum_{j,k} w_{jk} x_j x_k + \sum_{j,k,l} w_{jkl} x_j x_k x_l + \ldots\ldots), \tag{3.4}$$

  where σ is a nonlinear transfer function, w_0 is the threshold, x is the component of input vector X, and w are the trainable weights.

- compute the benefit β at output node

$$\beta = (d_i - y_i) y_i (1 - y_i) \tag{3.5}$$

- compute the weight changes

  Delta weight for FLNN is:

$$\Delta W_i = \eta \beta X_k \tag{3.6}$$

- Update the weight

$$W_i = W_i + \Delta W_i \tag{3.7}$$

Until termination condition is satisfied.

3.4.1.2 FLNNs' Applications

Functional Link Neural Networks have been successfully applied in variety of problems, such as system identification (Mirea and Marcu, 2002), pattern recognition (Kaita et al, 2002; Artyomov and Pecht, 2005), intelligent control (Patra and Bos, 2000), process optimization (Cass and Radl, 1996), signal processing (Patra and Pal, 1995), and optimal control (Pau and Phillips, 1995).

A research to analyze the robustness problem in Fault Detection and Isolation (FDI) has been conducted by Chow and Teeter (1997). They proposed a generalized version of Dynamic FLNN (GDFLNN), which was used to perform the Heating, Ventilating and Air-Conditioning (HVAC) thermal system identification and modelling. In order to provide the FLNN with adequate internal memory, an Auto-Regressive Moving Average (ARMA) filter was placed either as a Local Activation Feedback or as a Local Output feedback. The research work successfully proved that using the proposed GDFLNN reduced the design time from several days to several hours for each designated model.

Cass and Radl in their work (1996) used FLNN in process optimization and found that FLNN can be trained much faster than MLP without scarifying computational capability, which makes them more suitable in process modelling applications, where the ability to retrain or adapt to new data in real time is critical. Another research was constructed using FLNN that react invariantly under geometric transformations on the input space (Giles et al, 1998). The model has the advantage of inherent invariance, and only learned the desired signal.

Mirea and Marcu (2002) investigated the development and application of an FLNN with internal dynamic elements to system identifications. The internal dynamic elements are auto-regressive moving average filters (ARMA) that implement local activation feedback and local output feedback. Empirical results suggested that the proposed model reveal better approximation and generalization with reduced training and evaluation time, compared to FLNN with static structure. Meanwhile, Pao (1989) has shown that there is a significant increase in the rate of learning in the case

of FLNN in comparison with the generalized delta rule network, when used to learn the XOR problem.

3.4.2 Pi-Sigma Neural Network (PSNN)

Pi-Sigma Neural Network was first introduced by Shin and Ghosh (1991-b) to overcome the problem of weights explosion in FLNN. The network is a feedforward network with a single hidden layer and product units at the output layer. PSNN calculates the product of sum of the input components instead of the sum of products as in FLNN.

The motivation was to develop a systematic method for maintaining the fast learning property and powerful mapping capability of single layer FLNN whilst avoiding the combinatorial explosion in the number of free parameters when the input dimension is increased. In contrast to FLNN, the number of free parameters in PSNN increases linearly to the order of the network. For that reason, PSNN can overcome the problem of weights explosion that occurs in FLNN which rise exponentially to the number of inputs. Shin and Ghosh (1991-b) argued that PSNN not only requires less memory (weights and nodes), but typically needs at least two orders of magnitude less number of computations as compared to the MLP for similar performance level, and over a broad class of problems.

Figure 3.6 shows a PSNN with a single output. The network architecture of PSNN consists of two layers; the product layer and the summing layer. The inputs are connected to the summing layer by trainable weighted connections. The output from this layer is passed to the product unit (by non-trainable connections set to unity), which passes the signal through a nonlinear transfer function to produce the network output. For each increase in order, only one extra summing unit is required. The product units give the networks higher-order capabilities without suffering from the exponential increase in weights, which is a major problem in a single layer HONNs. The output of the PSNN is computed as follows:

$$Y = \sigma\left(\prod_{j=1}^{K} \sum_{k=1}^{N} \left(W_{kj} X_k + W_{jo} \right) \right) \tag{3.8}$$

where W_{kj} are adjustable weights, W_{j0} are the biases of the summing units, X_k is the input vector, K is the number of summing units (alternatively, the order of the network), N is number of input nodes, and 'σ' is a nonlinear transfer function.

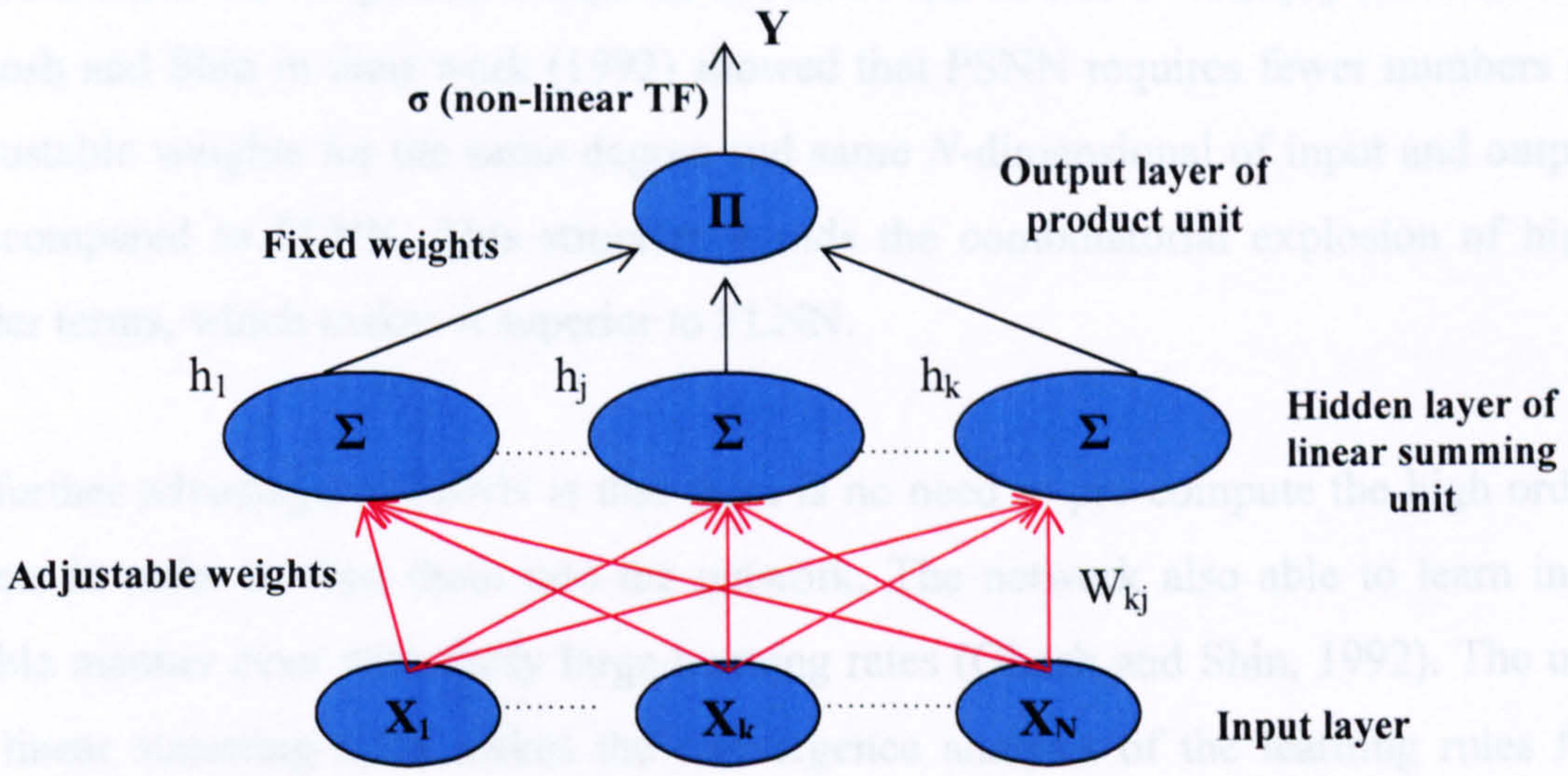


Fig. 3.6: Pi Sigma Neural Network of *K*-th order.
Bias nodes are not shown here for reason of simplicity.

It has a topology of a fully connected two-layered feedforward network. Since there are K summing units incorporated, it is called a K-th order PSNN. The number of summing units signifies the order of the network, i.e., a second order PSNN has two summing units, while a 3rd order PSNN has 3 summing units, and so on. In this case, W_{kj} from input X_k to the j-th summing units is a trainable weight. The weights between the summing and the output layer are fixed to unity, and they are not trainable. For that reason, the summing layer is not "hidden" as in the case of MLP. Such a network topology with only one layer of trainable weights drastically reduces the training time.

The structure of PSNN is highly regular in the sense that summing units can be added incrementally till an appropriate order of the network is achieved without over-fitting of the function, and without disturbing any connection established

previously. The order can be gradually increased until the desired low predefined error is reached. If multiple outputs are required, an independent summing layer is needed for each output. Thus, for an M-dimensional output vector y, and N-dimensional input vector x, a total of $\sum_{i=1}^{M}(N+1).K_i$ weights connections are needed, where K_i is the number of summing units for the i-th output. This allows a great flexibility since all outputs do not have to retain the same complexity. The reduction in the number of weights as compared to FLNN allows PSNN to enjoy fast training. Ghosh and Shin in their work (1992) showed that PSNN requires fewer numbers of adjustable weights for the same degree and same N-dimensional of input and output as compared to FLNN. This structure avoids the combinatorial explosion of high order terms, which makes it superior to FLNN.

A further advantage of PSNN is that there is no need to pre-compute the high order terms in order to feed them into the network. The network also able to learn in a stable manner even with fairly large learning rates (Ghosh and Shin, 1992). The use of linear summing units makes the convergence analysis of the learning rules for PSNN more accurate and tractable. The price to be paid is that the PSNN is not a universal approximator.

PSNN combines the fast learning abilities of single-layered feedforward networks with the nonlinear mapping of higher order neural networks, while using much fewer numbers of units. Despite not being a universal approximator, PSNN demonstrated competent ability to solve many scientific and engineering problems, as discussed in section 3.4.2.2.

3.4.2.1 Learning Algorithm of PSNN

Learning algorithm for Pi-Sigma Neural Network used in this research work is based on the gradient descent on the estimated Mean Squared Error (MSE), which is calculated as follows:

$$E = \frac{1}{N}\sum_{p=1}^{N}\left(d^p - y^p\right)^2 \tag{3.9}$$

where p denotes the p-th training data, d^p is the target output, whereas $y^p = \sigma\left(\prod_j h_j^p\right)$ is the network predicted output.

The learning algorithm for PSNN can be divided into the following:

For each training example,

- Calculate the output

  PSNN computes the output:

$$Y = \sigma\left(\prod_{j=1}^{K}\sum_{k=1}^{N}\left(W_{kj}X_k + W_{jo}\right)\right) \tag{3.10}$$

- compute the benefit β at output node

$$\beta = (d_i - y_i)y_i(1-y_i) \tag{3.11}$$

- compute the weight changes

  The delta weight is:

$$\Delta w_i = \eta\beta \prod_{j\neq i}^{M} h_j \quad x_k \tag{3.12}$$

- Update the weight

$$W_i = W_i + \Delta W_i \tag{3.13}$$

Until termination condition is satisfied.

3.4.2.2 PSNNs' Applications

Previous research work found that PSNNs are good models for various applications. Shin et al (1992) investigated the applicability of PSNN for shift, scale and rotation invariant pattern recognition. Preliminary results for both function approximation

and classification are extremely encouraging, and showed a faster performance of about two orders of magnitude over backpropagation to achieve similar quality of solution. Another work of Shin and Ghosh (1991-a) has introduced a so-called Binary Pi-Sigma Neural Network with binary input/output and the hardlimiting activation function instead of continuous input/output and sigmoidal activation function. Simulation results demonstrated that for low learning rates, the MSE always decreasing, indicating the stability of the asynchronous learning algorithm used. On the other hand, for large problem sizes, perfect learning was still achieved even with MSE ≥ 1, indicating the difficulty of the underlying mapping problems.

Ghosh and Shin (1992) used both analog PSNN (Shin and Ghosh, 1991-b) and binary PSNN (Shin and Ghosh, 1991-a) for classification and function approximation problems. Their results showed that PSNNs yield comparable or better results than single layer HONN, and when compared to the MLP, both PSNNs and single layer HONN performed much better in terms of giving correct solutions within a short training time.

Hussain and Liatsis (2002) proposed a new Recurrent Polynomial Network for predictive image coding that explores both multi-linear interactions between the input pixel as well as the temporal dynamics of the image formation process. They have extended the architecture of ordinary PSNN to include a recurrent connection from the output to the input layer. The network does not suffer from a slow convergence rate and because of the feedback connections and the existence of high order terms, it can be applied to highly nonlinear problem.

Knowles (2005) has investigated several types of HONNs including the PSNN for financial time series prediction. He has extended the use of PSNN in two different structures; the recurrent PSNN and the pipelined PSNN. Results showed that the pipelined PSNN is computationally more efficient than the ordinary pipelined recurrent network, as it maintain the same level of signal tracking abilities while using less weight.

3.4.3 Ridge Polynomial Neural Network (RPNN)

Although Pi-Sigma Neural Network has shown to provide good results in classification and function approximation, the network however is not a universal approximator due to the utilization of a reduced number of interconnected weights. To evade this drawback, Shin and Ghosh (1995) have introduced the Ridge Polynomial Neural Network; a generalization of PSNN, and the network is a universal approximator. RPNN has a well regulated structure which is constructed by adding gradually more complex PSNNs, therefore preserving all the advantages of PSNN.

Any multivariate polynomial can be represented in the form of a ridge polynomial and realized by RPNN whose output is determined according to the following equations (Shin and Ghosh, 1995):

$$f(x) = \sigma \sum_{i=1}^{N} P_i(x) \tag{3.14}$$

$$P_i(x) = \prod_{j=1}^{i} \left(\langle X, W_j \rangle + W_{j0} \right), i = 1, \ldots\ldots, N.$$

where 'σ' denotes a suitable nonlinear transfer function, typically the sigmoid transfer function, W_{j0} are the biases of the summing units in the corresponding PSNN units, N is the number of Pi-Sigma blocks used (or alternatively, the order of the RPNN), and $\langle X, W \rangle$ is the inner product of weights matrix W, and input vector X, such that:

$$\langle X, W \rangle = \sum_{i=1}^{d} x_i w_i \tag{3.15}$$

The details on the representation theorem to proof can be found in (Shin and Ghosh, 95).

RPNN can approximate any multivariate continuous functions on a compact set in multidimensional input space, with arbitrary degree of accuracy. In contrast to FLNN

which use multivariate polynomials that causes an explosion of weights, RPNN and PSNN utilize univariate polynomials which are easy to handle (Shin and Ghosh, 1995). Similar to the PSNN, RPNN has only a single layer of adaptive weights. The structure of RPNN, as shown in Figure 3.7, is highly regular in the sense that pi-sigma units can be added incrementally until an appropriate order of the network or the desired low predefined error is achieved without over-fitting of the function.

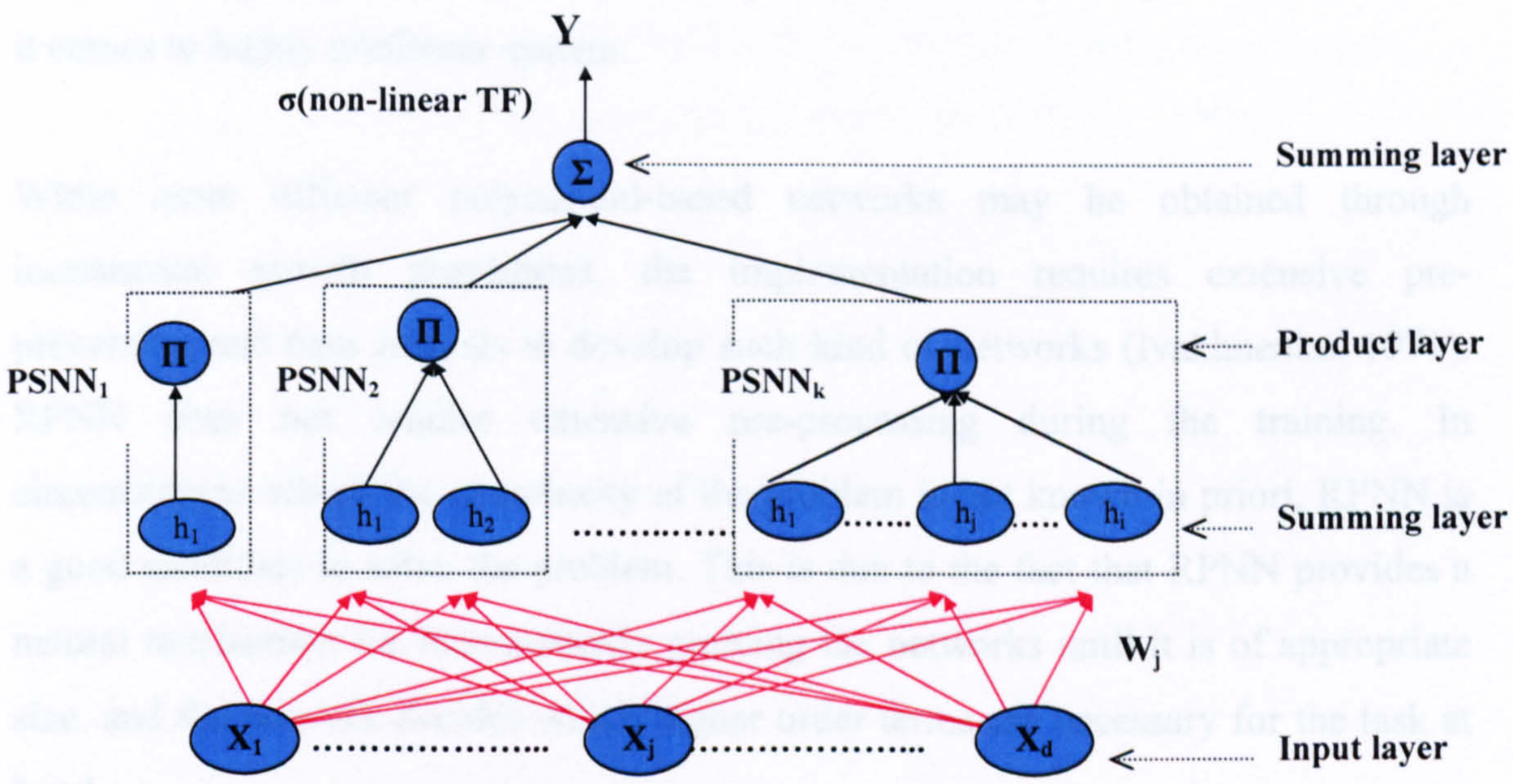


Figure 3.7: The Ridge Polynomial Neural Network of *k*-th order
Bias nodes are not shown here for reason of simplicity.

where k is the number of PSNN units used, h is summing unit (hidden unit) in each PSNN, and W_j are trainable weights.

RPNN provides a natural mechanism for incremental network growth, by which the number of free parameters is gradually increased. Unlike other growing networks such as self-organizing neural networks (SONN) (Tenorio and Lee, 1990) and the group method of data handling (GMDH) (Ivakhnenko, 1971), in which their structure grow to any arbitrary number of hidden layers and nodes, RPNN has a well regulated architecture.

As agued by Nikolaev and Iba in their work (2003), the constructive polynomial networks like GMDH and SONN do not attempt to improve the weights further once

the network is built. The reason is that the estimation of the network weights near the input layer is frozen when estimating the weights near the output layer, and the estimation of weights near the output layer does not influence the weights near the input layers. As a result, the network weights are not sufficiently tuned so that they are in tight interplay with respect to the concrete structure. Oh et al (2003) claimed that GMDH have some drawbacks; it tends to generate quite complex polynomial for relatively simple system, and also tends to produce an overly complex network when it comes to highly nonlinear system.

While more efficient polynomial-based networks may be obtained through incremental growth procedures, the implementation requires extensive pre-processing and data analysis to develop such kind of networks (Ivakhnenko, 1971). RPNN does not require extensive pre-processing during the training. In circumstances where the complexity of the problem is not known in priori, RPNN is a good candidate to solve the problem. This is due to the fact that RPNN provides a natural mechanism for incrementally growing the networks until it is of appropriate size, and the network decides which higher order terms are necessary for the task at hand.

3.4.3.1 Learning Algorithm of the RPNN

Since RPNN is a generalization of Pi-Sigma Neural Network, they adopt the same learning rule. Referring to equation (3.14), it is shown that P_i is obtainable as the output of a PSNN of order i. Therefore, the learning algorithm developed for the PSNN can be used for the RPNN, in addition to the constructive learning procedure (Shin and Ghosh, 1995). The algorithm can be divided into the following steps:

1. Initialization step: RPNN's order = 1. Assign suitable values for threshold *r*, learning rate *n*, *dec_r* and *dec_n*.
2. For all training patterns, do:
    - Calculate actual network output
    - Update the weights asynchronously

3. At the end of each epoch, calculate the error for the current epoch, e_c.
4. If $e_c < e_{th}$ or $t > t_{th}$,
    - Stop the training
5. Else do
    - If $\left|(e_c - e_p)/e_p\right| < r$
        - Add higher order Pi-Sigma unit
        - Reduce the threshold r; $r = r * dec\_r$
        - Reduce the learning rate n; $n = n * dec\_n$
        - $e_p = e_c$
        - order = order + 1
        - $t = t + 1$
        - Go to step 2
    - Else do
        - $t = t + 1$
        - $e_p = e_c$
        - Go to step 2

where e_c is the MSE for the current epoch, and e_p is the MSE for the previous epoch, e_{th} is threshold MSE for the training phase, t is number of training epoch, and t_{th} is threshold epoch to finish the training. Notice that every time a higher order PSNN is added, the weights of the previously trained PSNN networks are kept frozen, whilst the weights of the latest added PSNN are trained. During the training, only the weights of the latest added pi-sigma unit are attuned asynchronously. The algorithm for the RPNN endows the network with a parsimonious approximation of an unknown function in terms of network complexity (Shin and Ghosh, 1995).

3.4.3.2 RPNNs' Applications

RPNNs have become valuable computational tools in their own right for various tasks such as pattern recognition (Voutriaridis et al, 2003), image prediction (Liatsis and Hussain, 1999), function approximation (Shin and Ghosh, 1995; Shin and

Ghosh, 1992; Voutriaridis, 2003), time series prediction (Tawfik and Liatsis, 1997), data classification (Shin and Ghosh, 1995), and intelligent control (Karnavas and Papadopoulos, 2004). Liatsis and Hussain (1999) have presented a new 1-D predictor structure for Differential Pulse Code Modulation (DPCM) which utilizes Ridge Polynomial Neural Network. They found that, in the case of 1-D image prediction, the 3rd order RPNN can achieve high signal to noise ratio compression results. At a transmission rate of 1 bit/pixel, the 1-D RPNN system provides on average 13 dB improvements in the signal to noise ratio over the standard linear DPCM and a 9 dB improvement when compared to single layer HONN.

Voutriaridis et al. (2003) examined the capability of RPNNs in pattern recognition and function approximation. They used features from the image block representation of the characters and traditional invariant moments to test the ability of RPNNs as object classifiers. Meanwhile, to examine the powerful of RPNNs as approximators, they tested the networks to a number of multivariable functions. Simulation results demonstrated that RPNNs can give satisfactory results with significantly high recognition rate when used in character recognition and act as reliable approximators when used in function approximation.

The architecture of RPNNs has been tested successfully on a 4-carrier Orthogonal Frequency Division Multiplexing (OFDM) system (Tertois, 2002). The networks were placed in the receiver, and corrected the nonlinearities introduced by the transmitter's high-power amplifier. RPNNs in their work have shown good results in simulations and improved the performance of OFDM systems, or keep the same performance with lower power consumption.

Shin and Ghosh in their work (1995) have tested the network with a surface fitting problem, the classification of high dimensional data, and the realization of a multivariate polynomial function. They highlighted the capabilities of RPNN in comparison to MLP, Cascade Correlation, and Optimal Brain Damage (OBD). Result showed that the RPNN trained with the constructive learning algorithm provided a smooth and steady learning. Simulation results indicated that RPNN used less computation and memory (number of units and weights). Unlike OBD and MLP, a

significant advantage of RPNN is that the structure of the network is automatically determined during the training by the network itself.

RPNNs have also been tested for one step prediction of the Lorenz attractor and solar spot time series (Tawfik and Liatsis, 1997). The work proved that RPNNs have a more regular structure with a superior performance in terms of speed and efficiency, and shows good generalization capability when compared to Multilayer Perceptron.

Karnavas and Papadopoulos (2004) presented a design of an intelligent type controller using PSNNs and RPNNs concepts for excitation control of a practical power generating system. Both PSNNs and RPNNs controllers demonstrated good performance over a wide range of operating conditions. Both networks offer competitive damping effects on the generator oscillations, with respect to the Fuzzy Logic Excitation Controller (FLC). They also emphasized that the hardware implementation for the proposed PSNNs and RPNNs controllers is easier than that of FLC, and the computational time needed for real time applications is drastically reduced.

3.5 Chapter Summary

This chapter has investigated the nature and application of Higher-order Neural Networks as nonlinear prediction models. The utilization of higher order terms within the neural networks structure has also been discussed. Three types of HONNs have been discussed extensively; the FLNN, the PSNN, and the RPNN. The networks are computationally efficient nonlinear network and are capable of complex nonlinear mapping between their input and output pattern space. The use of higher order terms allows the networks to expand their input space into higher dimensional space where linear separability is possible. In the next chapter, the Dynamic Ridge Polynomial Neural Network which is expanded through the addition of feedback loop into the RPNN will be introduced.

CHAPTER 4: DYNAMIC RIDGE POLYNOMIAL NEURAL NETWORK

4.1 Introduction

This chapter proposes a novel recurrent neural network architecture. The new neural network architecture incorporates recurrent links into the structure of the ridge polynomial neural network. Feedforward HONNs discussed earlier in Chapter 3 can only implement a static mapping of the input vectors. In order to model dynamical functions of the brain, it is essential to utilize a system that is capable of storing internal states and can implement complex dynamic system. Neural networks with recurrent connections are dynamical systems with temporal state representations. The dynamic structure approach has been successfully used for solving varieties of problems, such as time series forecasting (Zhang and Chan, 2000; Steil, 2006), approximating a dynamical system (Kimura and Nakano, 2000), forecasting a stream flow (Chang et al, 2004), and system control (Reyes et al, 2000). Motivated by the ability of recurrent dynamic systems in real world applications, the proposed Dynamic Ridge Polynomial Neural Network (DRPNN) architecture has been used in this research work.

4.2 The Properties and Network Structure of DRPNN

In linear system, the use of past inputs values creates the Moving Average (MA) models. Meanwhile, the use of the past outputs values creates what is known as the Autoregressive (AR) models. Feedforward neural networks were shown to be a special case of Nonlinear Autoregressive (NAR) models, on the other hand Recurrent Neural Networks (RNNs) were shown to be a special case of Nonlinear ARMA models (NARMA). This means that RNNs have moving average components, therefore showing advantages over feedforward neural networks, similar to the advantages in which ARMA model posses over AR model (Connor et al., 1994).

Hence, RNNs are well suited for time series that posses moving average components (Connor et al., 1994).

Applications in forecasting and signal processing require explicit treatment of dynamics. Feedforward RPNN can only accommodate dynamic systems by including past inputs and target values in an augmented set of inputs. However, this kind of dynamic representation does not exploit a known feature of biological networks, that of internal feedback. The behaviour of the financial signal itself related to some past inputs on which the present inputs depends. DRPNN, on the other hand, incorporates a recurrent connection, and as a consequence of this feedback, the network outputs depend not only on the initial values of external inputs, but also on the entire history of the system inputs. Hence, the introduction of recurrence feedback in the ordinary feedforward RPNN is expected to improve the input-output mapping. This relates to the fact that the proposed DRPNN has the capability of having a memory to solve the underlying task and exhibiting a rich dynamic behaviour.

The rational of placing the recurrent connection from the output layer back to the input layer in the proposed DRPNN is that instead of learning with complex and fully connected recurrent architectures, redundant connections should be eliminated in order to significantly increase the network's generalization capability. This architecture is similar to the Jordan recurrent network (Jordan, 1986). The feedforward part of Jordan network is a restricted case of a non-linear AR model, while the configuration with context units fed by the output layer is a restricted case of non-linear MA model (Beale and Jackson, 1990). From this, the proposed DRPNN which has the feedback connection from the output layer to the input layer is seen to have an advantage over feedforward RPNN in much the same way that ARMA models have advantages over the AR.

The structure of the DRPNN is constructed from a number of increasing order of Pi-Sigma units with the addition of a feedback connection from the output layer to the input layer. The feedback connection feeds the activation of the output node to the summing nodes in each Pi-Sigma units, thus allowing each building block of Pi-Sigma unit to see the resulting output of the previous patterns. In contrast to RPNN,

the proposed DRPNN, as shown in Figure 4.1 is provided with memories which give the network the ability of retaining information to be used later. All the connection weights from the input layer to the first summing layer are learnable, while the rest are fixed to unity.

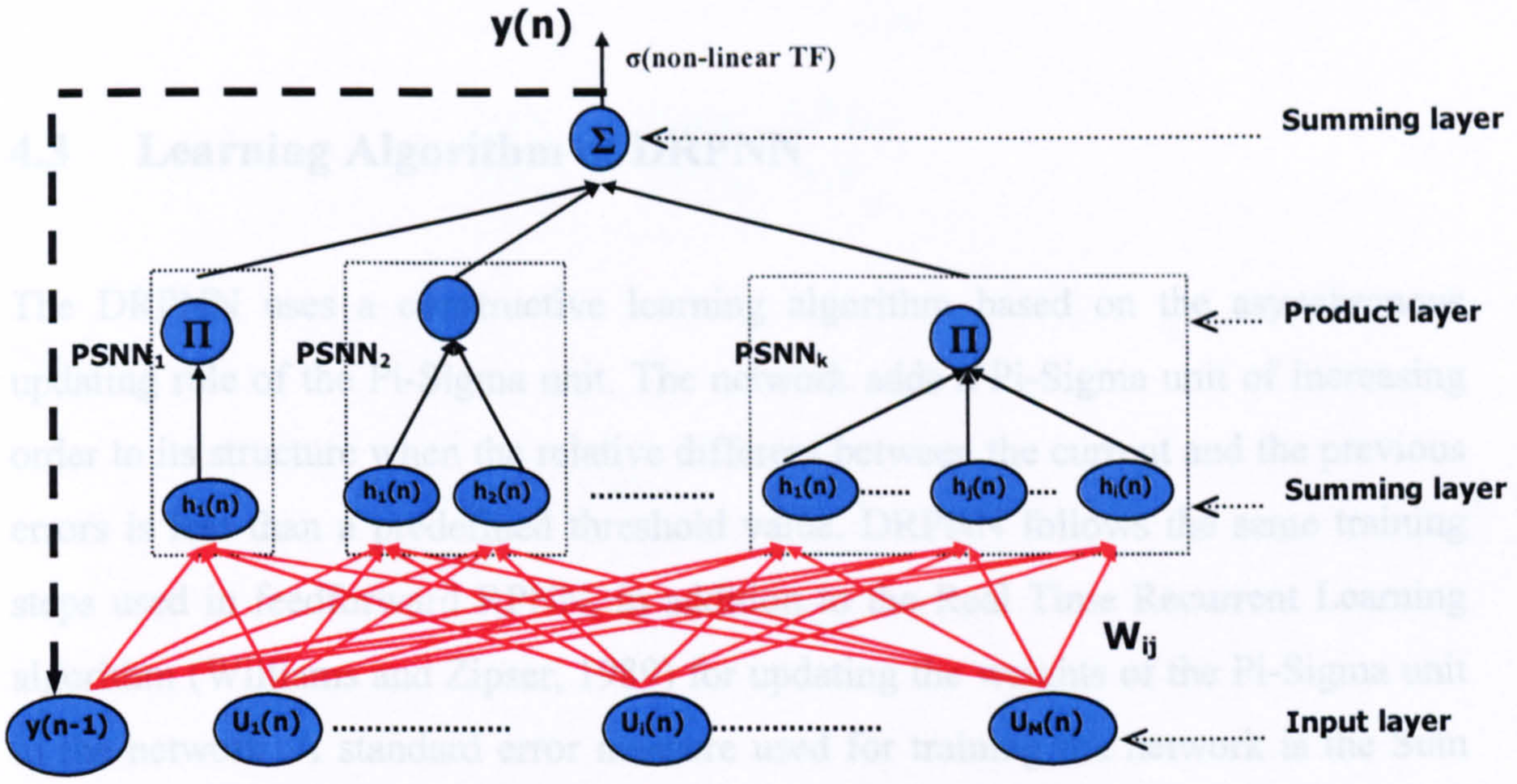


Figure 4.1: Dynamic Ridge Polynomial Neural Network of *k*-th order
(Bias nodes are not shown here for reason of simplicity)

Suppose that M is the number of external inputs $U(n)$ to the network, and let $y(n-1)$ to be the output of the DRPNN at previous time step. The overall input to the network are the concatenation of $U(n)$ and $y(n-1)$, and is referred to as $Z(n)$ where:

$$Z_i(n) = \begin{cases} U_i(n) & if \;\; 1 \leqslant i \leqslant M \\ y(n-1) & i = M+1 \end{cases} \tag{4.1}$$

The output of the k_{th} order DRPNN is determined as follows:

$$y(n) = \sigma\left(\sum_{i=1}^{k} P_i(n)\right)$$

$$P_i(n) = \prod_{j=1}^{i}\left(h_j(n)\right) \quad . \tag{4.2}$$

$$h_j(n) = \sum_{i=1}^{M+1} W_{ij} Z_i(n) + W_{jo}$$

where σ(.) is a suitable nonlinear activation function, k is the number of Pi-Sigma units used, $P_i(n)$ is the output of each PSNN block, $h_j(n)$ is the net sum of the sigma unit in the corresponding PSNN block, W_{jo} is the bias, σ is the sigmoid activation function, and n is the current time step.

4.3 Learning Algorithm of DRPNN

The DRPNN uses a constructive learning algorithm based on the asynchronous updating rule of the Pi-Sigma unit. The network adds a Pi-Sigma unit of increasing order to its structure when the relative different between the current and the previous errors is less than a predefined threshold value. DRPNN follows the same training steps used in feedforward RPNN, in addition to the Real Time Recurrent Learning algorithm (Williams and Zipser, 1989) for updating the weights of the Pi-Sigma unit in the network. A standard error measure used for training the network is the Sum Squared Error:

$$E(n) = \frac{1}{2}\sum e(n)^2 \tag{4.3}$$

The error between the target and forecast signal is determined as follows:

$$e(n) = d(n) - y(n) \tag{4.4}$$

where $d(n)$ is the target output at time n, $y(n)$ is the forecast output at time n.

At every time n, the weights are updated according to:

$$\Delta W_{kl}(n) = -\eta\left(\frac{\partial E(n)}{\partial W_{kl}}\right) \tag{4.5}$$

where η is the learning rate.

The value $\left(\frac{\partial E(n)}{\partial W_{kl}}\right)$ is determined as:

$$\frac{\partial E(n)}{\partial W_{kl}} = -e(n)\frac{\partial y(n)}{\partial W_{kl}} \tag{4.6}$$

$$\frac{\partial y(n)}{\partial W_{kl}} = \frac{\partial y(n)}{\partial P_i(n)}\frac{\partial P_i(n)}{\partial W_{kl}} \tag{4.7}$$

where

$$\frac{\partial y(n)}{\partial P_i(n)} = f'\left(\sum_{i=1}^{k} P_i(n)\right)\left(\prod_{\substack{j=1 \\ j\neq i}}^{i} h_j(n)\right) \tag{4.8}$$

and

$$\frac{\partial P_i(n)}{\partial W_{kl}} = W_{ij}\frac{\partial y(n-1)}{W_{kl}} + Z_i(n)\,\delta_{ik} \tag{4.9}$$

where δ_{ik} is the Krocnoker delta. Assume D as the dynamic system variable (the state of the ij^{th} neuron), where D is:

$$D_{ij}(n) = \frac{\partial y(n)}{\partial W_{kl}} \tag{4.10}$$

The state of a dynamical system is formally defined as a set of quantities that summarizes all the information about the past behaviour of the system that is needed to uniquely describe its future behaviour (Haykin, 1999). Substituting Equation (4.8) and (4.9) into (4.7) results in:

$$D_{ij}(n) = \frac{\partial y(n)}{\partial W_{kl}} = f'\left(\sum_{i=1}^{k} P_i(n)\right) \times \left(\prod_{\substack{j=1 \\ j\neq i}}^{i} h_j(n)\right)\left(W_{ij}D_{ij}(n-1) + Z_i(n)\,\delta_{ik}\right) \tag{4.11}$$

where $f'(.)$ is the first derivative of a nonlinear activation function.

For simplification, the initial values for $D_{ij}(n\text{-}1)=0$, and $Z_j(n\text{-}1)=0.5$. Then the weights updating rule is

$$\Delta W_{ij}(n) = \eta e(n) D_{ij}(n) + \alpha \Delta W_{ij}(n-1)$$
$$W_{ij}(n+1) = W_{ij}(n) + \Delta W_{ij}(n) \tag{4.12}$$

where W_{ij} are adjustable weights and ΔW_{ij} are total of weight changes.

4.4 Issues of Stability in DRPNN

While Recurrent Neural Networks have matured into a fundamental tool for solving many real world problems such as time series forecasting, approximating a dynamical system, forecasting a stream flow, string classification, character recognition, and system control, major difficulties for their application still remain. These are the known high numerical complexity of the training algorithm and the difficulties in assuring stability (Steil, 2005). In RNNs, the internal state evolves in time according to certain nonlinear state equations until it goes to equilibrium, or possibly other types of behaviour such as periodic or chaotic motion could occur (Atiya, 1988). However, one would be interested in having a steady and fixed output for every input applied to the network. Therefore, beginning in any initial condition, the state should ultimately go to a unique equilibrium. It is in fact that equilibrium state that determines the final output. The objective of the learning algorithm is to adjust the parameters of the network into small steps in order to move the unique equilibrium state in a way that will result finally in an output as close as possible to the required one. Since weight adjustment affects the evolution of states at every time steps during the network training, obtaining the error gradient is rather a complicated procedure (Atiya, 2000). This is due to the tendency of the network to become unstable.

One of the most useful properties of networks with recurrent connection is their ability to model the behaviour of arbitrary dynamical system. Hence, the existence of feedback in the proposed DRPNN is expected to improve the performance of a given network. Despite the potential and capability of the DRPNN which comprises the recurrent connection, the same problems of complexity and difficulty of training the network exist in the proposed DRPNN, which are:

- The states of the processing elements, denoted by D_{ij} in Equation 4.10, affect both the output and the gradient. Therefore, calculating the gradients and updating the weights of a recurrent network is much more difficult.
- The network is more difficult to train than ordinary RPNN. This relates to the fact that the training algorithm could become unstable which is the result of:
  - the error between the target and the output of the DRPNN may not be monotonically decreasing,
  - the gradient computation is more complicated,
  - and the convergence time may be long.

In an attempt to overcome the stability and convergence problems in the proposed DRPNN, the convergence of DRPNN is presented in section 4.5 to ensure that the network posses a unique equilibrium state.

4.5 The Stability Condition for DRPNN

Based on the stability theorem for a general network proposed by (Atiya, 1988) and shown in Equation 4.13, any network that satisfies this theorem exhibits no other behaviour except going to a unique equilibrium for a given input:

$$\sum_{i=1}\sum_{j=1} w^2_{ij} < \frac{1}{max(f')^2} \tag{4.13}$$

where w is the weight matrix and f' is the first derivative of a bounded and differentiable activation function.

From the given theorem, a unique fixedpoint is reached regardless of the initial condition. This means that for a given input, after a short transient period, the network will give a steady and fixed output, no matter what the initial network state was. In other words, beginning with any initial conditions, the state is to be attracted towards a unique equilibrium. In order to guarantee that the proposed DRPNN shows a unique equilibrium state, a derivation of the stability convergence of the proposed network will be presented.

Let $y_1(t+1)$ and $y_2(t+1)$ be 2 outputs for the DRPNN.

$$y_1(t+1) = f\left(\sum_{k=1}^{A}\prod_{L=1}^{k} h_{1L}(t+1)\right) \tag{4.14}$$

where f is a nonlinear transfer function.

$$y_2(t+1) = f\left(\sum_{k=1}^{A}\prod_{L=1}^{k} h_{2L}(t+1)\right) \tag{4.15}$$

$$\begin{aligned} h_{1L}(t+1) &= \sum_{i=1}^{M} W_{Li}X_i + W_{L(M+1)} + W_{L(M+2)}y_1(t) \\ &= \alpha_L + \beta_L y_1(t) \end{aligned} \tag{4.16}$$

with

$$\alpha_L = \sum_{i=1}^{M} W_{Li}X_i + W_{L(M+1)} \tag{4.17}$$

and

$$\beta_L = W_{L(M+2)} \tag{4.18}$$

while

$$\begin{aligned} h_{2L}(t+1) &= \sum_{i=1}^{M} W_{Li}X_i + W_{L(M+1)} + W_{L(M+2)}y_2(t) \\ &= \alpha_L + \beta_L y_2(t) \end{aligned} \tag{4.19}$$

The aim is to get J approaching '0', which means that the 2 outputs of a given input are close.

Let $J(t+1)$ be:

$$J(t+1) = \left\| y_1(t+1) - y_2(t+1) \right\| \tag{4.20}$$

where $\| \quad \|$ is the norm. Based on Mean Value Theorem (O'Connor and Robertson, 2000), which states that for a function $f(x)$ which is continuous on the closed interval $[a,b]$ and differentiable on the open interval (a,b), there exists a value c on the interval (a,b) such that

$$f'(c) = \frac{f(b) - f(a)}{b - a} \tag{4.21}$$

where f' is the derivation of the function. Hence

$$f(b)-f(a)=f'(c)\bullet(b-a) \tag{4.22}$$

and

$$\|f(b)-f(a)\|=\|f'(c)\|\bullet\|b-a\| \tag{4.23}$$

which leads to

$$\|f(b)-f(a)\|\leq max\|f'(c)\|\bullet\|b-a\| \tag{4.24}$$

substituting Equation (4.14) and (4.15) into Equation (4.20), results into

$$J(t+1)=\left\|f\left(\sum_{k=1}^{A}\prod_{L=1}^{k}h_{1L}(t+1)\right)-f\left(\sum_{k=1}^{A}\prod_{L=1}^{k}h_{2L}(t+1)\right)\right\| \tag{4.25}$$

using Mean Value Theorem, leads to

$$\begin{aligned}&\left\|f\left(\sum_{k=1}^{A}\prod_{L=1}^{k}h_{1L}(t+1)\right)-f\left(\sum_{k=1}^{A}\prod_{L=1}^{k}h_{2L}(t+1)\right)\right\|\\&\leq max|f'|\bullet\left\|\sum_{k=1}^{A}\prod_{L=1}^{k}h_{1L}(t+1)-\sum_{k=1}^{A}\prod_{L=1}^{k}h_{2L}(t+1)\right\|\end{aligned} \tag{4.26}$$

therefore, from Equation (4.25), Equation (4.26) becomes

$$J(t+1)\leq\max|f'|\bullet\left\|\sum_{k=1}^{A}\prod_{L=1}^{k}h_{1L}(t+1)-\sum_{k=1}^{A}\prod_{L=1}^{k}h_{2L}(t+1)\right\| \tag{4.27}$$

from Equation (4.14) & (4.16), let $g(y)$ be

$$\begin{aligned}g(y)&=\sum_{k=1}^{A}\prod_{L=1}^{k}(\alpha_L+\beta_L y)\\&=\sum_{k=1}^{A}\prod_{L=1}^{k}h_L(t+1)\end{aligned} \tag{4.28}$$

hence

$$\left\|\sum_{k=1}^{A}\prod_{L=1}^{k}h_{1L}(t+1)-\sum_{k=1}^{A}\prod_{L=1}^{k}h_{2L}(t+1)\right\|=\|g(y_1(t))-g(y_2(t))\| \tag{4.29}$$

using the Mean Value Theorem again, leads to:

$$\|g(y_1(t))-g(y_2(t))\|\leq\max|g'|\bullet\|y_1(t)-y_2(t)\| \tag{4.30}$$

hence, from Equation (4.27),(4.29) & (4.30), results into:

$$J(t+1)\leq(\max|f'|)\bullet(\max|g'|)\bullet\|y_1(t)-y_2(t)\| \tag{4.31}$$

let δ be

$$\delta = \left(\max\left|f'\right|\right) \bullet \left(\max\left|g'\right|\right) \tag{4.32}$$

then

$$J(t+1) \leqslant \delta . \left\| y_1(t) - y_2(t) \right\| \tag{4.33}$$

from Equation (4.20), Equation (4.33) becomes

$$J(t+1) \leqslant \delta . J(t) \tag{4.34}$$

The aim is to get both *J(t+1)* and *J(t)* approaching very close to zero, and for large *(t),* and for any value of *(t).* To achieve this, δ has to be very small value, which is less than 1. Hence, from Equation (4.32), when δ is < 1, leads into:

$$\left(max\left|f'\right|\right) \bullet \left(max\left|g'\right|\right) < 1 \tag{4.35}$$

from Equation (4.28), *g(y)* will be

$$g(y) = \sum_{k=1}^{A} \prod_{L=1}^{k} (\alpha_L + \beta_L y) \tag{4.36}$$

let *P(y)* be

$$P(y) = \prod_{L=1}^{k} (\alpha_L + \beta_L y) \tag{4.37}$$

then

$$\sum_{k=1}^{A} P(y) = \sum_{k=1}^{A} \prod_{L=1}^{k} (\alpha_L + \beta_L y) \tag{4.38}$$

therefore

$$g(y) = \sum_{k=1}^{A} P(y) \tag{4.39}$$

and

$$\left(g(y)\right)' = \sum_{k=1}^{A} P'(y) \tag{4.40}$$

from Equation (4.37)

$$\ln\left(P(y)\right) = \sum_{L=1}^{k} \ln(\alpha_L + \beta_L y) \tag{4.41}$$

and

$$\frac{P'(y)}{P(y)} = \sum_{L=1}^{k} \frac{\beta_L}{(\alpha_L + \beta_L y)} \tag{4.42}$$

hence

$$P'(y) = P(y).\sum_{L=1}^{k} \frac{\beta_L}{(\alpha_L + \beta_L y)} \tag{4.43}$$

substitute Equation (4.37) into Equation (4.43), having

$$P'(y) = \prod_{L=1}^{k}(\alpha_L + \beta_L y) \bullet \sum_{L=1}^{k} \frac{\beta_L}{(\alpha_L + \beta_L y)} \tag{4.44}$$

then

$$P'(y) = \sum_{L=1}^{k} \beta_L \bullet \sum_{L=1}^{k} \frac{1}{(\alpha_L + \beta_L y)} \bullet \prod_{L=1}^{k}(\alpha_L + \beta_L y) \tag{4.45}$$

$$P'(y) = \sum_{L=1}^{k} \beta_L \bullet \sum_{L=1}^{k} \prod_{\substack{S=1 \\ S \neq L}}^{k}(\alpha_S + \beta_S y) \tag{4.46}$$

$$P'(y) = \sum_{L=1}^{k} \beta_L \bullet \prod_{\substack{S=1 \\ S \neq L}}^{k}(\alpha_S + \beta_S y) \tag{4.47}$$

substituting Equation (4.47) into Equation (4.40), results into

$$(g(y))' = \sum_{k=1}^{A} \sum_{L=1}^{k} \beta_L \bullet \prod_{\substack{S=1 \\ S \neq L}}^{k}(\alpha_S + \beta_S y) \tag{4.48}$$

and

$$\left|(g(y))'\right| = \left|\sum_{k=1}^{A} \sum_{L=1}^{k} \beta_L \bullet \prod_{\substack{S=1 \\ S \neq L}}^{k}(\alpha_S + \beta_S y)\right| \tag{4.49}$$

therefore

$$\left|\sum_{k=1}^{A} \sum_{L=1}^{k} \beta_L \bullet \prod_{\substack{S=1 \\ S \neq L}}^{k}(\alpha_S + \beta_S y)\right| = \left|\sum_{k=1}^{A} \sum_{L=1}^{k} \beta_L\right| \bullet \left|\prod_{\substack{S=1 \\ S \neq L}}^{k}(\alpha_S + \beta_S y)\right| \tag{4.50}$$

hence

$$\left|\sum_{k=1}^{A}\sum_{L=1}^{k}\beta_L \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}(\alpha_S+\beta_S y)\right| \leq \sum_{k=1}^{A}\sum_{L=1}^{k}|\beta_L| \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}(|\alpha_S|+|\beta_S y|) \tag{4.51}$$

note that from Equation (4.17)

$$\alpha_L = \sum_{i=1}^{M} W_{Li}X_i + W_{L(M+1)} \tag{4.52}$$

therefore

$$|\alpha_L| = \sum_{m=1}^{M+1}|W_{Lm}| \tag{4.53}$$

note that from Equation (4.18) and Equation (4.51) results

$$\left|\sum_{k=1}^{A}\sum_{L=1}^{k}\beta_L \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}(\alpha_S+\beta_S y)\right| \leq \sum_{k=1}^{A}\sum_{L=1}^{k}|W_{L(M+2)}| \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}\left(\sum_{m=1}^{M+1}|W_{Sm}|+|W_{S(M+2)}|\right) \tag{4.54}$$

hence

$$\left|\sum_{k=1}^{A}\sum_{L=1}^{k}\beta_L \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}(\alpha_S+\beta_S y)\right| \leq \sum_{k=1}^{A}\sum_{L=1}^{k}|W_{L(M+2)}| \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}\sum_{m=1}^{M+2}|W_{Sm}| \tag{4.55}$$

therefore, from Equation (4.49) and (4.55) results into

$$\left|(g(y))'\right| = \sum_{k=1}^{A}\sum_{L=1}^{k}|W_{L(M+2)}| \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}\sum_{m=1}^{M+2}|W_{Sm}| \tag{4.56}$$

substituting Equation (4.56) into Equation (4.35), we get

$$(\max|f'|) \bullet \left(\max\left(\sum_{k=1}^{A}\sum_{L=1}^{k}|W_{L(M+2)}| \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}\sum_{m=1}^{M+2}|W_{Sm}|\right)\right) < 1 \tag{4.57}$$

therefore, the condition for DRPNN to converge is described by

$$\left(\max\left(\sum_{k=1}^{A}\sum_{L=1}^{k}|W_{L(M+2)}| \bullet \prod_{\substack{S=1 \\ S\neq L}}^{k}\sum_{m=1}^{M+2}|W_{Sm}|\right)\right) < \frac{1}{(\max|f'|)} \tag{4.58}$$

This work guarantees the stability of DRPNN for the equilibrium problem. The resulting condition will further be applied in the network training, which will be discussed later in Chapter 6.

4.6 Chapter Summary

In this chapter the Dynamic Ridge Polynomial Neural Network was presented as an extension of the ordinary feedforward Ridge Polynomial Neural Network. In order to represent a dynamic system, the functionality and architecture of the feedforward RPNN were extended by adding a feedback connection into the network. Subsequently, the stability and the convergence of the proposed network were implemented to ensure having steady and fixed output.

CHAPTER 5: FINANCIAL TIME SERIES FORECASTING

5.1 Introduction

Forecasting a time series is a common problem in many domains of science, and this has been addressed for a long time by scientists (Senjyu, 2002; Zumbach, 2001; Masters, 1993). This chapter aims at introducing the fundamentals of financial time series prediction, addressing the difficulties and comparing neural networks and traditional forecasting approaches, particularly to the prediction of financial market. A review on literature detailing the practical applications of neural networks in financial time series prediction is presented.

5.2 Time Series and their Properties

Time series generally refers to a sequence of data points, of any data series measured typically at successive times, spaced at time intervals. Practically, it is a collection of historical data of one system, such as a stock price, traffic data, and the pollution rates. A time series can be used in two ways for different purposes:

- Looking backward – the use of historical data to analyze the previous behaviour of a system. Applications include diagnosis or recognition of machine fault or human disease.
- Looking forward – the use of data to predict or forecast the future behaviour of a system. Applications include stock or price prediction and market demand forecast.

Time series analysis comprises methods that attempt to understand the behaviour of such time series, often either to understand the underlying theory of the data points, or to make forecasts. ***Time series forecasting*** is the use of a model to predict future events or future data points based on known past events. It is a process that produces a set of outputs by a given set of historical variables. Forecasting assumes that future

occurrences are based on present or past events, in which some aspects of the past patterns will continue into the future. Past relationship can then be discovered through study and observation. In other words, time series forecasting is to discover the relationship between present, past and future observations. According to Plummer (2000), the aim of time series forecasting is to observe or model the existing data series which can be in the form of financial data series (stocks, indices, exchange rates, etc), physically observed data series (sunspots, weather, etc), and mathematical data series (Fibonacci sequence, integrals of differential equations, etc).

Time series forecasting takes an existing series of data X_{t-n}, ..., X_{t-2}, X_{t-1}, X_t and forecasts X_{t+1}, X_{t+2},..... data values. Theoretically, these series can be seen as a continuous function of time variable t. For practical purposes, however, time is usually viewed in terms of discrete time steps. The size of the time interval depends on the problem at hand, and can be anything from milliseconds, hours to days, or even years. If the time series contain only one component, it is called a univariate time series; otherwise it is a multivariate time series. In a univariate series, the input variables are restricted to the signal being predicted, while in multivariate series, the raw data come from a variety of indicators which will form the actual inputs variables (Kaastra and Boyd, 1996). In a multivariate series, any indicator whether or not it is directly related to the output can be incorporated as the input variable (Cao and Tay, 2003).

5.3 Financial Time Series

Financial time series are among the best application domains for intelligent processing and advanced learning techniques (Magdon-Ismail et al., 1998). Financial time series has an economic interest to predict the financial value at some time in the future. This is because once the prediction of returns is successful, monetary rewards will be substantial.

Financial time series are available in different time scales; daily, hourly, or tick-by-tick stock prices of exchange rates, and they are simultaneously available in many different markets. The distances between point to point in the financial time series depend on the activity in the market. If there is a lot of action in the market, the price changes more often and frequently than during quite periods. These stock market fluctuations are the result of complex phenomena and their affect are translated into a blend of signs and losses that appear in stock price time series plot (Sitte and Sitte, 2000). The most noticeable variations are: trend, periodic variations and day-to-day variations. The trend is an identifiable long term variation in the stock market time series, while the periodic variations follow either seasonal patterns or the business cycle in the economy. Short-term and day-to-day variations appear at random and are difficult to predict, but they are often the source for stock trading gains and losses, especially in the case of day traders (Sitte and Sitte, 2000).

Financial time series data can be categorized into three categories; technical data, fundamental data, and derived entities (Hellstrom and Holmstrom, 1998):

5.3.1 Technical data

As reviewed by Kaastra and Boyd (1996), technical data are defined as lagged values of the dependent variable. The term 'lagged' means an element of the time series in the past. For example, at time *t*, the values *y(t-1), y(t-2),..,y(t-p)* are said to be lagged values of the time series *y*. Meanwhile, the term 'dependent variable' signifies the variable whose behaviour is being predicted. Technical data includes figures such as stock prices, volume, volatility, and etc. Typical types of daily technical data are as follows (Hellstrom and Holmstrom, 1998):

- Closing price (price of the last performed trade during the day)
- Highest traded price during the day
- Lowest traded price during the day
- Volume (total number of traded stock during the day)

5.3.2 Fundamental data

Fundamental data are economic variables which are believed to influence the dependent variable (Kaastra and Boyd, 1996). These are data describing current economic activity of the company whose stock prices are to be predicted. Fundamental data include information about current market situation as well as macroeconomic parameters, such as inflation, unemployment rate, and etc (Hellstrom and Holmstrom, 1998).

5.3.3 Derived Entities

Derived entities are produced by transforming and combining technical and fundamental data such as the following (Hellstrom and Holmstrom, 1998; Yao and Tan, 2001):

- The *k*-step returns which can be interpreted as the *k*-day price trend for the stock:

$$R(t) = \frac{y(t) - y(t-k)}{y(t-k)}$$

- The log-return

$$R(t) = log \frac{y(t)}{y(t-1)}$$

where $y(t)$ is the price or value at time t.

In most cases, the most obvious types of data selected to predict time series is the returns (Chenoweth and Obradovic, 1995; Franses, 1998). Hellstrom and Holmstrom in their work (1998) argued that both of these derived entities are often used for financial time series prediction for the following reasons:

- $R(t)$ has a relative constant range even if data for many years are used as inputs. The raw prices normally fluctuate very much and make it difficult to create a valid model for a longer period of time. Returns also fluctuate slightly, but they do so within approximately the same boundaries, and remain on the same scale.

- *R(t)* for different stocks may also be compared on an equal basis.
- It is easy to evaluate prediction accuracy by computing the correct sign prediction of *R(t)*.

5.4 The Prediction of Financial Time Series

The Prediction of financial time series is very difficult and a nontrivial problem since it depends on several known and unknown factors, and frequently data used for the prediction is noisy, uncertain and incomplete. The series are affected by many highly correlated economic, political and even psychological factors. Several difficulties can arise when handling time series forecasting, and as a result it has been suggested that some financial time series are not predictable (Schwaerzel, 1996).

The prediction of financial time series attracts interest due to its difficulty in practical application. They have a number of properties, which make the prediction challenging. Depending on the type of data series, a particular difficulty may or may not exist; among them are (Plummer, 2000):

1. Limited quantity of data.

   Limited data may be the most difficult problem in financial time series prediction. In order to form a more accurate model, it is desirable to use as large training set as possible. However, such large data is not always possible.

2. Noisy behaviour.

   Financial data are characterized as noisy data in which large amount of random and unpredictable day-to-day variations exist.

3. Non-stationary properties.

   Most financial data is non-stationary in nature, meaning that the statistical properties (e.g. mean and variance) of the data change over time. These changes are caused as a result of various business and economic cycles.

4. Outliers data.

   Values that do not appear to be consistent with the rest of the data set. They are correct but extremely unusual observations.

5. Random events.

   Many random and unpredictable events will occur which will affect the time series that need to be predicted. When major events occur, the fluctuation of the time series will increase for a short period. Meanwhile, small events will not affect the variable in a recognizable way, rather they will become part of the background noise of the signal which makes it more difficult to extract meaningful information. Such events come in two categories (Knowles , 2005):

   - Economic news such as announcements of interest rate changes, unemployment figures, takeovers, etc.
   - Non-economic events, which still have effects on the economic climate, such as elections, natural disasters, terrorist acts, etc.

Financial time series prediction is an interesting problem to traders and individuals. Researchers and practitioners have been striving for an explanation of the movement of financial time series. To maximize profits from the liquidity market, forecasting techniques have been used by different traders. Assisted by powerful computer technologies, traders no longer rely on a single technique to provide information about the future of the market. Thus, various kinds of forecasting methods have been developed by many researchers and experts (Yao and Tan, 2000). From statistical to artificial intelligence, there are various choices of techniques which can be used to make a forecast. The traditional methods for financial time series forecasting are based around statistical approaches. Nevertheless, none of these methods are completely satisfactory due to the nonlinear nature of most of the financial time series (Hussain et al., 2006-a). Other more advanced techniques such as Support Vector Machine (Cao and Tay, 2003), genetic algorithm (Zumbach, 2001; Thomas and Sycara, 1999; Allen and Karjalainen, 1999; Dunis et al., 1999), fuzzy logic (Abraham et al., 2001), and neural networks have been used for financial time series prediction.

Time series forecasting is perhaps the most useful and exciting application of neural networks. The objective is to discover the underlying structure of the mechanism generating the data and to find an appropriate model which can simulate the data

generation process. Figure 5.1 gives a basic architecture of a neural network as a time series predictor.

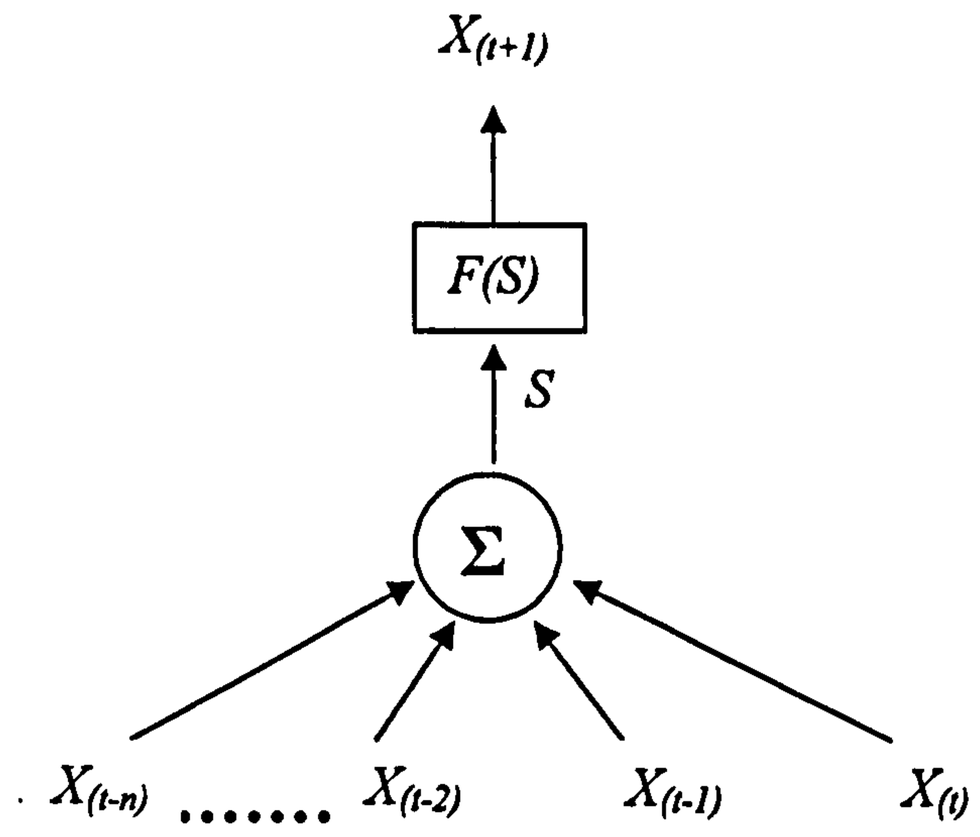


Figure 5.1: A general method of performing time series prediction with *n* inputs, where *F(S)* is the activation function, $X_{(t-n)}, \ldots, X_{(t-2)}, X_{(t-1)}, X_{(t)}$ are the input vectors, and *X(t+1)* is the predicted output.

Forecasting the behaviour of the financial market using neural networks is problematic. Multiple decisions, each of which affects the performance of the neural networks forecasting model, must be made, including which data to use, the size and the architecture of the neural network systems (Zhang, 2003). Some of the difficulties of using neural networks in financial time series applications are:

- There are infinitely many models which fit the training data well, but few of them generalize well. Supplementary degrees of freedom may lead to a better fitting of the model during the training of the network, but to worse generalization ability on the out-of-sample data (Lendasse et al., 2000).
- In order to form a more accurate model, it is desirable to use as large training set as possible. However, for the case of highly non-stationary data, increasing the size of training set results in more data with statistics that are less relevant to the task at hand being used in the creation of the model.
- The high noise and too many parameters (compared to the number of data available) make the models prone to overfitting (Lendasse et al., 2000; Dorffner, 1996).

- Require large number of sample data, due to their large number of free parameters (Dorffner, 1996). The limitation exists for the problems that some new founded companies do not have much of the previous data.

5.5 Conventional Prediction Methods and Neural Networks

Conventional statistical methods have been widely used to model the behaviour of financial time series and to forecast future values for time series (Dunis and Williams, 2002). Among them are Moving Average (MA), Auto Regressive model (AR), Auto Regressive Moving Average (ARMA) and exponential smoothing (Hussain et al., 2006-a). Although conventional statistical methods are perfect choice of modelling a lot of time series, they are not capable of modelling many time series generated by nonlinear systems, even if the underlying system is relatively simple (Gilde, 1996). These models are linear while most financial time series data show significant degrees of nonlinearity. Hence, they fail to capture the nonlinearities characteristic of financial time series.

Over the past few years, neural networks have been widely advocated as a new alternative modelling method to more traditional econometric and statistical approaches, claiming increasing success in the fields of economic and financial forecasting (Dunis and Huang, 2002). This has resulted in many publications comparing neural networks with traditional forecasting methods (Yumlu et al., 2005; Ho et al., 2002; Dunis and Williams, 2002; Dunis and Huang, 2002; Shachmurove and Witkowska, 2000; Yao and Tan, 2000; Yao et al., 1996; Moody, 1995; Kuan and Liu, 1995) and many more.

Kuan and Liu in their work (1995) showed that neural network models can describe in-sample data (training data) quite superior and that they also generate 'good' out-of-sample forecasts. Neural networks can tolerate noise and chaotic components better than most other methods (Masters, 1993). They promise attractive features to business forecasting, outperforming conventional statistical approaches (Yumlu et al., 2005; Gradojevic and Yang, 2000; Franses, 1998; Yao et al., 1996; Yao and Tan,

2000; Dunis and Huang, 2002; Ho et al., 2002). Because of the high volatility, complexity, nonlinearity, and noise market environment, neural network techniques usually are the prime candidates for prediction purposes when compete with statistical techniques (Leung et al., 2000; Dunis and Williams, 2002).

According to Refenes et al. (1994, cited by Yao and Tan, 2000), traditional statistical techniques used for forecasting financial time series have reached their limitation in applications where nonlinearities exist in the data set. The main attribute which distinguish neural networks time series modelling from conventional statistical methods is their ability to generate nonlinear relationship between a vector of time series input variable and a dependent series, with little or no priori knowledge about the nonlinearity in the series. This is opposed to the rigid structural form of most conventional series forecasting methods (Fieldsend and Singh, 2005).

Neural networks present a number of advantages over conventional methods of analysis, which are summarised as follows (Berardi, 2003; Garcia and Gencay, 2000; Hamm and Brorsen, 2000; Shachmurove and Witkowska, 2000; Zhang et al., 1998; Kuo and Reitsch, 1995):

- Neural networks make no assumptions about the nature of the distribution of the data and therefore they are not biased in their analysis. Instead of making assumptions about the underlying population, neural networks use the data to develop an internal representation of the relationship between the variables. Thus neural networks are well suited for problems whose solutions require knowledge that is difficult to specify but for which there are sufficient data or observations.
- Since financial time series data are dynamic in nature, it is necessary to have non-linear tools in order to discern relationship among time series data. Neural networks are capable of performing non-linear modelling, and they are best at discovering non-linear relationships.
- Neural networks perform well with missing or incomplete data. Traditional regression analyses are not adaptive; they process all past data together with new data. On the other hand, neural networks can adapt their weights when new input data becomes available.

- Neural networks can often correctly infer the unseen part of a population even if the sample data contain noisy information.
- It is relatively easy to obtain a forecast in a short period of time as compared with an econometric model.

5.6 Application of Neural Networks in Financial Time Series

The use of neural network models for the prediction of financial time series has shown significant improvements in terms of prediction and financial metrics (Cheng et al., 1996). This is not surprising since these models utilise more information such as inter market indicators, fundamental indicators and technical indicators. Furthermore, neural networks are capable of describing the dynamics of non-stationary time series due to their non-parametric, adaptive and noise tolerant properties (Cao and Tay, 2001).

A review on existing literature reveals financial studies on a wide variety of subjects such as stock price forecasting (Castiglione, 2000; Leung et al., 2000; Zekić, 1998), currency exchange rate forecasting (Hussain et al., 2006-b; Chen and Leung, 2005; Schwaerzel , 1996; Tenti, 1996; Kuan and Liu, 1995; Giles, et al., 2001; Yao et al., 1996; Walczak, 2001, Yao and Tan, 2000), returns prediction (Hussain et al., 2006-a; Shachmurove and Witkowska, 2000; Franses, 1998; Dunis and Williams, 2002; Chenoweth and Obradovic, 1995, Yao et al., 1996), predicting government treasury bond (Cheng et al., 1996), forecasting currency volatility (Yumlu et al., 2005; Dunis and Huang, 2002), sign prediction (Lendasse et al, 2000; Fernandez-Rodriguez et al., 2000), and others (Sitte and Sitte, 2000; Moody, 1995).

Yumlu et al. in their work (2005) have discussed the application of global, feedback and smoothed-piecewise neural prediction models for the Istanbul stock exchange. A conventional Exponential Generalized Autoregressive Conditional Heteroskedasticity (EGARCH) volatility model was implemented for comparison purpose. They observed that the smoothed-piecewise neural network model becomes advantageous in capturing volatility in index return series when compared to global

and feedback neural network models, and also to the conventional EGARCH volatility model.

Dunis and Huang in their work (2002) examined the use of non-parametric Neural Network Regression and Recurrent Neural Network regression models for forecasting and trading the currency volatility, with an application to the GBP/USD and USD/JPY exchange rates. Similarly, Dunis and Williams (2002) implemented Neural Network Regression to forecast foreign exchange rates on UER/USD series. The study was benchmarked against several traditional forecasting techniques including Naïve Strategy, MACD Strategy, ARMA Methodology, and Logit Estimation (Dunis and Williams, 2002). Their observations have confirmed the applicability of neural network for financial forecasting.

Another approach to financial time series forecasting can be found in Shachmurove and Witkowska's work (2000). The authors analyzed the predictability of major world stock markets of Canada, France, Germany, Japan, United Kingdom (UK), the United States (US), and the world excluding US (World) using Multilayer Perceptron models. They found that Multilayer Perceptron models predict daily stock returns better than the traditional ordinary least squares and general linear regression models, in terms of the Mean Squared Error.

The forecasting performance of the neural networks on the exchange rates between American Dollar and five other major currencies; Japanese Yen, Deutsch Mark, British Pound, Swiss Franc and Australian Dollar was reported by Yao and Tan (2000). The results showed that irrespective of Normalized Mean Squared Error, gradient or profit, the neural networks models used are much better than the traditional ARIMA model. They also concluded that a backpropagation network used in their study can achieve up to 73% of correctness in terms of gradients, when compared to the ARIMA method which achieve about 50% of correctness.

Gradojevic and Yang (2000) investigated whether introducing a market microstructure variable (that is, order flow) into a set of daily observations of microeconomic variables (interest rate, crude oil price) together with neural

network's technique can explain the Canada/U.S. dollar exchange rate movements better than linear and random walk models. They compared these models using Root Mean Squared Error and percentage of correctly predicted exchange rate changes. Empirical findings are in favour of the neural network model, which yields a very robust out-of-sample forecasting improvement in both performance measures.

Castiglione (2000) modelled the MLP to predict the price increment of several daily closing prices of different assets and indexes. The author found that the network has the potential to forecast the sign of the price increments with a success rate above 50%. Another study to evaluate and compare the performance of Multilayer Feedforward neural network and general regression neural network was carried out by Chen and Leung (2005). They measured the network's strength on the prediction of currency exchange correlation.

Chan et al. (2000) investigated financial time series forecasting using feedforward neural network and daily trade data from Shanghai Stock Exchange. To improve speed and convergence they used a conjugate gradient learning algorithm and multiple linear regressions for the weight initialization. They conclude that neural network can model the time series satisfactorily and that their learning and initialization approaches lead to improved learning and lower computation costs.

Using weekly data from 1984 to 1995, Yao et al. in their work (1996) have analysed the predictability of the British Pound, Deutsch Mark, Japanese Yen, Swiss France, and Australian Dollar against the US Dollar. They used the ARMA model as a benchmark. Correctness of sign and trading performance were used to evaluate the models. They concluded that using neural network models can produce higher correctness of sign, and consequently produce higher returns, than ARMA models. In addition, they state that without the use of extensive market data or knowledge, useful predictions can be made and significant profit can be achieved.

Neural networks are an emerging and challenging computational technology that can offer a new avenue to explore the dynamics of a variety of financial applications. They can make contributions to the maximization of returns, while reducing costs,

and limiting risks. Zekić in his work (1998) showed that neural networks have been widely used for many fields and applications, for example:

- Classification of stocks
- Recommendation for trading
- Predicting stock performance
- Predicting price changes of stock indexes
- Stock price prediction
- Modelling and forecasting the stock performance

5.7 Chapter Summary

One of the most challenging problems in economics is the forecasting of financial markets. Current research have shown that neural networks are promising tools for forecasting financial times series, as they were most implemented in mapping the underlying movement in the financial market. Numerous research and applications of neural networks in business have proven their advantage in relation to classical methods. Nevertheless, literatures on the use of Higher Order Neural Networks (HONNs) for financial time series prediction are limited. The following chapter focuses at the design and implementation of HONNs model, particularly for financial time series prediction.

CHAPTER 6: EXPERIMENTAL DESIGN

6.1 Introduction

The design of neural networks to successfully predict financial time series is a complex task. Several design factors can significantly impact the accuracy of network forecast, such as the selection of the input-output variables, the choice of data, the initial weight state, the stopping criterion during the training phase, and etc. Issues such as the learning parameters, the number of nodes and the activation function are also important. The aim of this chapter is to provide an overview of a step by step methodology to propose the design of neural networks for forecasting financial time series. These neural networks models include the Multilayer Perceptron (MLP), Functional Link Neural Network (FLNN), the Pi-Sigma Neural Network (PSNN), The Ridge Polynomial Neural Network (RPNN), and the proposed Dynamic Ridge Polynomial Neural Network (DRPNN). A method of designing the network forecasting models, as well as the generation of input-output pattern, the specification of parameters, and performance measures used is presented.

6.2 Variable Selection

Choosing a suitable forecasting horizon is the first step in financial forecasting. From the trading aspect, the forecasting horizon should be sufficiently long such that excessive transaction cost resulting from over-trading could be avoided (Cao and Tay, 2003). Meanwhile, from the prediction aspect, the forecasting horizon should be short enough as the persistence of financial time series is of limited duration. Thomason in his work (1999-a) suggested that a forecasting horizon of five days is a suitable choice for the daily data. Considering the trading and prediction aspects from both literatures, this research work consequently implements two forecast horizons; 1-day ahead, and 5-days ahead.

Success in designing a network for time series prediction depends on clear understanding of the problem. Knowing which input variables to be used in the network is important. Walczak in his work (2001) showed that the selection of input variables is a critical problem facing neural networks researchers and designers. Chenoweth and Obradovic (1995) suggested that an appropriate combination of most significant inputs leads to faster computation compared to the use of all available input variables. Yao and Tan in their work (2000) also argued that increasing the number of inputs does not necessarily increase the accuracy of time series prediction. This related to fact that the information provided by others inputs variables might already embed in the essential inputs variables. At this point of the design process, the concern is about the raw data from a variety of indicators, which will form the actual inputs to the network. These raw data, as discussed previously in Chapter 5, can be in the form of technical data (univariate signal), and fundamental data (multivariate signals). Univariate signals are data directly obtainable from the time series being forecast, and models that utilize univariate signals rely on the predictive capabilities of the time series itself. Meanwhile multivariate signals utilize information from outside the time series, in addition to the time series itself. For simplicity reason, this research work is restricted to the use of univariate signals.

6.3 Data Selection

To build a proper neural network forecasting model, sufficient experiments should be performed. To test the network capabilities only for one market or just for one particular time period will not promise an acceptable result. It will not lead to a robust model based on manually, trial and error, or adhoc experiments.

In this research work, ten noisy financial time series signals are considered as shown in Table 6.1. All financial time series were obtained from a historical database provided by Datastream® (2005), forepart from the IBM common stock closing price time series, which was taken from the Time Series Data Library (Hyndman, 2005). The signals were fed to the neural networks to capture the underlying rules of the movement in the financial markets.

Table 6.1: Financial time series signals used

	Time Series Data	Time Periods	Total
1	IBM common stock closing price (IBM)	17/05/1961 to 02/11/1962	360
2	Standard & Poor 500 stock index futures (CMESP)	01/01/1988 to 11/07/1995	1963
3	The United States 10-year government bond (CBT-10)	01/06/1989 to 11/12/1996	1965
4	The United States 30-year government bond (CBT-30)	01/10/1990 to 24/04/1998	1975
5	UK pound to EURO exchange rate (UK/EU)	03/01/2000 to 04/11/2005	1525
6	UK pound to US dollar exchange rate (UK/US)	03/01/2000 to 04/11/2005	1525
7	US dollar to EURO exchange rate (US/EU)	03/01/2000 to 04/11/2005	1525
8	Japanese yen to EURO exchange rate (JP/EU)	03/01/2000 to 04/11/2005	1525
9	The Japanese Yen to US dollar exchange rate (JP/US)	03/01/2000 to 04/11/2005	1525
10	The Japanese Yen to UK pound exchange rate (JP/UK)	03/01/2000 to 04/11/2005	1525

As can be noticed from Table 6.1, six time series used in this research work are the exchange rates signals. It is worth pointing out that most of the published research in financial time series prediction has focused on the exchange rate forecasting, for example (Hussain et al., 2006-b; Chen and Leung, 2005; Schwaerzel, 1996; Tenti, 1996; Kuan and Liu, 1995; Giles et al., 2001; Yao et al., 1996; Walczak, 2001; Yao and Tan, 2000). The foreign exchange market is the largest and most liquid of the financial market with an estimated $1 trillion traded everyday (Yao and Tan, 2000). Foreign exchange rates are among the most important economic indices in the international monetary markets. The trading of currencies has grown enormously due to the general trend of globalization, the increase of the import and export of commodities all over the world, and an increased interest in international investments (Schwaerzel, 1996).

The U.S. Government Bonds were also used in this research work because it is a significant variable in many econometric and financial models. According to Cheng et al. (1996), the U.S treasury market is the largest financial market in the world, with over 3 trillion dollars in securities traded on an around-the-clock basis. It yields the greatest return on investment, and it is a highly-liquid asset. In addition to the U.S. Government Bonds, The S&P 500 is widely regarded as the best single gauge of the U.S. equities market. The S&P 500 is an index containing the stocks of 500 leading companies in leading industries of the U.S. economy. The index is the most

notable of the many indices owned and maintained by Standard & Poor's, a division of McGraw-Hill. The IBM closing price, owned by the world's largest information technology company was selected as it is a well known time series, described by (Box et al., 1994).

6.4 Data Pre-processing

In this research work, two sets of experiments are performed, the non-stationary and the stationary data sets. For non-stationary signals, all the data listed in Table 6.1 is presented to the networks directly without any transformation. The data are scaled between the upper and lower bounds of the transfer function. On the other hand, the stationary version of the signals needs some series of transformations before passing them to the networks. It is worth noting that the precise values of daily prices (the non-stationary signals) are often not as meaningful to trading as its relative magnitude and the high-frequency component in financial data are often more difficult to be modelled (Cao and Tay, 2003).

The idea of transforming the original signal into the stationary version is due to the characteristics of the financial data which exhibit high volatility, complexity, and noise. Pre-processing and proper sampling of input data can give a significant impact on the forecasting performance (Kaastra and Boyd, 1996). To smooth out the noise and to reduce the trend, the original raw data was pre-processed into a stationary series (a shown in Figure 6.1: Part 1 and Part 2) by transforming them into measurements of relative different in percentage of price (RDP) (Thomason, 1999-a).

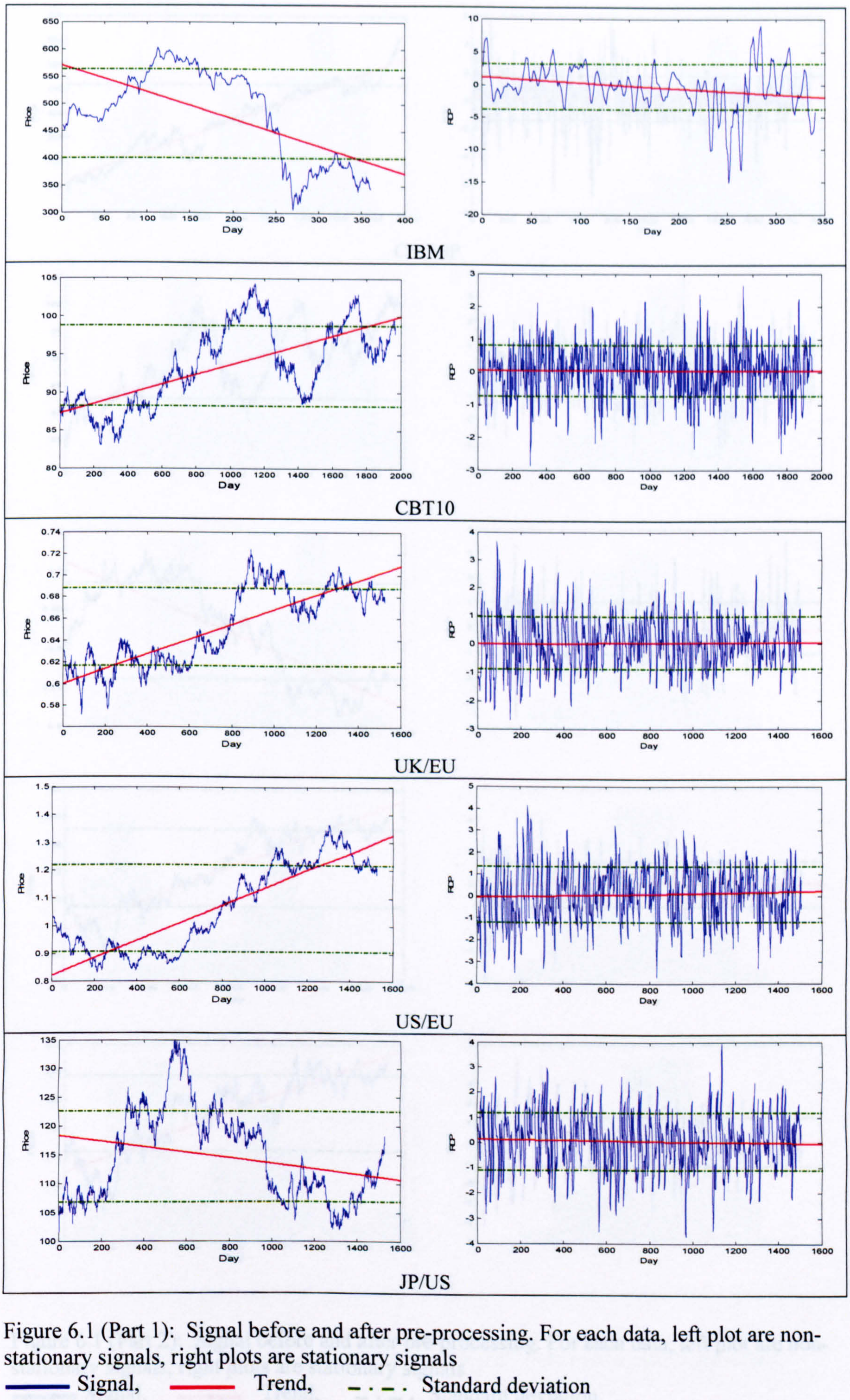


Figure 6.1 (Part 1): Signal before and after pre-processing. For each data, left plot are non-stationary signals, right plots are stationary signals

Signal, Trend, Standard deviation

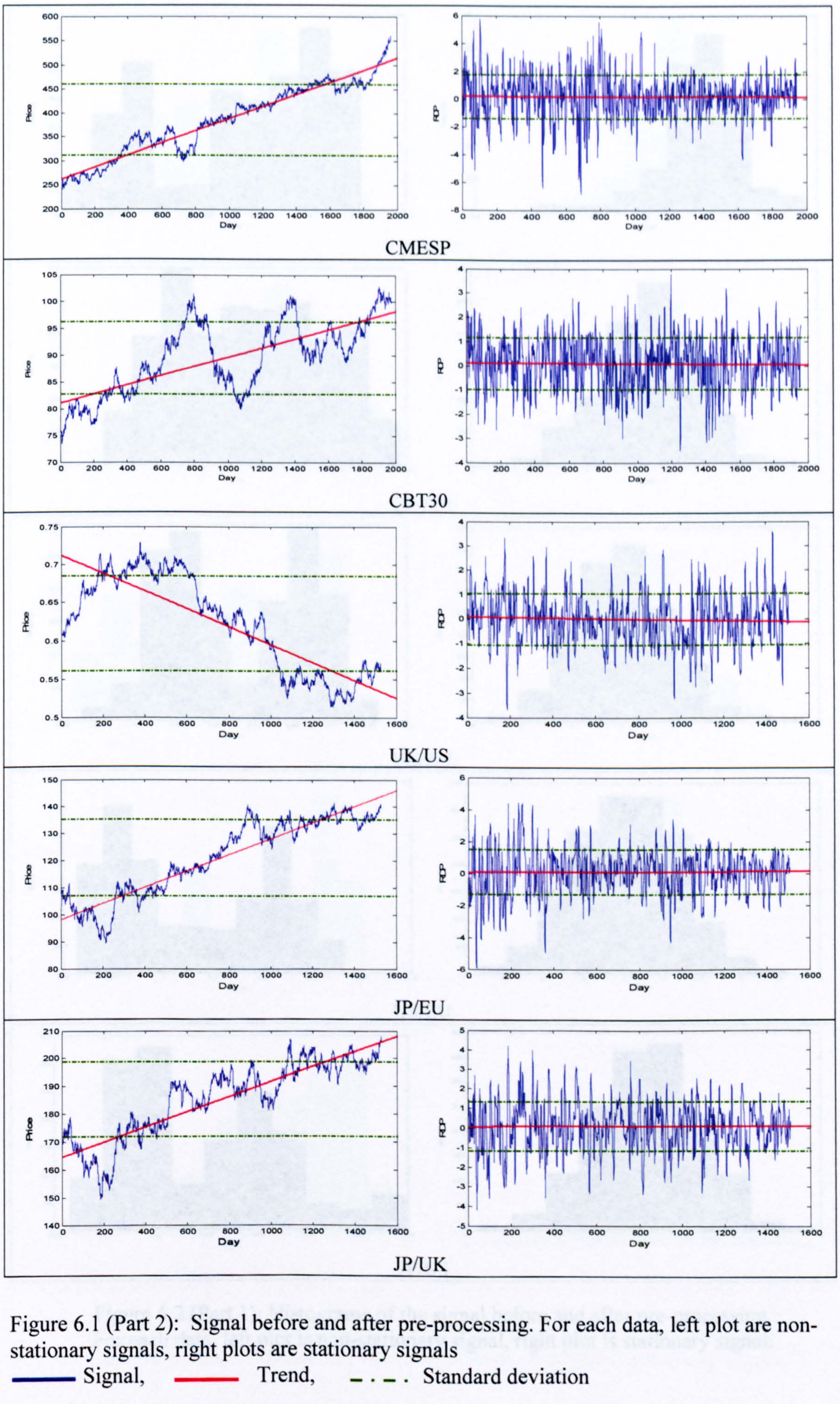


Figure 6.1 (Part 2): Signal before and after pre-processing. For each data, left plot are non-stationary signals, right plots are stationary signals

Signal, Trend, Standard deviation

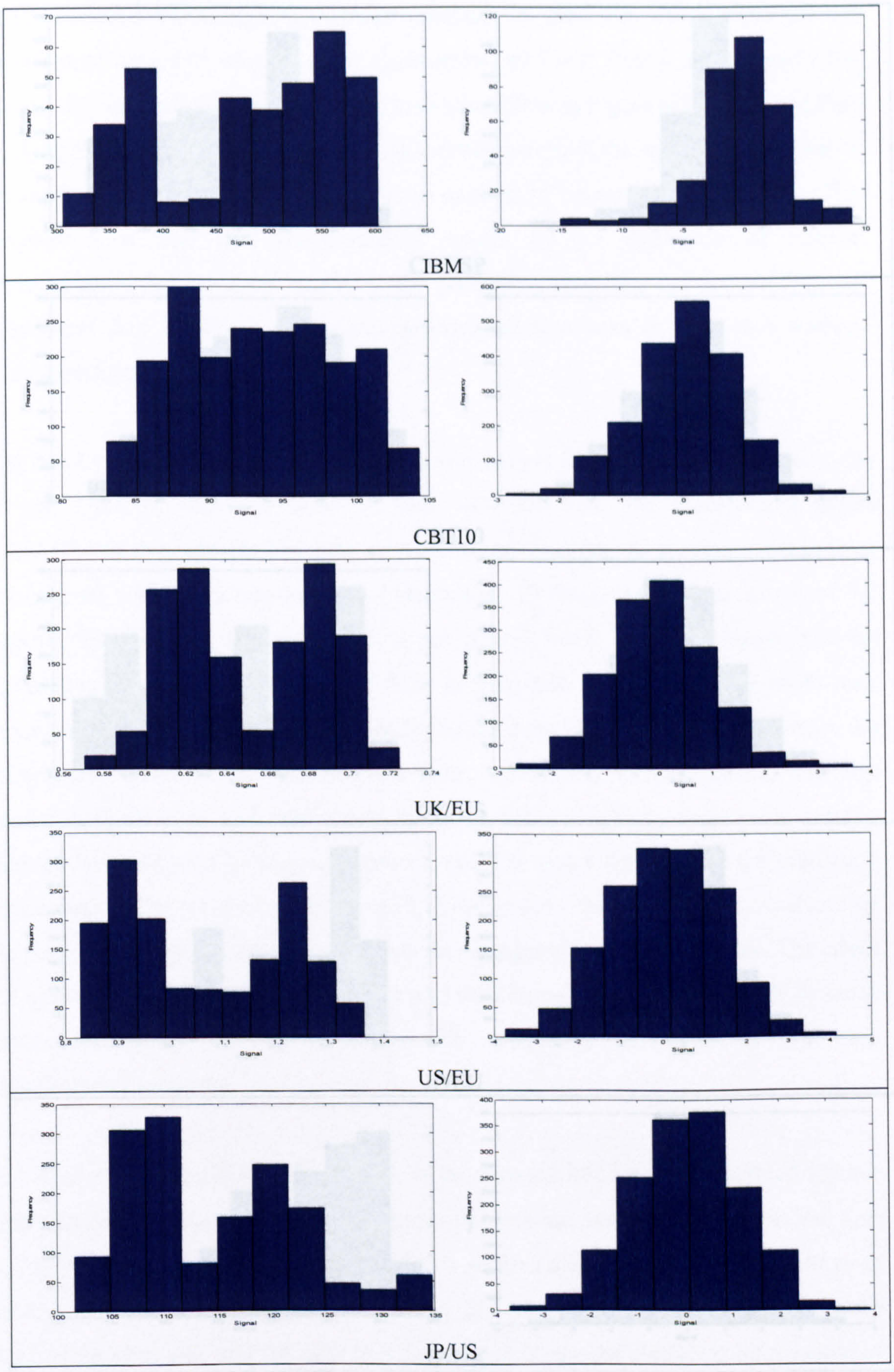


Figure 6.2 (Part 1): Histograms of the signal before and after pre-processing. For each data, left plot is non-stationary signal, right plot is stationary signal.

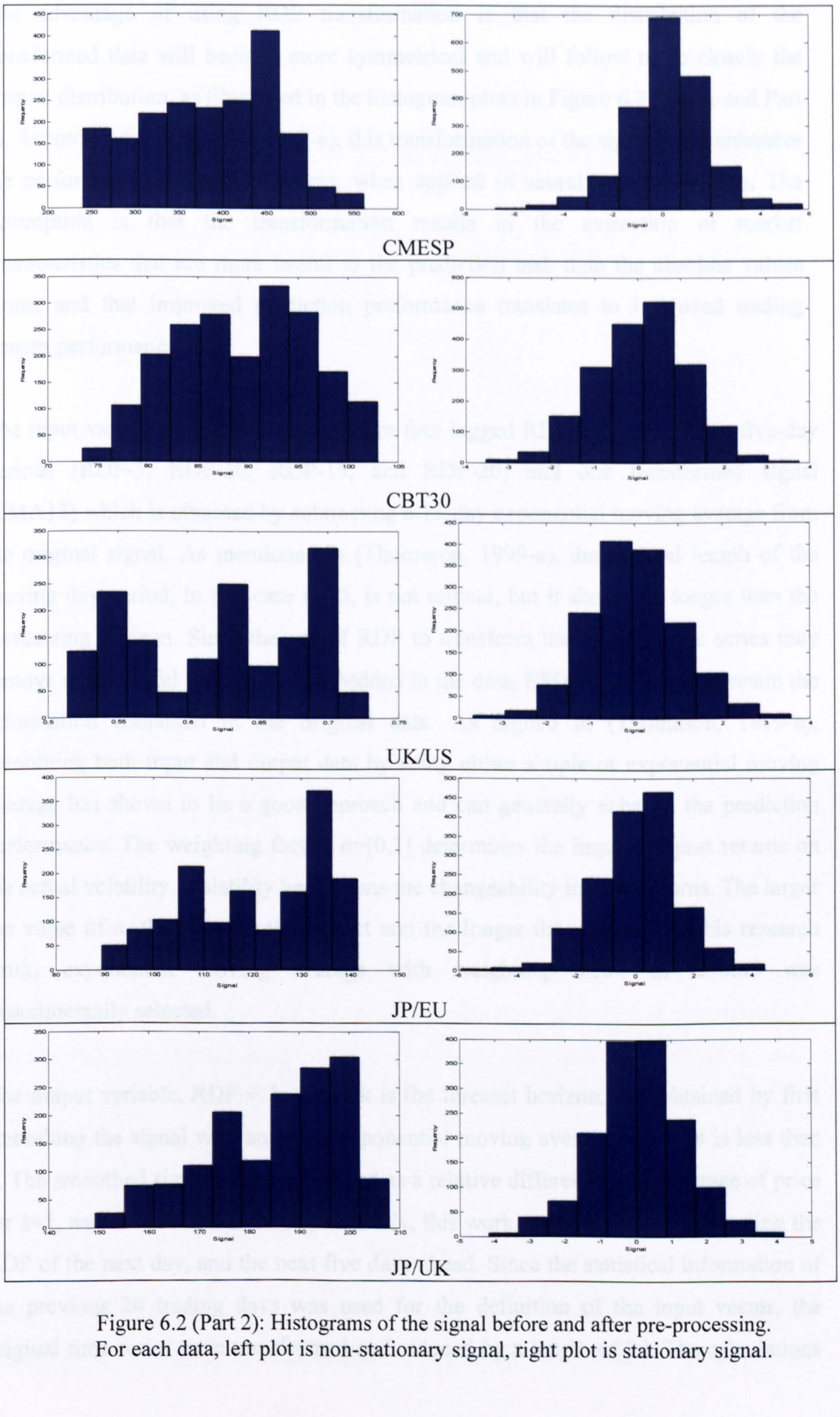


Figure 6.2 (Part 2): Histograms of the signal before and after pre-processing.
For each data, left plot is non-stationary signal, right plot is stationary signal.

The advantage of using RDP transformation is that the distribution of the transformed data will become more symmetrical and will follow more closely the normal distribution, as illustrated in the histogram plots in Figure 6.2 (Part 1 and Part 2). According to Thomason (1999-a), this transformation of the signal often enhances the performance of trading systems, when applied in neural network models. The assumption is that the transformation results in the extraction of market characteristics that are more useful to the prediction task than the absolute values alone, and that improved prediction performance translates to improved trading system performance.

The input variables were determined from four lagged RDP values based on five-day periods (RDP-5, RDP-10, RDP-15, and RDP-20) and one transformed signal (EMA15) which is obtained by subtracting a 15-day exponential moving average from the original signal. As mentioned in (Thomason, 1999-a), the optimal length of the moving day period, in this case is 15, is not critical, but it should be longer than the forecasting horizon. Since the use of RDP to transform the original time series may remove some useful information embedded in the data, EMA15 was used to retain the information contained in the original data. As argued in (Thomason, 1999-b), smoothing both input and output data by using either simple or exponential moving average has shown to be a good approach and can generally enhance the prediction performance. The weighting factor, α=[0,1] determines the impact of past returns on the actual volatility. Volatility here means the changeability in asset returns. The larger the value of α, the stronger the impact and the longer the memory. In this research work, exponential moving average with weighting factor of α=0.85 was experimentally selected.

The output variable, RDP + *k*, where k is the forecast horizon, was obtained by first smoothing the signal with an *n*-day exponential moving average, where *n* is less than 5. The smoothed signal is then presented as a relative difference in percentage of price for k=1, and k=5. As pointed out previously, this work concentrates on forecasting the RDP of the next day, and the next five days ahead. Since the statistical information of the previous 20 trading days was used for the definition of the input vector, the original time series were transformed and reduced by a length of 20. The calculations

for the transformation of input and output variables are presented in Table 6.2.

Table 6.2: Calculations for transformation of input and output variables

	Indicator	Calculations
Input variables	EMA15	$p(i) = EMA_{15}(i)$ $EMA_n(i) = \frac{\alpha^0 p_i + \alpha^1 p_{i-1} + \alpha^2 p_{i-2} + \ldots + \alpha^{n-1} p_{i-n+1}}{\alpha^0 + \alpha^1 + \alpha^2 + \ldots + \alpha^{n-1}}$
	RDP-5	$(p(i) - p(i-5)) / p(i-5) * 100$
	RDP-10	$(p(i) - p(i-10)) / p(i-10) * 100$
	RDP-15	$(p(i) - p(i-15)) / p(i-15) * 100$
	RDP-20	$(p(i) - p(i-20)) / p(i-20) * 100$
Output variable	RDP+k	$(p(i+k) - p(i)) / p(i) * 100$ $p(i) = EMA_3(i)$

where $EMA_n(i)$ is the n-day exponential moving average of the i-th day.
$p(i)$ is the signal of the i-th day.
α is weighting factor
k is forecast horizon; 1 or 5.

Subsequent to transformation, all the input and output variables were scaled in order to avoid computational problems and to meet algorithm requirements. The reasons for using data scaling is to reduce the range difference in the data and to process outliers, which consist of sample values that occur outside the normal (expected) range. Furthermore, the data are scaled to accommodate the limits of the network's transfer function. Manipulation of the data using this process produces a new bounded dataset. The calculation for the standard minimum and maximum normalization method is as follows:

$$x' = (max_2 - min_2) * \left(\frac{x - min_1}{max_1 - min_1} \right) + min_2 \qquad (6.1)$$

where x' refers to the normalized value, x refers to the observation value (original value), $\min_1$ and $\max_1$ are the respective minimum and maximum values of all

observations, and min_2 and max_2 refer to the desired minimum and maximum of the new scaled series. One of the desirable effects of minimum and maximum normalization is the preservation of the relationships of the original series (Thomason, 1998). In accordance to the selected Sigmoid transfer function, the input-output variables were normalized between the interval [0.2, 0.8]. The choice of the interval is to avoid difficulty in getting network outputs too close to the two endpoints of Sigmoid transfer function. This relates to the fact that the two endpoints can only be reached by infinite input values.

6.5 Data Partition

The data sets used in this research work were segregated in time order. In other words earlier period of data are used for training, and the data of the later period are used for testing. The main purpose of sorting them into this order is to discover the underlying structure or trend of the mechanism generating the data, that is to understand the relationship exist between the past, present and future data.

Table 6.3 (a): Data partition for stationary signals

Neural Networks	Data set	Prediction of				
		US/EU, UK/EU, UK/US, JP/EU, JP/US, JP/UK	IBM	CMESP	CBT10	CBT30
MLP FLNN PSNN	Training	753	170	971	973	977
	Validation	376	85	486	486	489
	Out-of-sample	376	85	486	486	489
RPNN DRPNN	Training	1129	255	1457	1459	1466
	Out-of-sample	376	85	486	486	489

Table 6.3 (b): Data partition for non-stationary signals

Neural Networks	Data set	Prediction of				
		US/EU, UK/EU, UK/US, JP/EU, JP/US, JP/UK	IBM	CMESP	CBT10	CBT30
MLP FLNN PSNN	Training	758	175	976	978	982
	Validation	379	88	489	489	492
	Out-of-sample	379	88	489	489	492
RPNN DRPNN	Training	1137	263	1465	1467	1474
	Out-of-sample	379	88	489	489	492

For the MLP, FLNN, and PSNN, each signal was divided into three data sets which are the training, validation and the out-of-sample which represent 25%, 25%, and

50% of the entire data, respectively. Out-of- sample data is the unseen data that has not yet being used during the training of the networks, and it is reserved for the use of testing of the networks. For the RPNN and DRPNN, the data were partitioned into two categories: the training and the out-of-sample data, with a distribution of 75% and 25%, respectively. Table 6.3 (a) and 6.3 (b) demonstrate the number of data points used for each data set, for both non-stationary and stationary versions.

6.6 Network Models Topology

Network models topology describes the architecture of the network models and the way in which the network is organized. This section addresses the selection of number of input-output nodes, network's order, hidden layer nodes, and the transfer function.

Number of input-output nodes

The number of nodes in the input layer is pre-determined by how many different input categories (independent variables) are used. Each of these input categories represents one input node. In real application, it is difficult to know how much information (in terms of number of variable or size of the input vector) must be used to properly learn the dynamics of the financial time series. Obviously, the quantity of the information increases with the number of the variables. However more input variables will lead to more parameters in the network, which increase the over-fitting problem (Lendasse et. al., 2000). As mentioned initially in Section 6.4, the number of input nodes for all networks used in this research work was set to 5.

On the other hand, the decision on the number of output nodes (dependent variables) is a straightforward task as it is directly related to the problem under study. For a time series forecasting problem, the number of output nodes often corresponds to the forecasting horizon. As mentioned initially, two types of forecasting horizon employed in this research work; the one-day-ahead and the five-days-ahead prediction.

Network's order for the HONNs

For FLNNs and PSNNs, the higher order terms were empirically selected between 2 and 5, whereas for RPNNs and DRPNNs, the network's order was incrementally grown from 1 to 5.

Number of hidden layers and hidden nodes for the MLPs

There is no perfect formula for determining the optimum number of hidden nodes. However a rough approximation can be obtained by the geometric pyramid rule proposed by Masters (1993). According to him, for a three-layer MLP with n input nodes and m output nodes, the hidden layer would have $\sqrt{nxm}$ nodes. The actual number of hidden nodes can still range from one-half to two times the geometric pyramid rule depending on the complexity of the problem. In practice, MLPs with one or two hidden layers were widely used and have performed very well in forecasting problems (Yao and Tan, 2001; Cybenko, 1989; Hornik et al., 1989; Kaastra and Boyd, 1996). In this research work, the MLPs were trained with one hidden layer, and the hidden nodes were experimentally varied from 3 to 8.

Transfer function

The purpose of transfer function is to determine the output of a processing neuron and to prevent outputs from reaching very large values which can 'paralyze' the network and thereby inhibit training (Kaastra and Boyd, 1996). The activation function is sometimes called a "transfer", and activation functions with a bounded range are often called "squashing" functions, such as the commonly used hyperbolic tangent and logistic sigmoid functions. It is commonly believed that the activity of biological neurons follows such sigmoid transfer function (Duch and Jankowski, 1999). In fact, better performance is often obtained when using sigmoid function at the output neurons which prevents the network to "overshoot" the correct output values (Thimm and Fiesler, 1997). The function is smooth and it is easy to calculate its derivative (Duch and Jankowski, 1999). The Sigmoid function is commonly used in time series prediction since they are nonlinear and continuously differentiable which are desirable properties for network learning (Kaastra and Boyd, 1996). Hence

in this research work, the sigmoid transfer function is used in all neural network models. The sigmoid function is calculated as follows:

$$f(x) = \frac{1}{\left(1 + \exp^{(-x)}\right)} \tag{6.2}$$

where x is neuron's output.

6.7 Training of the networks

One of the primary goals in training neural networks is to ensure that the network can perform well on data that it has not been used during the training previously, through the process of finding a set of optimal weights. Training is performed by repeatedly showing the network representative examples of the inputs, paired with the desired outputs. During each learning or training iteration, the magnitude of the error between the desired and actual outputs is computed. This is used to make adjustments to the internal network parameters or weights according to some specific learning algorithm.

The network is trained directly on the training set; a data set corresponding to a period much back in time. On the other hand, a data set corresponding to the most recent period of time was used for testing. Figure 6.3 and Figure 6.4 illustrate how the neural network is used to learn the financial time series using the non-stationary signal and stationary signal, respectively. The MLP, FLNN and PSNN were trained with the incremental backpropagation learning algorithm (Haykin, 1999), whereas the RPNN and DRPNN were trained with a constructive learning algorithm (Shin and Ghosh, 1995), as described in Chapters 2 and 3. In this research work, experimental simulations showed that longer training was not always giving better results. The reason is that continuing training for much longer epochs may produce improved training errors but at the same time it might give poorer forecasting results due to overfitting of data. It is anticipated that 3000 epochs is enough for the networks to converge. Therefore, all networks were trained with maximum number of 3000 epochs.

During the training process, the initial learning parameters such as the learning rate, momentum term, range of weights initialization, and the choice of stopping criteria used can highly affect the network learning speed and generalization performance (Thimm and Fiesler, 1994; Thimm and Fiesler, 1997). The optimal values for these parameters are usually unknown priori because they depend mainly on the training data set. Nevertheless, it is important to have a good approximation of the optimal initial value of these parameters as it can reduce the required training time. In this research work, a various sets of parameters (refer to Table 6.4) were experimentally chosen within the learning process to yield the best performance on out of sample data. The learning parameters for each network are based on the training algorithm used (as explained in Chapter 2 and Chapter3).

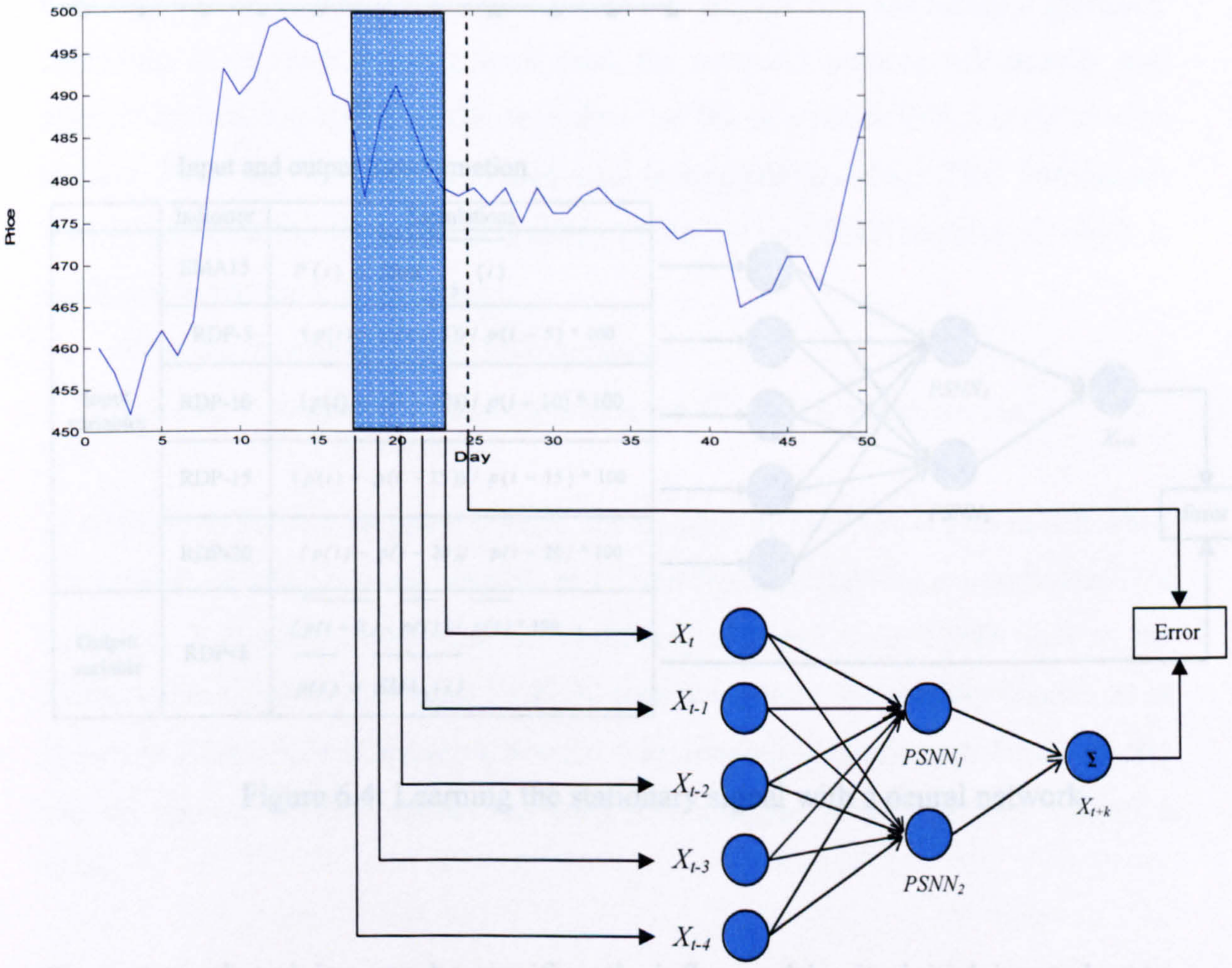


Figure 6.3: Learning the non-stationary signal with a neural network

Table 6.4: The learning parameters used in all neural networks

Neural Networks	Initial Weights	Momentum	Learning Rate (n)	dec_n	Threshold (r)	dec_r
MLP FLNN PSNN	[-0.5,0.5]	0.5	0.1 or 0.05	-	-	-
RPNN DRPNN	[-0.5,0.5]	0.5	[0.05, 0.5]	0.8	[0.00001, 0.7]	[0.05,0.2]

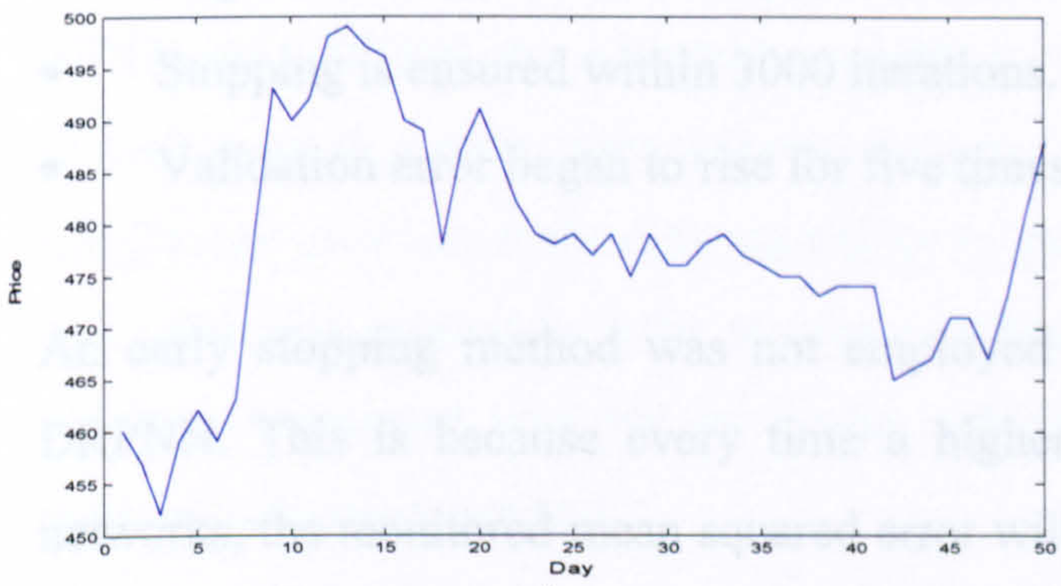


Figure 6.4: Learning the stationary signal with a neural network

Since network training can be significantly influenced by its initial internal state, which involves different initial learning parameters and different set of random weights, an average of 20 independent runs have been performed for all the neural networks in order to obtain fair and more robust comparative evaluation.

For every networks training, there will be a point at which training should be stopped. Early stopping criterion was utilized for the MLP, an FLNN, and PSNN. In this procedure, the networks were trained by observing the point at which the validation error began to rise, and then restored the network weights at the iteration cycle where the validation error was minimum. To avoid over-fitting and slow convergence of the training phase, the stopping criteria are determined by the following conditions, one of which is sufficient to end the training phase:

- Stopping is ensured within 3000 iterations.
- Validation error began to rise for five times continuously.

An early stopping method was not employed for the training of the RPNN and DRPNN. This is because every time a higher order PSNN unit is added to the networks, the monitored mean squared error will slightly increase before it gradually decreases. If an early stopping were used, the networks training will usually stop after a PSNN unit is added, at the same time that the new added PSNN is about to be trained. This will result in truncated and incomplete learning. Two termination criteria are used for stopping the training of RPNN and DRPNN, one of which is sufficient to end the training phase:

- Stopping is ensured within 3000 iterations.
- Training is stopped after accomplishing the 5^{th} order network learning.

In addition to the two stopping criteria, the training of DRPNN is also halted when the network learning become unstable and divert from the stability convergence. This condition is checked every time before adding a higher order PSNN unit to the networks. In other words, when DRPNN does not satisfy the stability condition, as shown in Equation 58 (Chapter 4, Section 4.5), training is terminated. This indicates that an optimal DRPNN model has been accomplished at the previous network's order.

6.8 Model Selection

How well the network generalized was deduced by analyzing its performance on the test set. Following the training of various architectures, a single model must be selected from all generated models for use in the final prediction (generalization). In accordance with the objective function (MSE), the model with lowest MSE on the validation set is routinely selected (for MLP, FSNN, and PLNN). In the case of RPNN and DRPNN, the adjustable weights in the final iteration during the training of each network's order were selected for generalization purpose. These set of tuned weights are used to test the network ability to generalize on out-of-sample data, using that particular network's order structure. When testing at this stage is completed, training is resumed with the increasing network's order, and this continues until stopping criteria is reached.

For all simulations which were carried on as part of this research work, the algorithms were written and run using Matlab version 7.0.4, on a machine with Windows XP 2000, Intel processor (Pentium 4), CPU of 3.00 GHz, and 1 GB of RAM.

6.9 Performance Metrics

There are a number of statistical measures that can be used in evaluating the performance of the neural networks, such as the Mean Squared Error (MSE), the Normalized Mean Squared Error (NMSE), and the Mean Absolute Error (MAE). For financial time series forecasting, the aim of the prediction is to achieve trading profits based on prediction results rather than emphasizing on the forecasting accuracy. As a result, financial criteria were frequently used to test whether the model is of economic value in practice (Dunis and Williams, 2002; Yao and Tan, 2000; Hamm and Brorsen, 2000; Cheng et al., 1996; Tenti, 1996).

In order to provide a more complete comparative evaluation, empirical testing was used in this work encompass not only on the more traditional criteria of MSE and

NMSE, but also a collection of analyses adapted in recent financial literature (Cao and Tay, 2003; Dunis and Williams, 2002; Hussain and Liatsis, 2002; Walczak, 2001; Yao and Tan, 2000; Fernandez-Rodriguez et al, 2000). The prediction performance of this work were measured using five financial metrics, and four statistical and signal processing metrics, as shown in Table 6.5. The objective of using financial metrics is to use the networks predictions to generate profit, whereas the statistical and signal processing metrics were used to provide accurate tracking of the signals. In order to measure profits generated from the networks predictions, a simple trading strategy was used. If the network predicts a positive change for the next *k*-day price (for non-stationary signal) or the next *k*-day RDP (for stationary signal), a 'buy' signal is sent, otherwise a 'sell' signal is sent. The descriptions for all the metrics used are given in the following subsections (Cao and Tay, 2003; Dunis and Williams, 2002; Hussain and Liatsis, 2002).

Annualized Return (AR)

The ability of the networks as traders was evaluated by the AR, a real trading measurement which is used to test the possible monetary gains and to measure the overall profitability in a year, through the use of the 'buy' and 'sell' signals (Dunis and Williams, 2002). The AR is a scaled calculation of the observed change in the time series value, when the sign of the change is correctly predicted.

Transaction Cost (TC)

Transaction cost is a penalty applied to the network each time a buy or sell signal is sent; as such actions would have a financial cost in the real world.

Maximum Drawdown (MDD)

MDD is the minimum of the accumulated losses and is used as a risk assessment measure for various financial prediction models. It measures the downside risk, which is the maximum loss of the model during the sample period.

Annualized Volatility

Volatility is the measure of the changeability in asset returns, which means less volatility is preferable. It describes the variability in a stock price and is used as an estimate of investment risk and for profit possibilities. The volatility is of great interest for financial analyst and provides useful information when estimating investment risk in real trading.

Sharpe Ratio

A risk-adjusted measure of return, with higher ratios preferred to those that are lower. The higher the sharpe ratio, the higher the return, and the lower the volatility.

Normalized Mean Squared Error (NMSE)

NMSE is also used to measure the deviation between the target and the predicted signals. The smaller the values of the NMSE, the closer the predicted signals are to the target signals.

Mean Squared Error (MSE)

MSE is the square of the error between the actual and forecast signals. It is the most frequently used accuracy measure in the literature.

Correct Directional Change (CDC)

CDC measures the capacity of a model to correctly predict the subsequent actual change of a forecast variable.

Signal to Noise Ratio (SNR)

SNR is given in decibels and is used in many other digital applications such as electronic communications and image processing. It contrasts the amount of meaningful information given by the signal, with the amount of background noise which is distraction from the signal. A higher ratio indicates a clearer reading of the signal.

In financial forecasting parlance, accuracy typically refers to profitability. The reason is that, from the trading prospective, the objective is to use the network's predictions to generate profit. Indeed, it does not add value from a financial forecasting perspective, when the network produces very low prediction error, while at the same time it attains a lower profit return. Therefore, it is important for the network to predict the correct direction of change of the signal. Certainly, any model that can predict the direction of change to 100% would be optimal from a profit point of view, regardless of what the error is. However, it appears that the number of direction changes that are correctly predicted is not as important to the annualized return (AR). This relates to the fact that the size of the changes that are correctly predicted will have a greater effect on the AR. If a model is accurate at predicting many smaller changes, it will lose profitability if it fails on the larger changes. Conversely, if a model is accurate at predicting the larger changes, its profitability will be eroded if it fails on many smaller changes. This trade-off is not reflected in the Mean Squared Error (MSE) metric. Low forecast errors and trading profits are not necessarily synonymous since a single large trade forecasted incorrectly by the network could have accounted for most of the trading system's profits (Kaastra And Boyd, 1996). Therefore, it is important to consider the out-of-sample profitability when dealing with financial time series prediction.

Table 6.5: Performance metrics and their calculations

Financial metrics	**Signal Processing and Statistical metrics**
Annualized Return (%AR)	**Normalized Mean Squared Error (NMSE)**
$AR = \frac{Profit}{All\ profit} * 100$ $Profit = \frac{252}{n} * CR$ $CR = \sum_{i=1}^{n} R_i$ $R_i = \begin{cases} +\lvert y_i \rvert & if\ \ (y_i)(\hat{y}_i) \geqslant 0, \\ -\lvert y_i \rvert & otherwise \end{cases}$ $All\ Profit = \frac{252}{n} * \sum_{i=1}^{n} abs(R_i)$	$NMSE = \frac{1}{\sigma^2 n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ $\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2$ $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$
Maximum Drawdown (MDD)	**Mean Squared Error (MSE)**
$MDD = \min\left(\sum_{t=1}^{n} \left(CR_t - \max\left(CR_1, ..., CR_t \right) \right) \right)$ $CR_t = \sum_{i=1}^{t} R_i, t = 1, ..., n$ $R_i = \begin{cases} +\lvert y_i \rvert & if\ \ (y_i)(\hat{y}_i) \geqslant 0, \\ -\lvert y_i \rvert & otherwise \end{cases}$	$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
Annualized Volatility (AV)	**Correct Directional Change (CDC)**
$AV = \sqrt{252} * \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (R_i - \bar{R})^2}$	$CDC = \frac{1}{n} \sum_{i=1}^{n} d_i$ $d_i = \begin{cases} 1 & if\ \ (y_i - y_{i-1})(\hat{y}_i - \hat{y}_{i-1}) \geq 0, \\ 0 & otherwise \end{cases}$
Sharpe Ratio (SR)	
$SR = \frac{Annualized\ \ Return}{Annualized\ \ Volatility}$	
Transaction Cost (TC)	**Signal to Noise Ratio (SNR)**
$TC = 0.01 * Number\ of\ Transaction$ $Number\ of\ Transaction = \sum_{i=1}^{n} L_i$ $L_i = \begin{cases} 1 & if (\hat{y}_i * \hat{y}_{i-1}) < 0, \\ 0 & otherwise \end{cases}$	$SNR = 10 * log_{10}(sigma)$ $sigma = \frac{m^2 * n}{SSE}$ $SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ $m = max(y_i)$

where n is the total number of data patterns.
y and $\hat{y}$ represent the target and predicted output value, respectively.

6.10 Chapter Summary

Financial time series exhibit dynamic behaviour over time. The signals have to be adequately organized and processed before presenting them to the neural network. Lots of attention in the design stage has been given to the pre-processing method to reduce the trend and embedded noise. Together with a careful choice of training parameters and network paradigms, and using appropriate approaches that prevent over-fitting during network training, it is proposed that the developed networks can give some promising results, not only on the forecast error but most importantly on the profit gained.

CHAPTER 7: SIMULATION RESULTS AND ANALYSIS

7.1 Introduction

In this chapter, the simulation results using the MLP and four Higher Order Neural Network architectures; the FLNN, PSNN, RPNN, and the proposed DRPNN, are presented. This chapter is divided into five main sections. Following this section, Section 7.2 discusses the networks predictions using the stationary signals. In section 7.3, the results of the non-stationary (original) versions of the datasets are given. In both section 7.2 and 7.3, analysis on profit return, network convergence, training epochs, performance of networks with increasing order or number of hidden nodes, transaction cost, CPU time, and learning curves are provided. Section 7.4 provides a discussion on the simulation results presented in section 7.2 and 7.3.

7.2 Prediction of Stationary Signals

In this section, extensive reviews of the stationary prediction of ten financial time-series datasets are discussed. Simulation results from the prediction of one step ahead and five steps ahead are given. The results gathered from the best average simulations and best single simulations are presented.

7.2.1 One Step Ahead Prediction using Stationary Signals

As we are concerned with financial time series prediction, in these extensive experiments, our primary interest is not to assess the predictive ability of the network models, but to concentrate on the profitable value contained in the networks predictions. During generalization, the work focuses more on how the network generates the profits. For this reason, the neural networks structure, which provides

the highest percentage of Annualized Return (AR) on out-of-sample data (unseen data) is considered the best model.

7.2.1.1 Best Average Simulation Results

To obtain a fair and more robust comparative evaluation which involves different network architectures, and different starting point of random weights values, a committee of 20 runs has been used to arrive at a trading decision. Table 7.1 through Table 7.5 summarize the average results of 20 simulations obtained on unseen data from ten signals using five neural network architectures. In each table, the network's order (for HONNs), and number of hidden nodes (for MLPs) for the best selected network topology is given in the second column. For the MLPs, the network structures that give the best average results were mostly realized with networks of five hidden nodes. In the case of FLNNs and PSNNs, networks with 2^{nd} and 3^{rd} order were found to appear most in the table. Meanwhile, both RPNNs and DRPNNs in most cases were comprised of the 2^{nd} order network structure. This indicates that the interaction between the input signals for HONNs of order two to three appear to have learned the signals and contains significant information for the prediction task.

The results of the Annualized Return (not accounting the transaction cost) from Table 7.1 through Table 7.5 obviously demonstrated that the proposed DRPNNs profitably attained the highest profit return compared to the MLPs and all other HONNs models in all time series, except for the IBM signal. In the case of predicting the IBM, RPNN has shown to obtain the best profit return. Forecasting the nine signals, DRPNNs outperformed other networks on the average AR by 1.96% to 10.19%. By looking at the other financial metrics; the maximum drawdown, volatility and sharpe ratio, results in Table 7.1 to Table 7.5 clearly show that the best values were dominant by DRPNNs, except for the prediction of IBM signal, in which RPNN gave better results for this signal. It is worth pointing here that for the maximum drawdown and sharpe ratio, a bigger value is preferable. Meanwhile for volatility, a lower value is desirable. When measuring the NMSE, MSE, and SNR, it can be noticed that DRPNNs also outperform all other networks in all signals. MLPs

made the highest NMSE, MSE, and lowest SNR when predicting four out of ten signals, namely the CMESP, CBT-30, JP/US and JP/UK. In the case of evaluating the Correct Directional Change (CDC) obtained by all networks, DRPNNs achieved the highest values in six out of ten signals, which are the CMESP, CBT-10, UK/US, US/EU, JP/EU, and JP/US.

Table 7.1: Best average result from the MLPs

Time Series	No.Hidden Nodes	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	4	81.1081	-2.0021	18.1462	4.4734	0.456727	0.00512	59.35	20.97
CMESP	6	80.4081	-0.6831	4.3145	18.6370	0.428508	0.000507	65.13	29.75
CBT-10	5	81.3089	-0.4004	2.9002	28.0357	0.398103	0.002254	65.44	24.53
CBT-30	7	78.5305	-0.7587	3.7283	21.0636	0.423555	0.002662	66.78	23.52
UK/EU	5	69.6529	-0.5718	2.8022	24.8585	0.45132	0.001858	63.25	24.42
UK/US	5	76.4311	-0.8688	3.9218	19.4891	0.398401	0.002813	62.48	23.57
US/EU	3	76.6488	-1.6467	4.0910	18.7367	0.400491	0.002182	66.27	23.46
JP/EU	3	73.7938	-0.7619	3.7037	19.9248	0.438785	0.000943	64.8	26.48
JP/US	7	74.3301	-0.8493	4.0752	18.2396	0.483811	0.002159	61.95	24.72
JP/UK	5	74.1686	-0.6448	3.6320	20.4215	0.461703	0.001303	60.28	25.75

Table 7.2: Best average result from the FLNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	**2**	81.4285	-1.9084	18.1109	4.4971	0.456703	0.00512	57.74	20.97
CMESP	**2**	80.2347	-0.6831	4.3200	18.5738	0.423157	0.0005	65.03	29.81
CBT-10	**4**	80.3706	-0.4945	2.9214	27.5121	0.398967	0.002259	65.38	24.52
CBT-30	**5**	78.8850	-0.7586	3.7184	21.2146	0.422164	0.002653	67.38	23.53
UK/EU	**3**	69.4822	-0.5786	2.8051	24.7720	0.451294	0.001858	62.93	24.42
UK/US	**5**	76.6273	-0.8688	3.9161	19.5674	0.390958	0.002761	62.79	23.65
US/EU	**2**	76.1346	-1.6467	4.1080	18.5338	0.403777	0.0022	66.67	23.43
JP/EU	**3**	73.5144	-0.7535	3.7117	19.8063	0.440153	0.000946	63.2	26.47
JP/US	**2**	74.1289	-1.0967	4.0811	18.1642	0.479966	0.002142	62.33	24.75
JP/UK	**3**	74.2100	-0.6511	3.6310	20.4380	0.455859	0.001287	60.16	25.8

Table 7.3: Best average result from the PSNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	80.9979	-1.9084	18.1673	4.4592	0.461342	0.005172	57.92	20.93
CMESP	4	79.6335	-0.6831	4.3391	18.3529	0.426327	0.000504	64.81	29.78
CBT-10	3	80.0114	-0.6198	2.9294	27.3137	0.400508	0.002268	65.87	24.51
CBT-30	5	78.9800	-0.7585	3.7158	21.2555	0.423368	0.002661	67.02	23.52
UK/EU	3	69.4939	-0.5786	2.8050	24.7770	0.450935	0.001856	63.52	24.43
UK/US	4	77.4628	-0.8383	3.8914	19.9064	0.396223	0.002798	63.11	23.59
US/EU	3	76.8080	-1.6753	4.0857	18.7994	0.401166	0.002186	66.48	23.46
JP/EU	2	74.3537	-0.7535	3.6878	20.1624	0.439175	0.000944	63.76	26.48
JP/US	2	74.7229	-0.8493	4.0637	18.3878	0.481702	0.002149	61.72	24.74
JP/UK	3	73.9829	-0.6476	3.6368	20.3430	0.458442	0.001294	60.63	25.78

Table 7.4: Best average result from the RPNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	82.3496	-1.9083	17.9912	4.5774	0.456075	0.005113	59.4	20.98
CMESP	2	80.7359	-0.6831	4.3040	18.7591	0.424756	0.000502	65.31	29.79
CBT-10	2	81.3068	-0.4004	2.9003	28.0337	0.396061	0.002243	65.72	24.55
CBT-30	2	79.4325	-0.7586	3.7031	21.4505	0.423554	0.002661	67.22	23.52
UK/EU	5	69.4302	-0.5641	2.8060	24.7477	0.456389	0.001879	63.91	24.37
UK/US	2	78.2625	-0.7846	3.8672	20.2402	0.408806	0.002887	62.43	23.46
US/EU	2	76.7915	-1.0731	4.0860	18.7959	0.401822	0.002189	64.99	23.45
JP/EU	2	75.0900	-0.7535	3.6664	20.4820	0.434805	0.000934	65.13	26.52
JP/US	2	74.8359	-0.8493	4.0604	18.4309	0.47924	0.002138	61.85	24.76
JP/UK	3	74.2434	-0.6476	3.6301	20.4532	0.452	0.001276	61.45	25.84

Table 7.5: Best average result from the DRPNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	79.5567	-1.9085	18.3478	4.3384	0.417100	0.004676	58.99	21.37
CMESP	2	82.6975	-0.5523	4.2396	19.5071	0.329659	0.00039	66.54	30.89
CBT-10	3	86.2569	-0.3424	2.7815	31.0133	0.300985	0.001704	69.42	25.75
CBT-30	2	83.9919	-0.6588	3.5682	23.5403	0.341816	0.002148	66.42	24.45
UK/EU	2	77.1638	-0.5676	2.6580	29.0339	0.365767	0.001506	63.56	25.33
UK/US	2	82.0593	-0.6773	3.7477	21.8991	0.318307	0.002248	64.93	24.54
US/EU	2	82.6223	-0.7092	3.8780	21.3071	0.329805	0.001797	68.44	24.31
JP/EU	3	83.7068	-0.7379	3.3901	24.6944	0.362191	0.000778	65.37	27.32
JP/US	2	77.7430	-0.7975	3.9719	19.5744	0.399054	0.001781	62.45	25.56
JP/UK	2	80.7760	-0.4945	3.4508	23.4087	0.37351	0.001054	60.81	26.67

For demonstration purpose, the annualized return achieved in all network models is depicted in Figure 7.1. Meanwhile, the maximum average number of epochs reached for the prediction of all data signals during the training of the networks is shown in Table 7.6. Apart from the prediction of UK/US, US/EU and JP/EU signals, results given in Table 7.6 demonstrate that the proposed DRPNNs reveal to use least number of epochs to converge on all data, which is 1.15 to 55.49 times faster than other networks. For the prediction of UK/US, US/EU and JP/EU signals, RPNNs appeared to utilize least epochs. Out of ten signals, FLNNs apparently have shown to require larger number of epochs to complete the whole training, specifically on the CBT-30, UK/EU, UK/US, US/EU, JP/EU, and JP/UK.

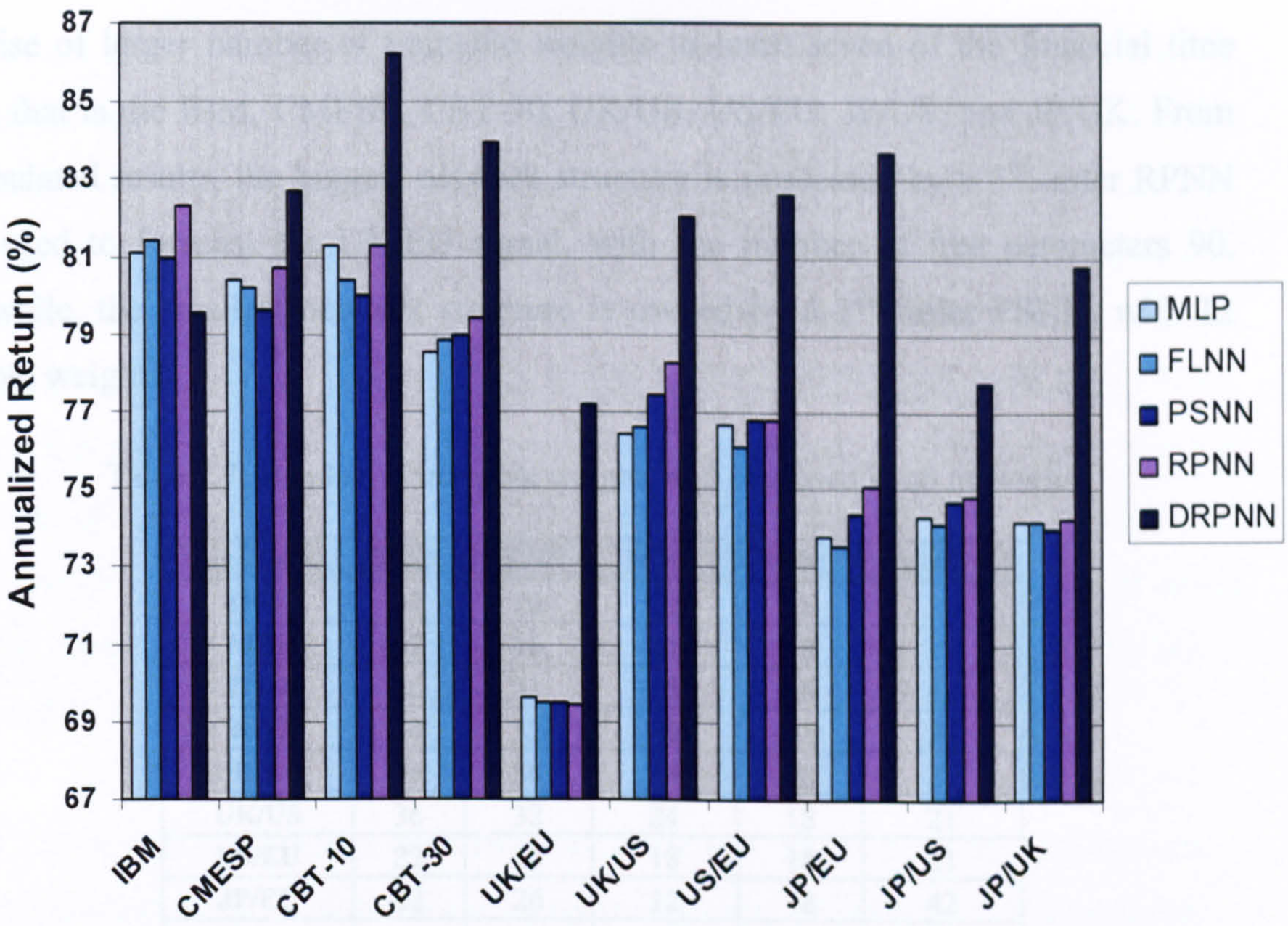


Figure 7.1: Best average annualized return from all network models

Table 7.6: The average maximum epoch reached during training

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	2543	2473	1929	2897	480
CMESP	390	375	433	158	137
CBT-10	1234	154	156	187	48
CBT-30	2298	2608	618	287	47
UK/EU	415	924	237	172	132
UK/US	1712	3000	304	21	110
US/EU	1017	1647	486	27	73
JP/EU	1118	3000	1702	119	258
JP/US	2837	2705	2163	292	178
JP/UK	638	2851	738	142	96

In accordance to the number of epochs used, an analysis of the network size, principally the number of trainable weights employed in the network is of important to judge a network parsimony and simplicity. Table 7.7 demonstrates the results of the number of trainable weights and bias utilized in all network models, calculated from the network structure given in Table 7.1 through Table 7.5. When examining the number of free parameters, it appears that most of the smallest network structures are prevailing by the PSNNs, followed by RPNNs and FLNNS. The MLPs obviously

comprise of larger number of trainable weights to learn seven of the financial time series; that is the IBM, CMESP, CBT-30, UK/US, US/EU, JP/US, and JP/UK. From the tabulated results, the biggest network structure is possessed by a 5^{th} order RPNN when used to forecast the UK/EU signal, with the number of free parameters 90. Meanwhile, the smallest network structure is owned by a 2^{nd} order PSNN, with 12 trainable weights.

Table 7.7: Number of trainable weights and bias used in all networks

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	29	16	12	18	21
CMESP	43	16	24	18	21
CBT-10	36	31	18	18	42
CBT-30	50	32	30	18	21
UK/EU	36	26	18	90	21
UK/US	36	32	24	18	21
US/EU	22	16	18	18	21
JP/EU	22	26	12	18	42
JP/US	50	16	12	18	21
JP/UK	36	26	18	36	21

In order to test the modelling capabilities and the stability of all network modes, Figure 7.2 and Figure 7.3 illustrate the best average result of AR and NMSE, respectively, tested on unseen data, when used to predict the financial signals. The performance of the networks was evaluated with the number of higher order terms increased from 1 to 5 (for RPNNs and DRPNNs) and from 2 to 5 (for FLNNs and PSNNs), and different number of hidden nodes increased from 3 to 8 (for MLPs). The plots in Figure 7.2 indicate that the performance of the proposed DRPNN continues to rise when a 2^{nd} order Pi-Sigma unit was added to the networks, and the AR began to drop when the 3^{rd} order Pi-Sigma unit was added, except for the prediction of JP/EU and CBT-10 signals in which the AR continue to increase along with the network growth. For the RPNNs, the AR for the IBM and UK/EU signals keep increasing until network of order five. For the CMESP, CBT-10, CBT-30, UK/US, JP/EU, US/EU, and JP/US signals, RPNNs' performance start to degrade after a 3^{rd} order Pi-Sigma unit was added to the networks, except for the prediction of JP/UK where the AR only drop when 4^{th} order Pi-Sigma unit was added. In most cases, when the networks order or number of hidden nodes were expanded, the plots for PSNNs and MLPs show a rise in the AR. Their performance then starts to degrade when they reached order higher than

three or four (for PSNNs), and hidden nodes more than five, six, or seven (for MLPs). This can be seen when PSNNs used to predict the CMESP, UK/US, UK/EU, US/EU signals, and when MLPs predicting the CMESP, UK/US, UK/EU, JP/UK, and CBT-30 signals. Meanwhile, in some plots, the networks demonstrate an up and down movement, namely when PSNNs predicting the JP/EU, JP/UK, CBT-10 signals, and when MLPs predicting the IBM, US/EU, JP/US, and CBT-10 signals. For the FLNNs, most of the plots demonstrate an up and down movement of AR, namely for the prediction of CMESP, UK/EU, US/EU, JP/EU, JP/UK, and CBT-10 signals. Apart from some cases, the performance of the networks show an increment in the AR along with the networks' growth (for UK/US and CBT-30 signals), and a continuously decreased of AR (for IBM and JP/US signals).

Figure 7.3 depicts the average performance of various neural network architectures using the NMSE measure with increasing networks order or number of hidden nodes. For the prediction of CBT-30, JP/EU, US/EU, UK/EU, and IBM, the NMSE of DRPNNs started to rise up when a 3rd order Pi-Sigma unit was added to the networks. For the CBT-10 and JP/UK signals, the plots for DRPNNs demonstrate a continuously decreasing NMSE. Conversely for the prediction of CMESP, UK/US, and JP/US signals, the plots show an increasing NMSE. In the case of RPNNs, the plots for prediction of IBM, CBT-10, UK/EU and JP/EU signals reveal an incessantly decreasing NMSE, and that of for CMESP signal shows a continuously increasing NMSE. Meanwhile, RPNNs when used to forecast the CBT-30, UK/US, US/EU, JP/US and JP/UK signals demonstrate an up and down movements of NMSE. PSNNs exhibit an increased NMSE along with the network growth in most of the signals prediction. When using the FLNNs to predict the CMESP, UK/EU, JP/US, and CBT-10 signals, the NMSE were increasing and began to drop when networks' order were expanded, except for the IBM signal in which the NMSE stay increase until network of order five. Forecasting the CBT-30, UK/US, US/EU, JP/EU, JP/UK signals, the plots for the FLNNs demonstrate a small up and down movement of NMSE. Meanwhile, the performance of MLPs overall show a little of up and down movement, namely when predicting the IBM, UK/EU, US/EU, JP/US, JP/UK, and CBT-30 signals.

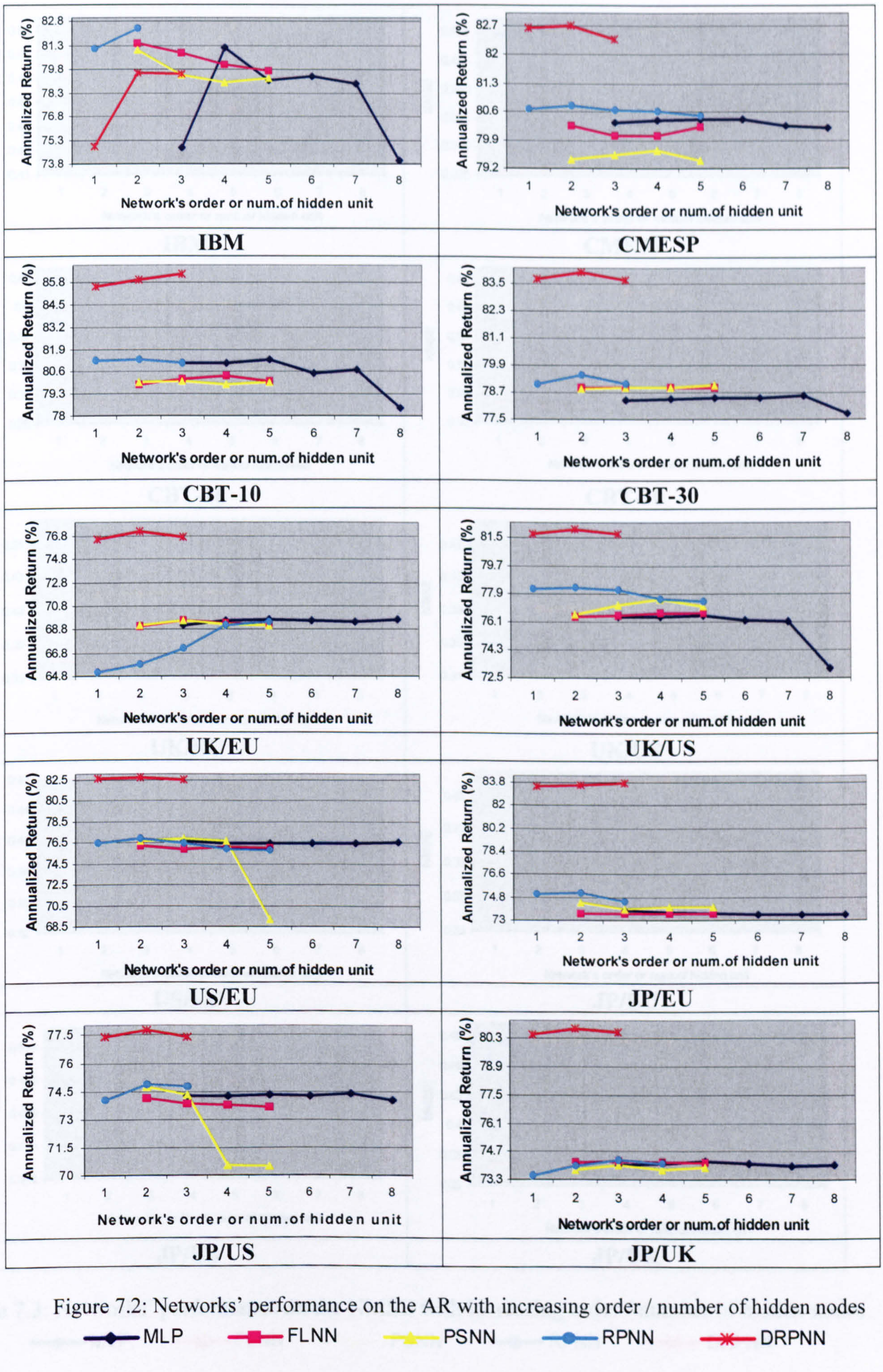


Figure 7.2: Networks' performance on the AR with increasing order / number of hidden nodes

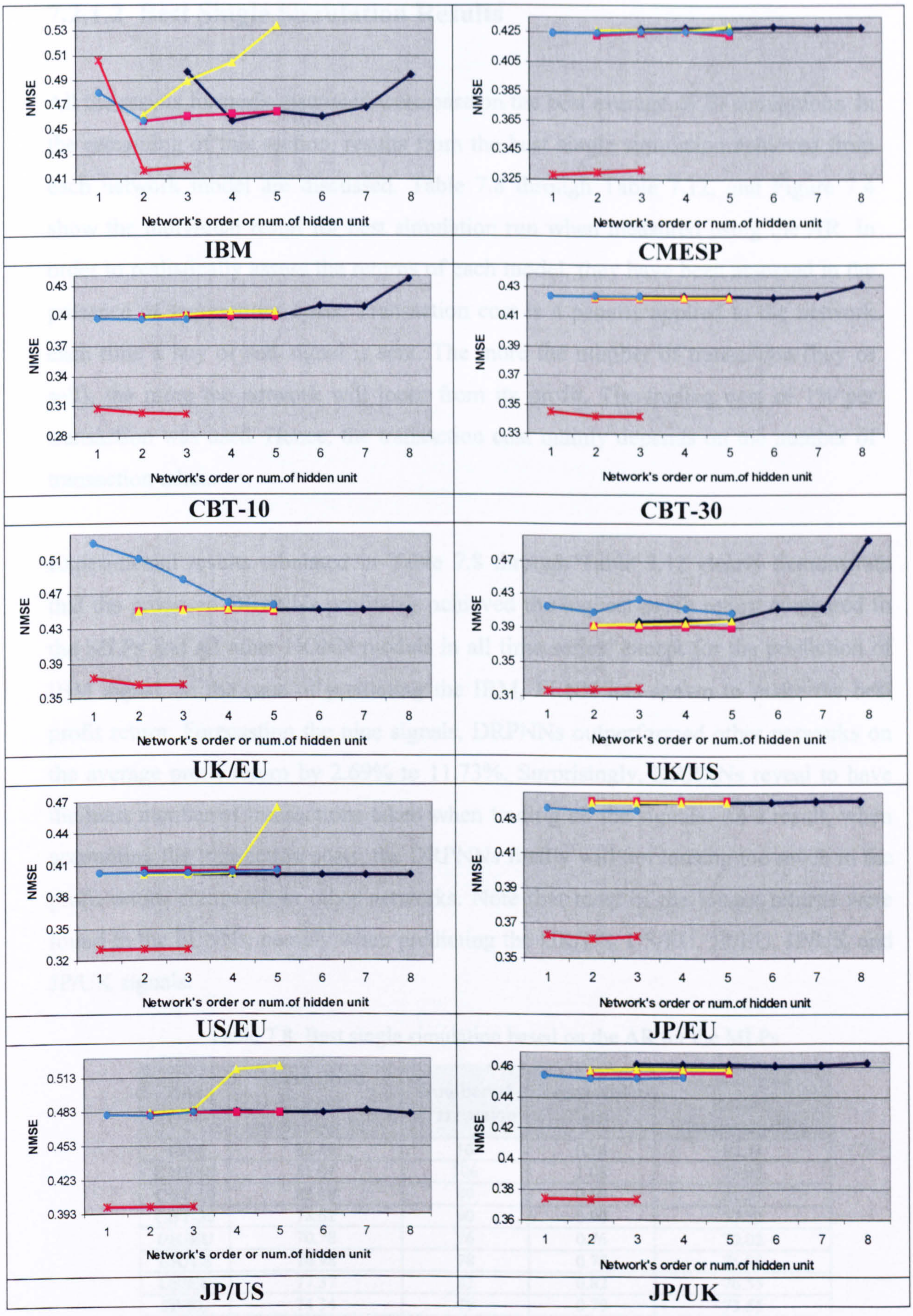


Figure 7.3: Networks' performance on the NMSE with increasing order / number of hidden nodes

7.2.1.2 Best Single Simulation Results

All the results formerly discussed were base on the best average of 20 simulations. In the remaining of this section, results from the best single simulation achieved from each network model are discussed. Table 7.8 through Table 7.12, and Figure 7.4 show the individual result for best simulation run when measured using the AR. In order to realistically assess the returns of each model, they have been assessed in the presence of transactions costs. Transaction cost is a penalty applied to the network each time a buy or sell signal is sent. The more the number of transaction (buy or sell), the more the network will loose from its profit. The trading cost of 1% per transaction was used. Hence, the transaction cost mainly depends on the number of transaction taken.

Experimental results tabulated in Table 7.8 through Table 7.12 clearly demonstrate that the proposed DRPNNs profitably achieved the highest profit return compared to the MLPs and all other HONN models in all time series, except for the prediction of IBM signal. In the case of predicting the IBM, FLNN has shown to make the best profit return. Forecasting the nine signals, DRPNNs outperformed other networks on the average profit return by 2.69% to 11.73%. Surprisingly, DRPNNs reveal to have the least number of transactions taken when trading all the signals. As a result, when accounting the transaction costs, the DRPNNs finally will not loosing too much in the profit return compared to other networks. Note that most of the lowest returns were found in the FLNNs, namely when predicting the UK/US, US/EU, JP/EU, JP/US, and JP/UK signals.

Table 7.8: Best single simulation based on the AR for the MLPs

Time Series	Annualized Return (excluding TC)	Number of Transaction	Transaction Cost	Annualized Return (including TC)
IBM	82.49	16	0.16	82.33
CMESP	81.01	106	1.06	79.95
CBT-10	82.16	89	0.89	81.27
CBT-30	78.62	90	0.90	77.72
UK/EU	70.78	76	0.76	70.02
UK/US	76.98	78	0.78	76.20
US/EU	77.37	82	0.82	76.55
JP/EU	74.34	79	0.79	73.55
JP/US	74.47	78	0.78	73.69
JP/UK	74.91	83	0.83	74.08

Table 7.9: Best single simulation based on the AR for the FLNNs

Time Series	Annualized Return (excluding TC)	Number of Transaction	Transaction Cost	Annualized Return (including TC)
IBM	83.13	18	0.18	82.95
CMESP	81.33	106	1.06	80.27
CBT-10	81.22	91	0.91	80.31
CBT-30	78.90	88	0.88	78.02
UK/EU	70.70	76	0.76	69.94
UK/US	76.91	80	0.80	76.11
11US/EU	77.26	78	0.78	76.48
JP/EU	73.51	79	0.79	72.72
JP/US	74.27	80	0.80	73.47
JP/UK	74.22	79	0.79	73.43

Table 7.10: Best single simulation based on the AR for the PSNNs

Time Series	Annualized Return (excluding TC)	Number of Transaction	Transaction Cost	Annualized Return (including TC)
IBM	83.03	16	0.16	82.87
CMESP	79.99	120	1.20	78.79
CBT-10	81.14	97	0.97	80.17
CBT-30	79.51	86	0.86	78.65
UK/EU	70.33	76	0.76	69.57
UK/US	77.87	74	0.74	77.13
US/EU	77.33	82	0.82	76.51
JP/EU	74.88	79	0.79	74.09
JP/US	74.80	82	0.82	73.98
JP/UK	74.15	79	0.79	73.36

Table 7.11: Best single simulation based on the AR for the RPNNs

Time Series	Annualized Return (excluding TC)	Number of Transaction	Transaction Cost	Annualized Return (including TC)
IBM	83.03	16	0.16	82.87
CMESP	81.27	110	1.1	80.17
CBT-10	81.72	92	0.92	80.80
CBT-30	79.62	86	0.86	78.76
UK/EU	71.74	74	0.74	71.00
UK/US	80.39	64	0.64	79.75
US/EU	78.56	82	0.82	77.74
JP/EU	75.69	77	0.77	74.92
JP/US	75.13	80	0.80	74.33
JP/UK	75.35	77	0.77	74.58

Table 7.12: Best single simulation based on the AR for the DRPNNs

Time Series	Annualized Return (excluding TC)	Number of Transaction	Transaction Cost	Annualized Return (including TC)
IBM	81.47	12	0.12	81.35
CMESP	83.76	80	0.80	82.96
CBT-10	87.52	75	0.75	86.77
CBT-30	85.17	70	0.70	84.47
UK/EU	78.45	54	0.54	77.91
UK/US	83.75	55	0.55	83.20
US/EU	83.49	49	0.49	83.00
JP/EU	85.02	57	0.57	84.45
JP/US	78.72	60	0.60	78.12
JP/UK	81.59	51	0.51	81.08

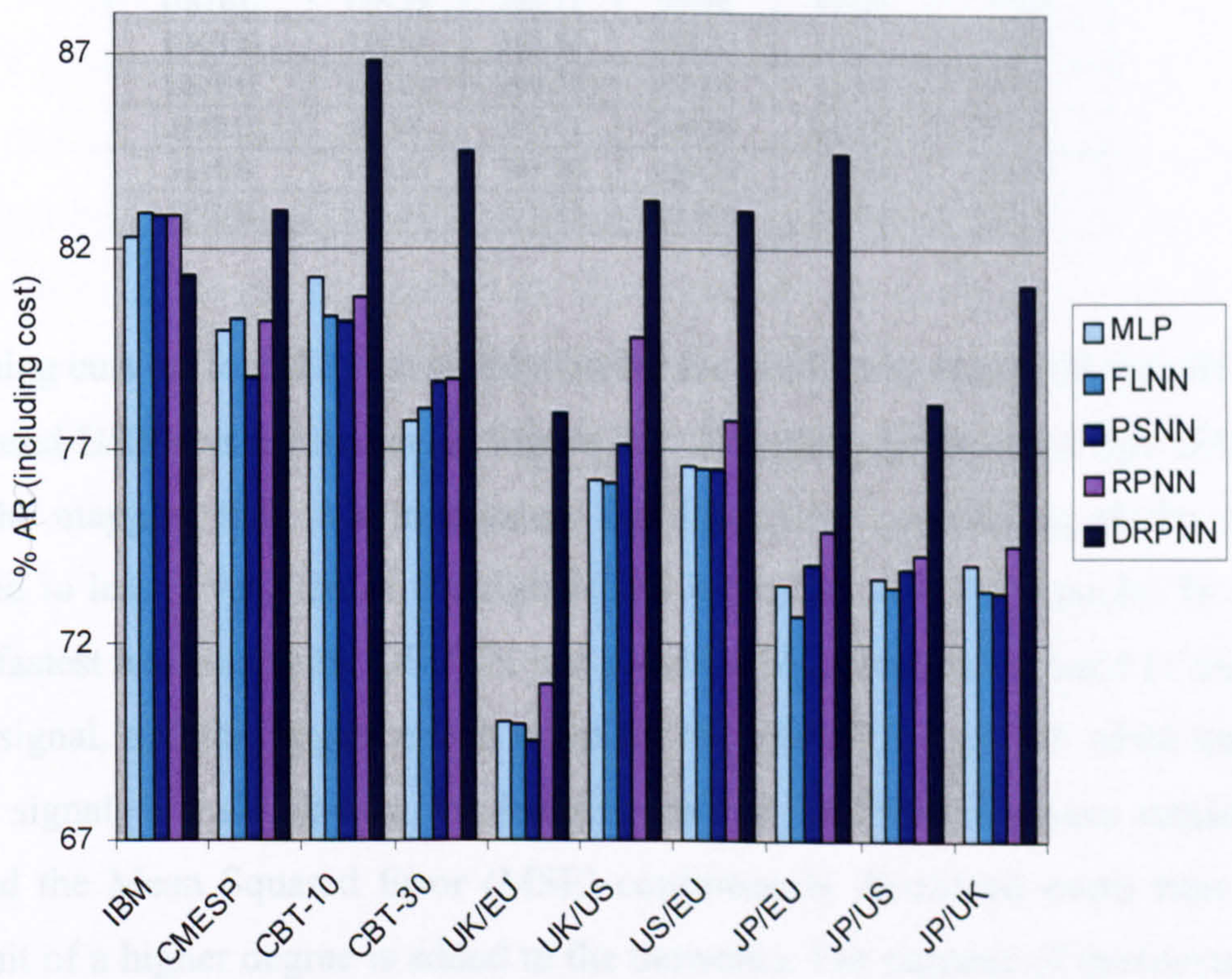


Figure 7.4: Best ARs (including the transaction cost)

In order to compare the speed of the networks to execute and complete the training, Table 7.13 shows the amount of CPU time for all networks when used to learn all the signals. CPU time is the amount of time a computer program uses in processing on a CPU, often measured in clock ticks. It is used as a point of comparison for CPU usage of a program. The CPU time was based on a machine with Windows XP 2000, Intel processor (Pentium 4), CPU of 3.00 GHz, and 1 GB of RAM. The proposed network, DRPNNs broadly used the least CPU time when compared to other neural

networks. Training the IBM, CBT-30, UK/EU, JP/US and JP/UK signals, DRPNNs outperformed other network models by 1.05 to 17.37 time faster in CPU time. RPNNs made the best CPU time when trained three of the signals, namely the UK/US, US/EU, and JP/EU signals. Most of the longest CPU times were found in FLNNs, which is when training the CBT-30, UK/EU, UK/US, US/EU, JP/EU, JP/US and JP/UK signals.

Table 7.13: CPU time usage for training each neural network

Predictor	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	94.27	56.86	130.95	93.69	54.09
CMESP	99.53	27.25	113.88	52.50	95.34
CBT-10	80.33	11.95	34.81	66.33	37.63
CBT-30	445.39	484.44	256.14	110.87	27.89
UK/EU	138.53	382.11	99.48	42.69	37.28
UK/US	329.61	382.42	214.53	9.48	73.88
US/EU	132.61	366.55	89.16	11.19	67.19
JP/EU	88.98	393.61	269.09	40.55	71.09
JP/US	102.55	381.20	159.14	158.34	66.94
JP/UK	315.31	373.45	80.906	67.00	37.77

The learning curves from the best simulation for the prediction of all data signals using the proposed DRPNNs are shown in Figure 7.5. The plots demonstrate that DRPNNs learned the mapping task in a moderately rapid learning, considering all the curves when used to learn every ten of the signals end up at less than 600 epochs. In actual fact, the fastest learning using DRPNN just required 30 epochs when used to train the CBT-10 signal, and the largest epoch taken by the DRPNNs was 595 when learning the IBM signal. For all signals, the learning curves for DRPNNs were remarkably stable and the Mean Squared Error (MSE) continuously decreased every time a Pi-Sigma unit of a higher degree is added to the networks. For purpose of demonstration, Appendix 2 shows the respective learning curves for the other network models; the MLPs, FLNNs, PSNNs, and RPNNs, collectively with the DRPNNs. In most cases, DRPNNs, RPNNs, and FLNNs learnt all the signals very quickly when compared to other network models. It is shown that DRPNNs have accomplished the fastest learning on four time series, which is when used to predict the IBM, CBT-10, CBT-30, and JP/UK time series. The RPNNs have made the fastest learning on four signals; which are the JP/EU, US/EU, UK/US and UK/EU signals. Meanwhile the FLNNs converged fastest on the prediction of CMESP and JP/US signals. MLPs utilized the largest epochs when used to train the IBM, CMESP, CBT10, CBT-30, JP/US, US/EU,

and UK/EU, while the FLNNs showed the largest number of epochs when learning the IBM, JP/UK, JP/EU, and UK/US signals.

Following the learning curve plots, Figure 7.6 shows the best prediction on all signals using the proposed DRPNNs. The plots were taken from the first 100 data points from the unseen part of the data, except for the IBM as the signal has less than 100 point of out-of-sample data. For demonstration purpose, the relevant plots for the other network models are shown in Appendix 3. By looking at the plots in Figure 7.6 and Appendix 3, it can be spotted that the original and predicted signals are pretty close to each other. This may indicate that all the networks are likely capable at mapping the underlying movements in stationary financial markets. Meanwhile, Figure 7.7 presents the histograms of the prediction errors using DRPNNs, which signifies that all the prediction errors are near to zero and follow closely to the normal distribution. Obviously, signal's error that approaching zero is desirable properties in Neural Network predictions. Histograms of the prediction errors for the other four networks; the MLPS, FLNNs, PSNNs, and RPNNs are shown in Appendix 4, which demonstrate desirable prediction errors that close to zero.

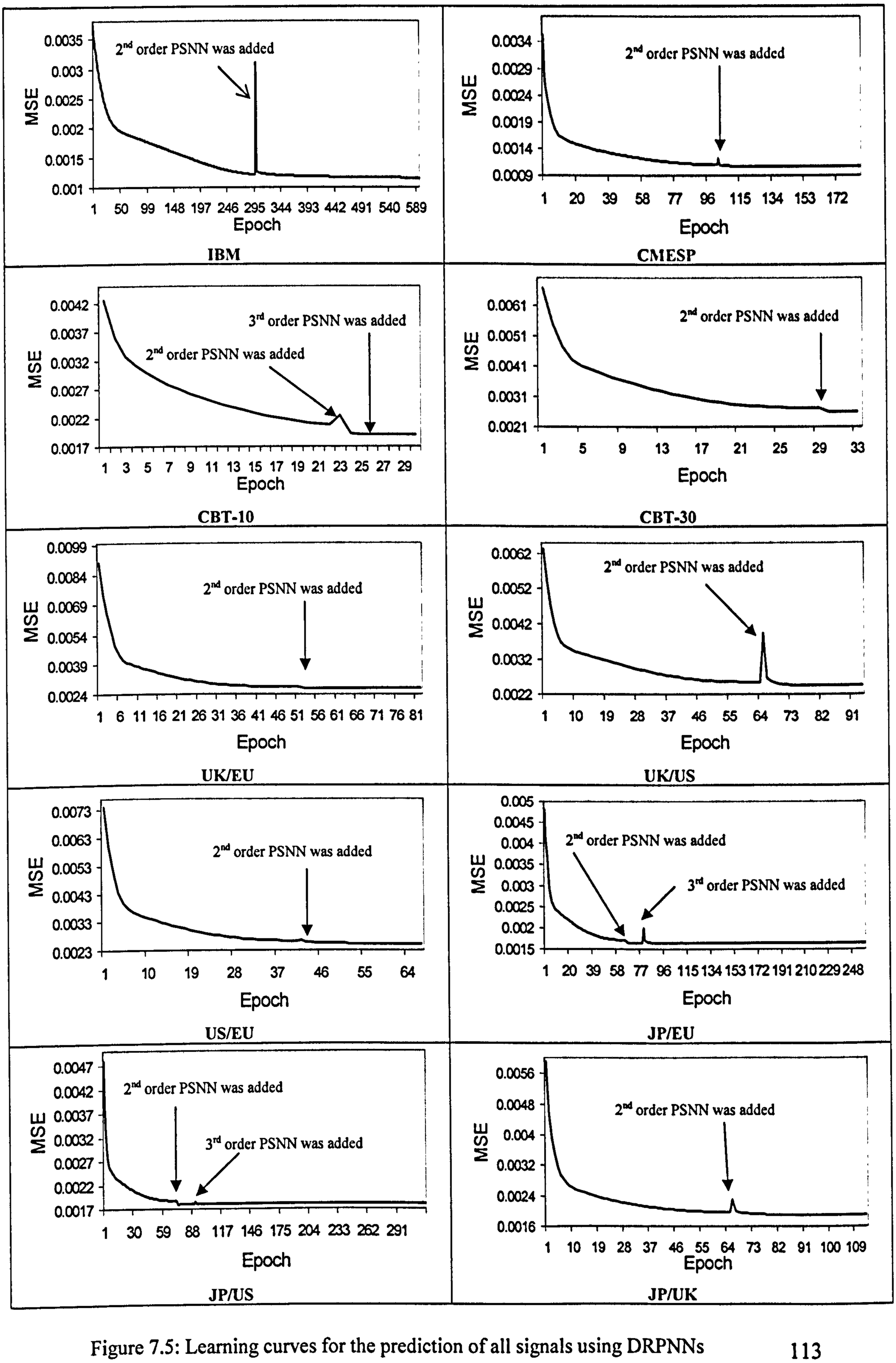


Figure 7.5: Learning curves for the prediction of all signals using DRPNNs

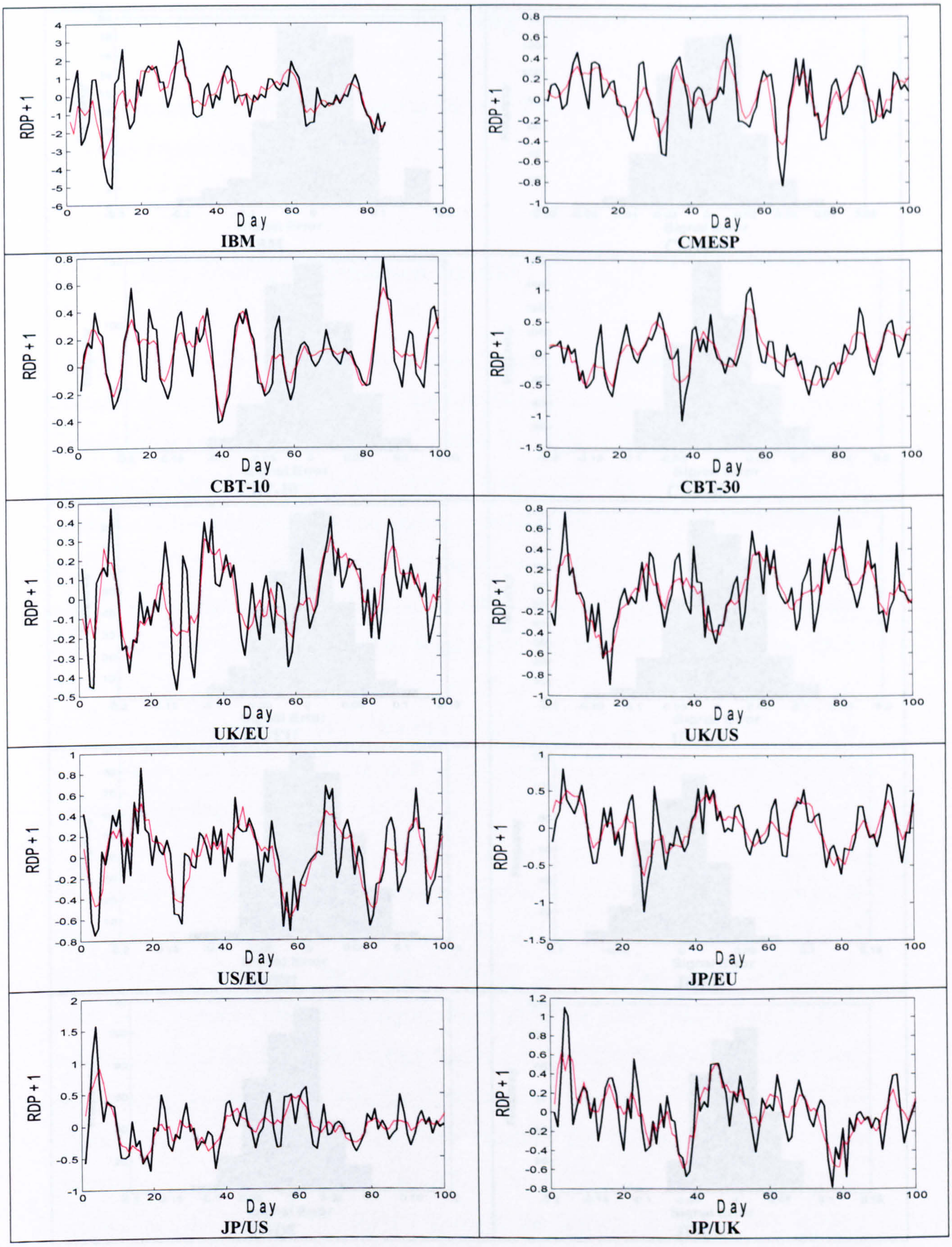


Figure 7.6: Best forecasts made by DRPNN on all data signals
Original signal, Predicted signal

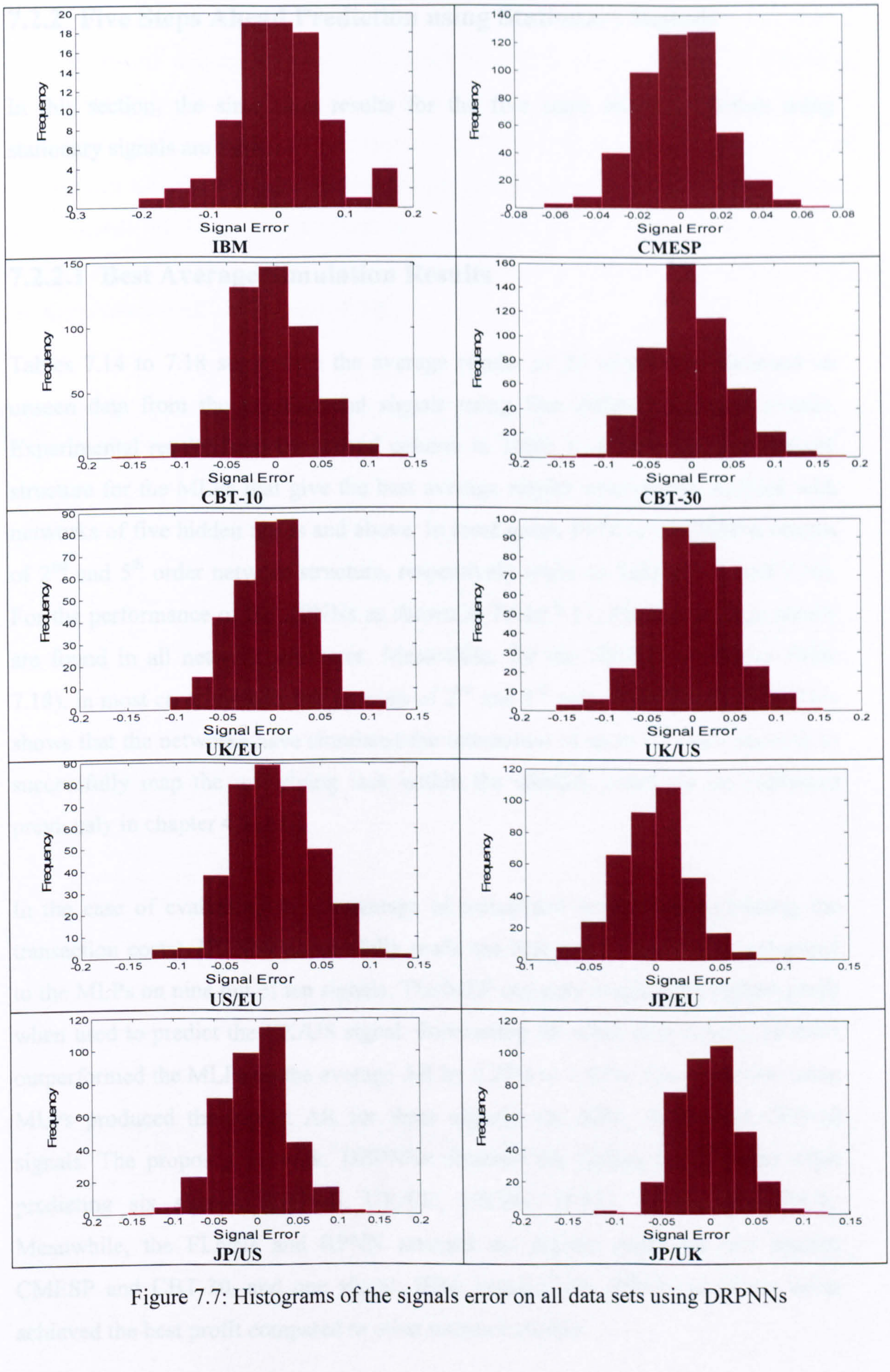


Figure 7.7: Histograms of the signals error on all data sets using DRPNNs

7.2.2 Five Steps Ahead Prediction using Stationary Signals

In this section, the simulation results for the five steps ahead prediction using stationary signals are shown.

7.2.2.1 Best Average Simulation Results

Tables 7.14 to 7.18 summarize the average results of 20 simulations obtained on unseen data from the ten financial signals using five different neural networks. Experimental results from the second column in Table 7.14 show that the network structure for the MLPs that give the best average results were mostly realized with networks of five hidden nodes and above. In most cases, FLNNs and PSNNs consist of 2^{nd} and 5^{th} order network structure, respectively (refer to Tables 7.15 and 7.16). For the performance of the RPNNs as shown in Table 7.17, the best average results are found in all network structures. Meanwhile, for the DRPNNs (refer to Table 7.18), in most cases the network consists of 2^{nd} and 3^{rd} order network structure. This shows that the networks have simulated the interaction of up to 3^{rd} order network to successfully map the underlying task within the stability condition (as explained previously in chapter 4 and 6).

In the case of evaluating the percentage of annualized return (not accounting the transaction costs), HONNs successfully made the best profit return when compared to the MLPs on nine out of ten signals. The MLP can only attained the highest profit when used to predict the UK/US signal. Forecasting the other nine signals, HONNs outperformed the MLPs on the average AR by 0.25% to 2.69%. The prediction using MLPs produced the lowest AR for three signals; the IBM, JP/EU and CBT-10 signals. The proposed network, DRPNNs obtained the highest profit return when predicting six signals; CBT-10, UK/EU, US/EU, JP/EU, JP/US, and JP/UK. Meanwhile, the FLNNs and RPNN attained the highest profit on two signals; CMESP and CBT-30, and one signal; IBM, respectively. PSNN, however, never achieved the best profit compared to other network models.

In terms of other financial measures; the maximum drawdown, volatility and sharpe ratio, it appears that most of the best values were dominant by DRPNNs. DRPNNs also show the highest values in six out of ten signals when assessed with the correct directional change (CDC). When measuring the maximum drawdown, it can be noticed that all HONN models have lower maximum loss compared to the MLPs, and this suggests that HONNs have less downside risk. In the case of evaluating the NMSE, MSE, and SNR, HONNs models outperformed the MLPs in all signals, except for the prediction of UK/EU and JP/UK.

Table 7.14: Best average result from the MLPs

Time series	No.Hidden Nodes	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	7	89.4021	-6.7628	51.2332	1.7308	0.3343	0.004377	64.11	21.68
CMESP	8	85.6451	-2.0462	13.5867	6.3036	0.2945	0.000835	64.81	27.39
CBT-10	7	86.1028	-1.9829	9.503	9.006	0.2537	0.001935	67.99	25.20
CBT-30	4	88.6882	-1.3067	11.2442	7.8756	0.2153	0.001502	65.58	25.75
UK/EU	8	86.6448	-1.5427	8.3108	10.4258	0.2207	0.001018	66.01	26.66
UK/US	3	88.1342	-1.5179	12.5555	7.0138	0.2088	0.001720	60.35	25.71
US/EU	3	87.8804	-2.6447	12.3825	7.071	0.2375	0.001742	66.24	23.81
JP/EU	7	87.0519	-1.7986	11.1881	7.7777	0.2156	0.000719	64.69	27.84
JP/US	6	83.5513	-2.0705	12.4382	6.7103	0.2694	0.001766	58.49	25.60
JP/UK	5	88.9711	-1.9827	10.8699	8.1808	0.2083	0.001001	59.51	26.61

Table 7.15: Best average result from the FLNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	3	90.2120	-3.6676	50.4281	1.7855	0.2764	0.003619	64.18	22.48
CMESP	4	85.8998	-2.0445	13.5617	6.3306	0.2946	0.000836	64.51	27.39
CBT-10	2	86.2743	-1.8786	9.4415	9.134	0.2510	0.001915	67.94	25.24
CBT-30	2	89.1699	-1.0847	11.1642	7.984	0.2138	0.001494	64.80	25.77
UK/EU	3	85.6429	-1.5427	8.3975	10.1945	0.2238	0.001032	65.93	26.60
UK/US	4	88.0578	-1.5179	12.5584	7.0104	0.2069	0.001704	59.97	25.75
US/EU	2	87.4584	-2.6447	12.4262	7.0226	0.2414	0.001771	66.22	23.74
JP/EU	3	87.3362	-1.5221	11.1546	7.8243	0.2134	0.000712	64.64	27.88
JP/US	5	84.7522	-1.5873	12.2948	6.8935	0.2573	0.001687	58.52	25.79
JP/UK	2	88.8414	-1.9827	10.8919	8.1478	0.2084	0.001010	59.04	26.59

Table 7.16: Best average result from the PSNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	90.1036	-5.3672	50.8658	1.7559	0.2858	0.003742	65.00	22.34
CMESP	4	85.5844	-2.0450	13.5994	6.2901	0.2954	0.000838	64.63	27.38
CBT-10	5	86.1676	-1.8786	9.448	9.1203	0.2515	0.001918	67.52	25.23
CBT-30	4	88.7304	-1.1513	11.226	7.9003	0.2167	0.001512	65.10	25.73
UK/EU	5	86.3448	-1.5427	8.3362	10.3578	0.2234	0.001030	66.55	26.61
UK/US	2	87.9867	-1.5581	12.6208	6.9384	0.2056	0.001694	60.05	25.77
US/EU	2	87.5358	-2.3908	12.4018	7.0497	0.2369	0.001738	66.15	23.82
JP/EU	5	87.0561	-1.8327	11.1831	7.7846	0.2133	0.000711	64.04	27.89
JP/US	5	83.5263	-1.5937	12.431	6.7192	0.2656	0.001742	58.73	25.65
JP/UK	5	88.8596	-1.9827	10.8778	8.1689	0.2090	0.001003	59.92	26.60

Table 7.17: Best average result from the RPNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	3	90.7125	-4.3144	50.8542	1.7855	0.2701	0.003536	63.33	22.58
CMESP	3	85.6441	-2.0449	13.5994	6.2949	0.2959	0.000839	64.80	27.37
CBT-10	5	86.5960	-1.8977	9.5041	9.1138	0.2564	0.002055	67.13	25.15
CBT-30	3	88.7508	-1.2623	11.2273	7.905	0.2138	0.001492	65.12	25.76
UK/EU	3	86.6437	-1.4312	8.3389	10.3918	0.2314	0.001068	65.17	26.46
UK/US	5	87.1448	-1.5143	12.7251	6.8484	0.2089	0.001721	61.13	25.70
US/EU	2	88.3191	-1.5891	12.273	7.1966	0.2506	0.001838	64.23	23.58
JP/EU	4	87.4830	-1.8623	11.2154	7.8009	0.2152	0.000718	64.24	27.85
JP/US	2	84.8365	-2.9646	12.4862	6.7981	0.2926	0.001919	58.59	25.24
JP/UK	4	89.2521	-1.4883	10.8557	8.2226	0.2084	0.001912	59.68	26.60

Table 7.18: Best average result from the DRPNNs

Time series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	90.7075	-3.6165	50.0070	1.8152	0.3059	0.004006	62.98	22.05
CMESP	2	85.7694	-2.0457	13.617	6.2582	0.2909	0.000825	65.54	27.45
CBT-10	3	87.3525	-1.6071	9.3363	9.3575	0.2549	0.001944	66.87	25.18
CBT-30	3	88.0966	-1.1630	11.3063	7.7923	0.2137	0.001491	64.77	25.78
UK/EU	2	87.5726	-1.0129	8.3067	10.5427	0.2231	0.001029	65.75	26.62
UK/US	3	87.4671	-1.4861	12.668	6.905	0.2160	0.001779	61.25	25.56
US/EU	2	88.8278	-1.4518	12.2705	7.2402	0.2577	0.001890	64.11	23.46
JP/EU	3	87.8136	-1.8327	11.093	7.916	0.2159	0.000720	64.92	27.83
JP/US	2	86.2388	-2.7460	12.659	6.823	0.3026	0.001984	59.37	25.09
JP/UK	2	89.4970	-1.3552	10.7994	8.2875	0.2120	0.001017	61.24	26.54

For demonstration purpose, the annualized return achieved in all network models (as given in Tables 7.14 to 7.18) is plotted in Figure 7.8. Meanwhile, the maximum average number of epochs reached for the prediction of all data signals during the training of the networks is shown in Table 7.19. DRPNNs and RPNNs have revealed to use less number of training cycles (epochs) to converge on all data, which is equivalent to being 1.07 to 600 times faster than the other networks. In view of that, DRPNNs have shown to require the least number of epochs to converge on six out of ten signals. In all financial signals, FLNNs and MLPs appeared to utilize more epochs to complete the training.

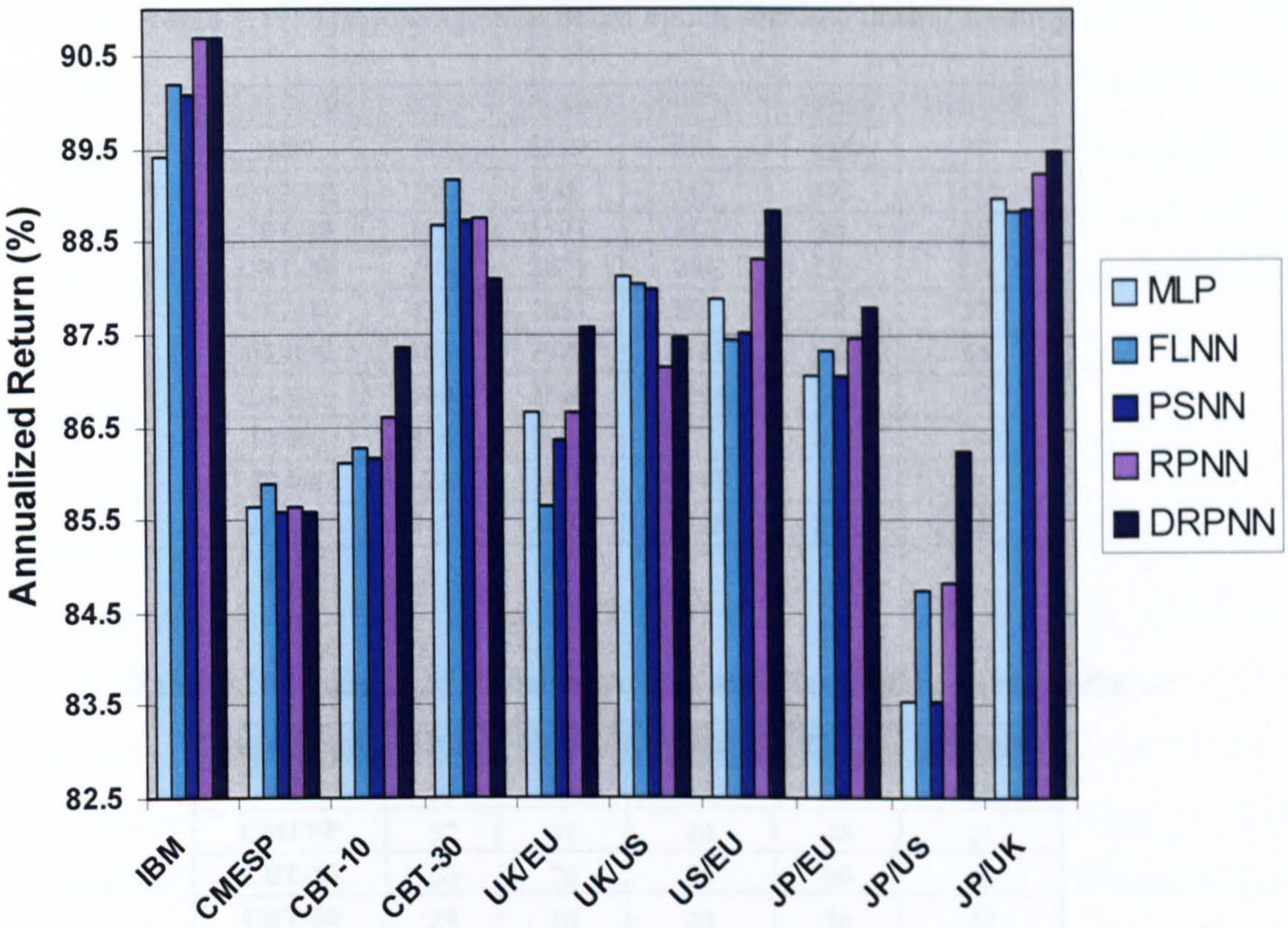


Figure 7.8: Best average annualized return from all network models

In accordance to the number of epochs used, Table 7.20 demonstrates the results on the number of trainable weights and biases utilized in all network models calculated from the network structure given previously in Tables 7.14 to 7.18. It can be noticed that most of the smallest network structures is dominant by FLNNs, followed by PSNNs and DRPNNs. The MLPs and RPNNs obviously comprise of larger number of trainable weights to learn the financial time series. From the tabulated results, the biggest network structure is possessed by the RPNNs of order five, that is when used to predict the CBT-10 and UK/US signals, with the number of free parameters of 90. Meanwhile, the smallest network structure is owned by a 2^{nd} order PSNNs, with 12 trainable weights. MLPs never present with the smallest network size.

Table 7.19: The average maximum epoch reached during training

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	568	2519	651	425	79
CMESP	907	645	312	193	255
CBT-10	1395	1104	242	86	30
CBT-30	744	2870	244	155	216
UK/EU	1365	2851	893	44	57
UK/US	2050	2983	2543	245	54
US/EU	3000	3000	1294	24	5
JP/EU	3000	3000	871	817	130
JP/US	699	1489	1141	8	9
JP/UK	1179	3000	1078	298	42

Table 7.20: Number of trainable weights and bias used in all networks

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	50	26	12	36	21
CMESP	57	31	24	36	21
CBT-10	50	26	30	90	42
CBT-30	29	16	24	36	42
UK/EU	57	26	30	36	21
UK/US	22	31	12	90	42
US/EU	22	16	12	18	21
JP/EU	50	26	30	60	42
JP/US	43	32	30	18	21
JP/UK	36	16	30	60	21

To configure the modelling capabilities and the stability of the neural networks, Figures 7.9 and 7.10 illustrate the best average result of AR and NMSE, respectively, tested on out-of-sample data, when used to predict the financial signals. The performance of the networks was evaluated with the number of higher order terms increased from 1 to 5 (for RPNNs and DRPNNs), and from 2 to 5 (for FLNNs and PSNNs), and number of hidden nodes increased from 3 to 8 (for MLP). The plots in Figure 7.9 indicate that all HONNs models generally learned the data steadily with the AR continues to increase along with the network growth. After sometimes when they reached a certain higher order structure, the performance start to degrade, and usually the AR will not rise up again. This can be seen when FLNNs predicting the IBM, UK/US, and UK/EU; PSNNs predicting the CMESP, and CBT-30; RPNNs predicting the IBM, CMESP, US/EU, JP/EU, JP/UK, and CBT-30. In some cases, the performances of the networks show an up and down movement of AR (FLNNs when

predicting the CMESP, JP/US, JP/EU, JP/UK, CBT-10, and RPNN when predicting the JP/US). Some of the plots in Figure 7.9 demonstrate decreasing AR (FLNNs when predicting the US/EU, CBT-30, and PSNN when predicting UK/US). When predicting the IBM, UK/EU, and US/EU signals, PSNNs showed increased AR only after a 4^{th} or 5^{th} order network structure were utilized. Meanwhile, the AR keep rising from network of order one to five when using PSNNs to predict the JP/US, JP/EU, JP/UK, CBT-10; and when using RPNNs to predict the UK/US, UK/EU, and CBT-10.

DRPNNs specifically show an increasing AR from network structure of order one to three when predicting the CMESP, UK/US, JP/EU, CBT-10, and CBT-30 signals. For the prediction of IBM, UK/EU, US/EU, JP/US, and JP/UK; the AR began to drop beyond the 2^{nd} order structure. On the other hand, the performance of the MLPs in most of the plots in Figure 7.9 show an up and down movement, indicating that there is no clear pattern whether the profit is going up or down when the number of hidden nodes in the network were appended. Meanwhile, MLP when used to predict the CBT-10 signal generated a continuously increasing profit with the increment number of hidden nodes.

Figure 7.10 demonstrates the average performance of the NMSE with increasing networks order or number of hidden nodes. RPNNs and DRPNNs exhibit drastically decreased NMSE along with the network growth. Apart from the prediction of CBT-10 and JP/EU, the NMSE for RPNNs and DRPNNs started to rise up when a 3^{rd}, and 2^{nd} order Pi-Sigma unit, respectively, is added to the networks. On the contrary, the MLPs, FLNNs, and PSNNs revealed to show a little of an up and down motion on the NMSE, and in certain cases, the NMSE keep on increasing, except for the prediction of US/EU when using the PSNN.

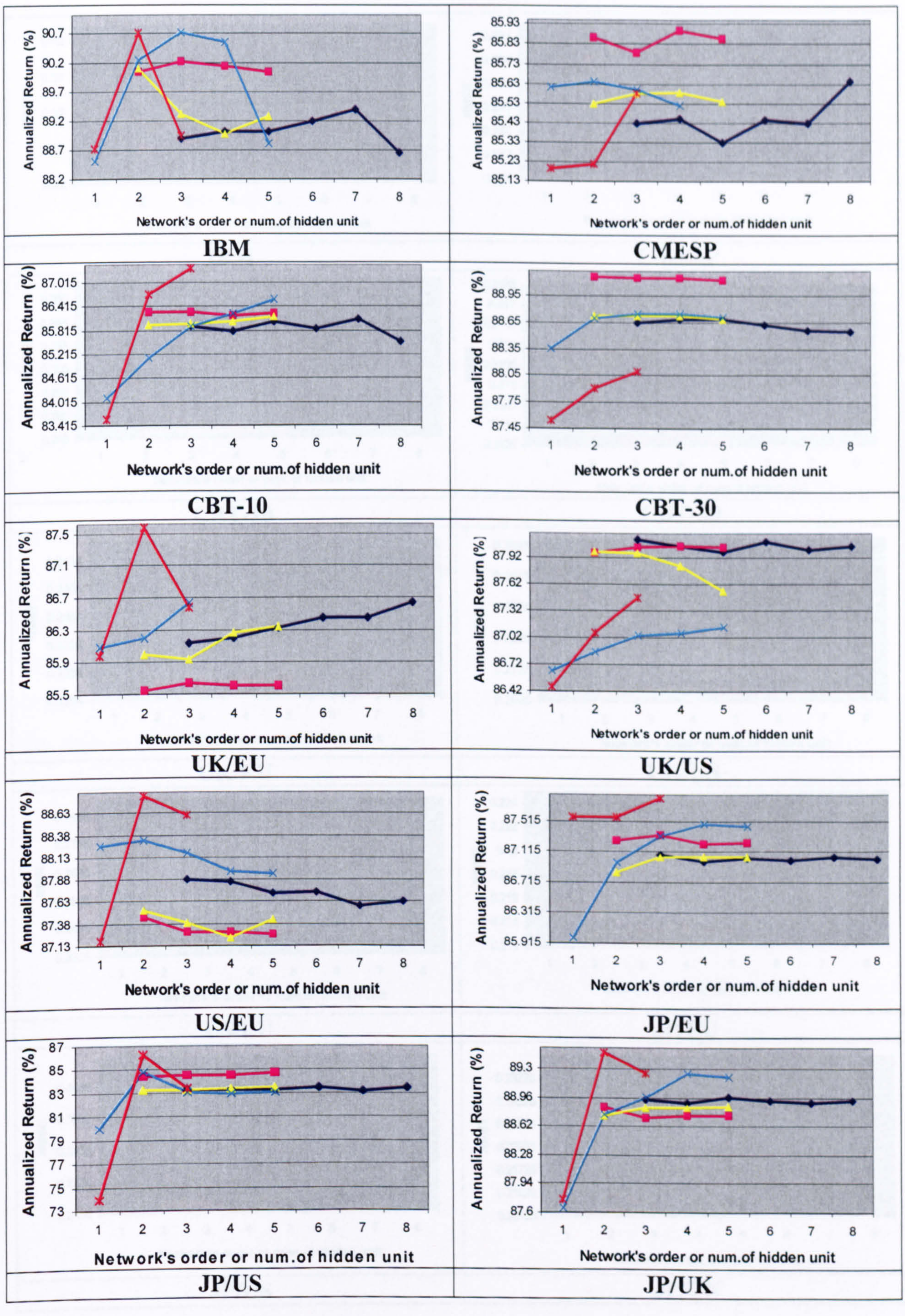


Figure 7.9: Networks' performance on the AR with increasing order / number of hidden nodes

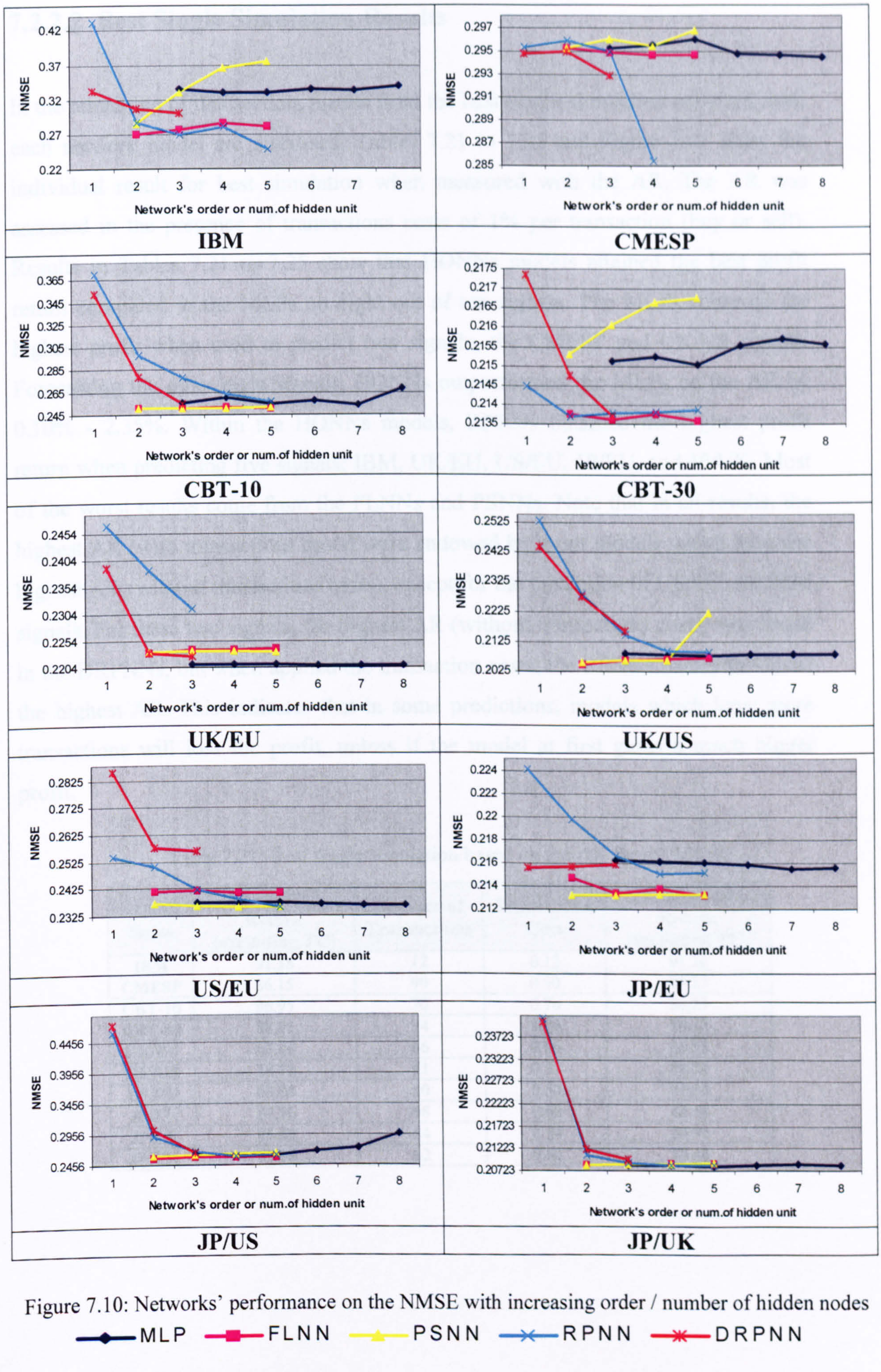


Figure 7.10: Networks' performance on the NMSE with increasing order / number of hidden nodes

7.2.2.2 Best Single Simulation Results

In the remaining of this section, results from the best single simulation achieved from each network model are discussed. Tables 7.21 to 7.25 and Figure 7.11 show the individual result for best simulation when measured with the AR. The AR was assessed in the presence of transactions costs of 1% per transaction (buy or sell). Results in Tables 7.21 to 7.25 show that HONNs models attained the best profit return compared to the MLPs on eight out of ten signals. The MLPs achieved the highest profit when used to predict two signals; the CMESP and UK/US signals. Forecasting the other eight signals, HONNs outperformed the MLPs on the AR by 0.10% - 2.35%. Within the HONNs models, RPNNs obtained the highest profit return when predicting five signals; IBM, UK/EU, US/EU, JP/EU, and JP/UK. Most of the worst results come from the FLNNs and PSNNs. Note that in all results, the highest AR (with transactions costs) were endowed by those models which have the highest AR (without transactions costs), except for the prediction of US/EU and IBM signals. For these two signals, the highest AR (without transactions costs) was found in the DRPNNs, but when applied the transaction costs, the RPNNs finally produced the highest AR. This indicates that in some predictions, models which have more transactions will loss the profit, unless if the model at first gains a much bigger profit.

Table 7.21: Best single simulation based on the AR for the MLPs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	91.38	12	0.12	91.26
CMESP	86.15	90	0.90	85.25
CBT-10	86.95	70	0.70	86.25
CBT-30	88.91	74	0.74	88.17
UK/EU	87.04	66	0.66	86.38
UK/US	88.75	51	0.51	88.24
US/EU	88.04	50	0.50	87.54
JP/EU	87.26	68	0.68	86.58
JP/US	85.50	54	0.54	84.96
JP/UK	89.08	62	0.62	88.46

Table 7.22: Best single simulation based on the AR for the FLNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	91.98	14	0.14	91.84
CMESP	85.93	90	0.90	85.03
CBT-10	86.79	72	0.72	86.07
CBT-30	89.41	76	0.76	88.65
UK/EU	86.28	66	0.66	85.62
UK/US	88.12	50	0.50	87.62
11US/EU	87.59	50	0.50	87.09
JP/EU	87.49	67	0.67	86.82
JP/US	85.04	62	0.62	84.42
JP/UK	88.85	64	0.64	88.21

Table 7.23: Best single simulation based on the AR for the PSNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	90.25	12	0.12	90.13
CMESP	86.03	88	0.88	85.15
CBT-10	87.04	72	0.72	86.32
CBT-30	88.91	74	0.74	88.17
UK/EU	86.37	66	0.66	85.71
UK/US	88.00	50	0.50	87.50
US/EU	87.73	50	0.50	87.23
JP/EU	87.06	68	0.68	86.38
JP/US	83.54	66	0.66	82.88
JP/UK	89.08	62	0.62	88.46

Table 7.24: Best single simulation based on the AR for the RPNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	93.06	12	0.12	92.94
CMESP	86.04	86	0.86	85.18
CBT-10	88.59	70	0.70	87.89
CBT-30	88.94	76	0.76	88.18
UK/EU	88.81	66	0.66	88.15
UK/US	88.26	53	0.53	87.73
US/EU	89.73	42	0.42	89.31
JP/EU	88.64	64	0.64	88.00
JP/US	86.95	56	0.56	86.39
JP/UK	90.14	62	0.62	89.52

Table 7.25: Best single simulation based on the AR for the DRPNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	93.09	16	0.16	92.93
CMESP	86.02	98	0.98	85.04
CBT-10	89.38	78	0.78	88.60
CBT-30	89.05	78	0.78	88.27
UK/EU	88.55	68	0.68	87.87
UK/US	88.73	51	0.51	88.22
US/EU	89.74	50	0.50	89.24
JP/EU	88.13	72	0.72	87.41
JP/US	87.20	60	0.60	86.60
JP/UK	90.07	62	0.62	89.45

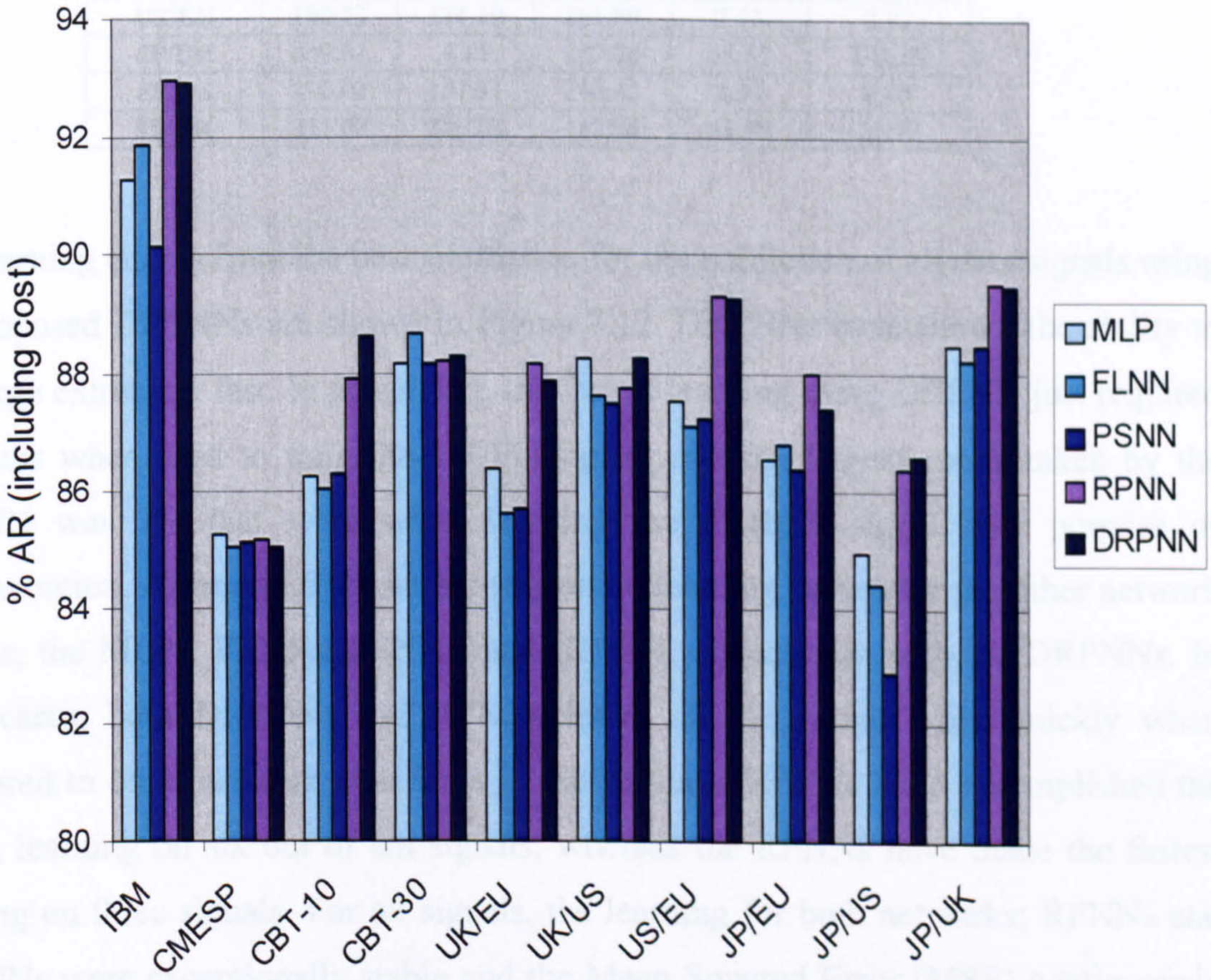


Figure 7.11: Best AR (including the transaction cost)

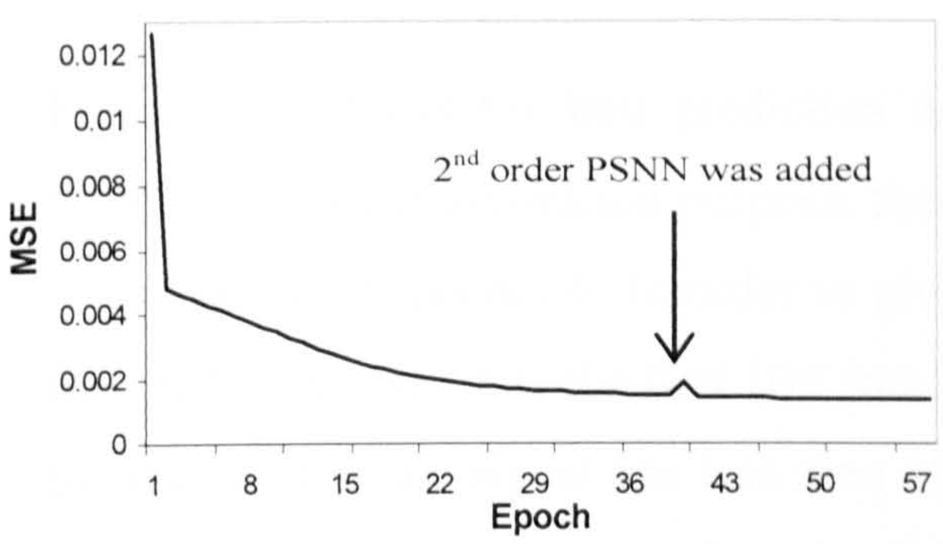
Table 7.26 shows the results of CPU time taken by all neural networks during the training of various signals. Results from the table demonstrated that RPNNs and DRPNNs in most of the cases, took the least CPU time when compared to other neural networks. DRPNNs made the best time when training the IBM, CBT-30, US/EU, and JP/UK signals. Forecasting the four mentioned signals, DRPNNs outperformed other networks by 1.07 to 128.15 time faster in CPU time. RPNNs

took the quickest time when training the CMESP, CBT-10, UK/EU, UK/US, and JP/US signals. Meanwhile, most of the longest CPU times were found in FLNNs, that is when training the IBM, CBT-30, UK/EU, US/EU, and JP/UK signals, followed by MLPs, when training the CMESP, CBT-10, and JP/EU signals.

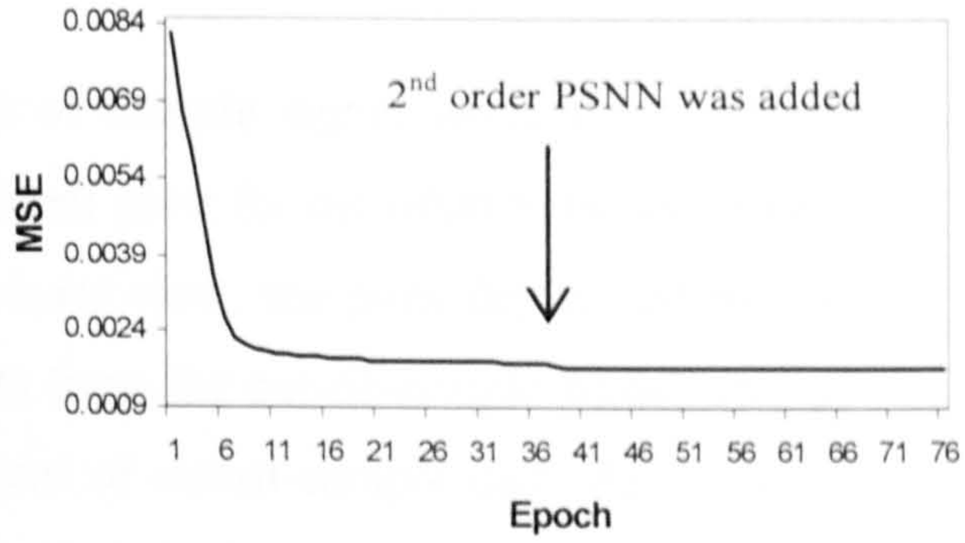
Table 7.26: CPU time usage for training each neural network

Predictor	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	29.38	72.47	31.34	18.75	17.58
CMESP	520.98	63.94	51.75	42.97	60.03
CBT-10	99.30	282.52	33.83	7.30	35.64
CBT-30	286.80	476.55	110.03	66.62	22.06
UK/EU	299.72	376.42	201.72	6.05	38.97
UK/US	288.84	374.89	539.13	12.66	80.50
US/EU	190.23	374.19	261.89	7.13	2.92
JP/EU	649.61	4.11	187.66	63.63	101.08
JP/US	190.02	151.91	247.45	2.31	4.03
JP/UK	311.08	386.09	142.98	51.84	21.77

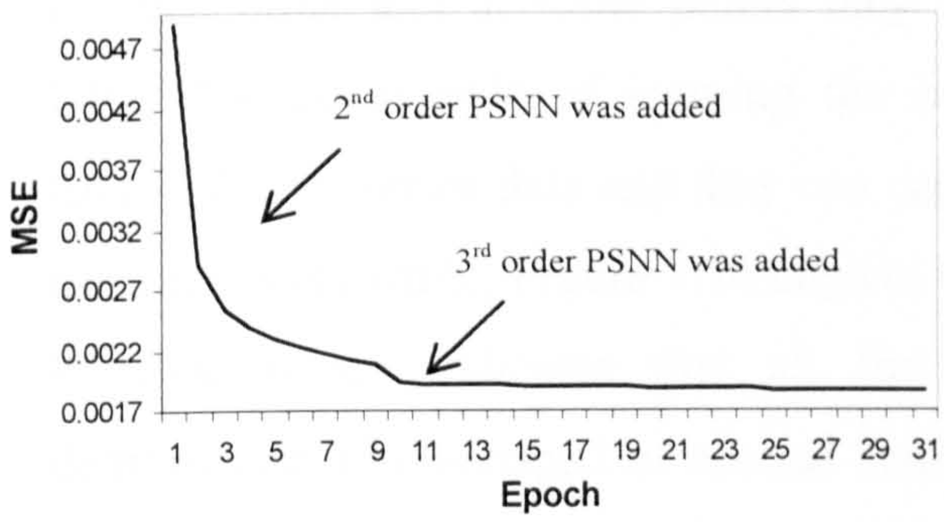
The learning curves from the best simulation for the prediction of all data signals using the proposed DRPNNs are shown in Figure 7.12. DRPNNs have shown the ability to converge extremely fast. In actual fact, the fastest learning using DRPNN just required 7 epochs when used to train the US/EU signal, and the largest epoch taken by the DRPNN was 76, that were when learning the CMESP signal. For purpose of demonstration, Appendix 5 shows the respective learning curves for the other network models; the MLPs, FLNNs, PSNNs, and RPNNs, collectively with the DRPNNs. In most cases, both DRPNNs and RPNNs learnt all the signals very quickly when compared to other network models. It is shown that DRPNNs have accomplished the fastest learning on six out of ten signals, whereas the RPNNs have made the fastest learning on three signals. For all signals, the learning for both networks; RPNNs and DRPNNs were exceptionally stable and the Mean Squared Error (MSE) continuously decreased every time a Pi-Sigma unit of a higher degree is added to the networks. The longest learning was when using the MLP, training the JP/EU signal, and using the FLNNs to train the JP/UK and US/EU signals, which finished off at the maximum epochs of 3000. Recall that the number of maximum epoch pre-determined for training all the networks is 3000. Out of ten signals, the MLPs utilized largest epochs on five of them, namely the CMESP, CBT10, CBT-30, UK/EU, and JP/EU. This is followed by the FLNNs and PSNNs, in which they took largest epochs when learnt three (US/EU, JP/US, JP/UK) and two (IBM, UK/US) signals, respectively.



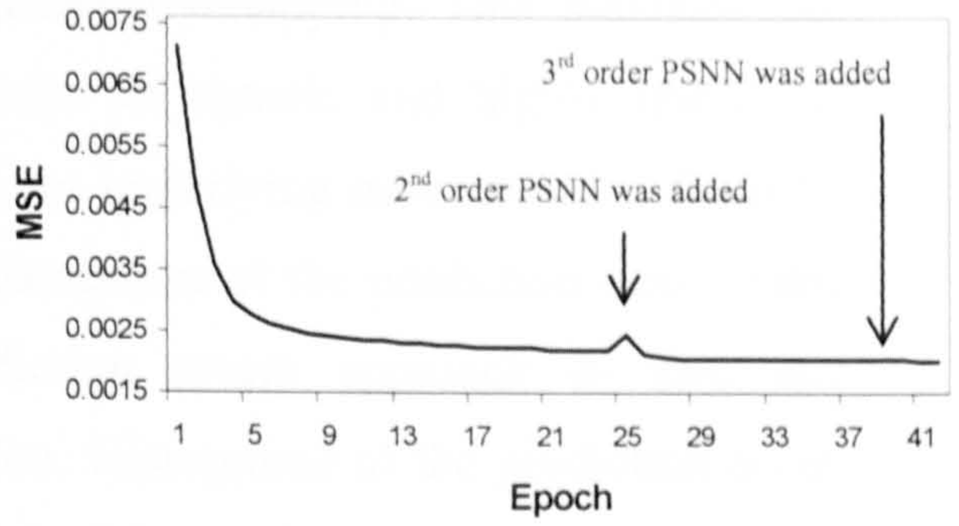
IBM



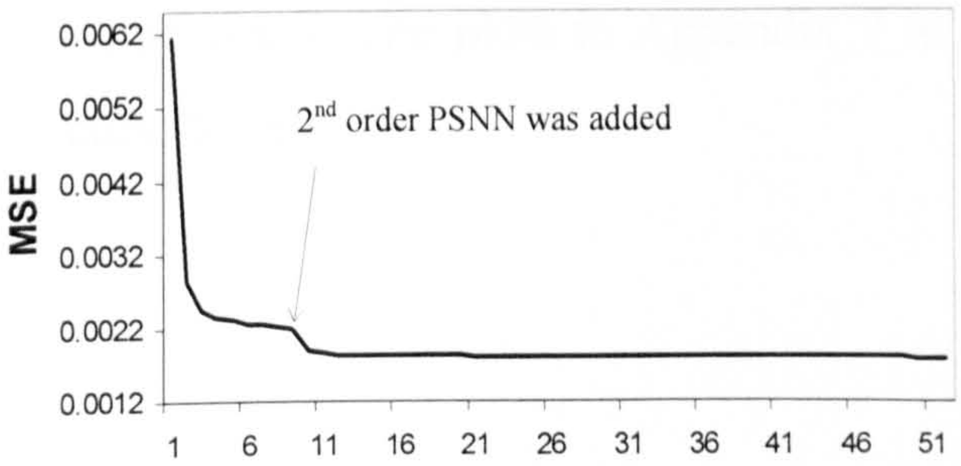
CMESP



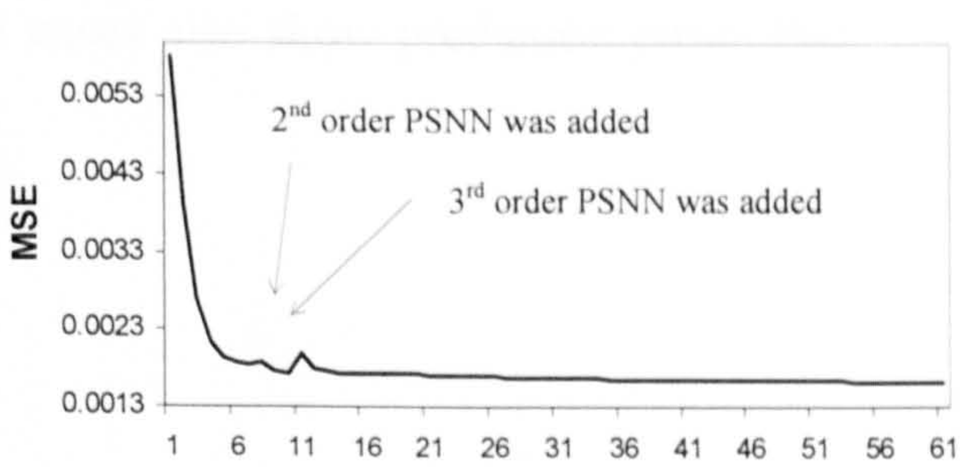
CBT-10



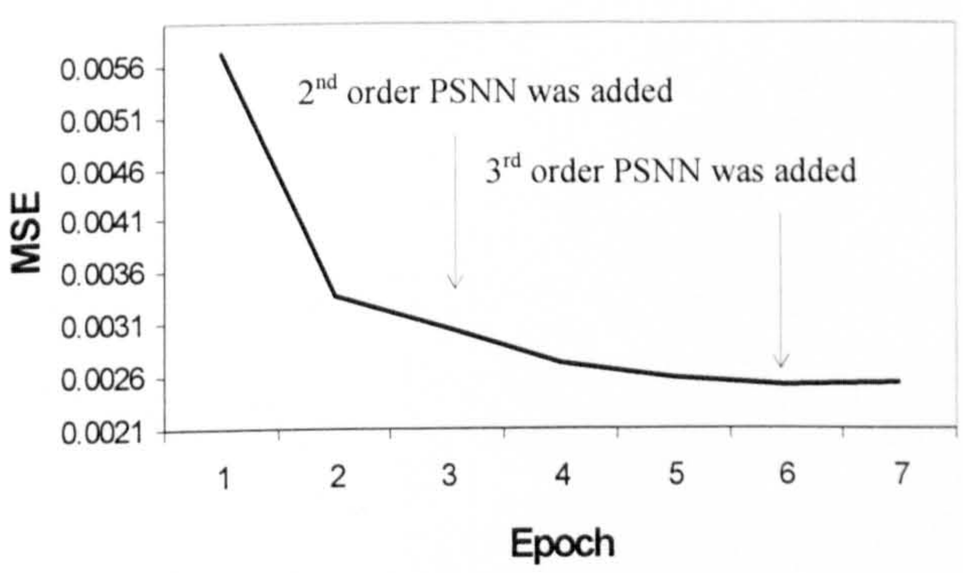
CBT-30



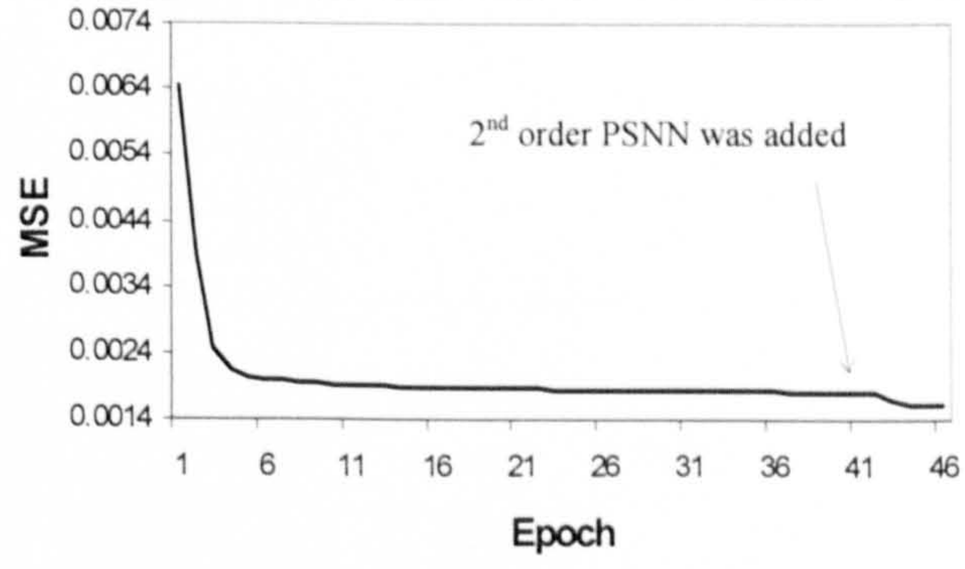
UK/EU



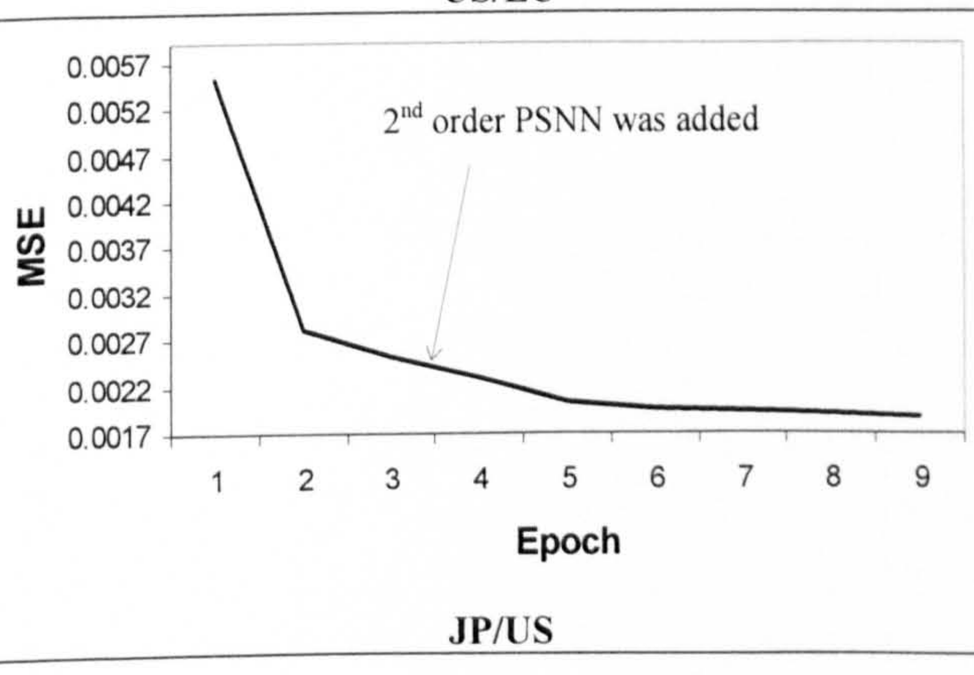
UK/US



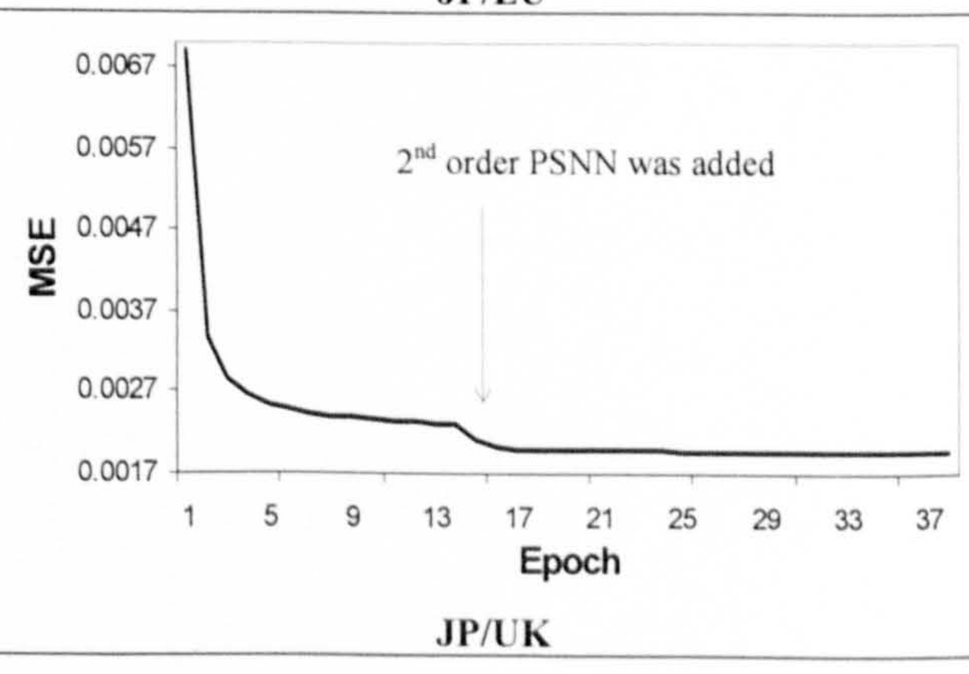
US/EU



JP/EU



JP/US



JP/UK

Figure 7.12: Learning curves for the prediction of all signals using DRPNNs

Figure 7.13 shows the best prediction on out of sample signal using the proposed DRPNNs. For demonstration purpose, the pertinent plots for the other network models are shown in Appendix 6. In order to give a closer view, the plots depict just part of the prediction, which is the first 100 data points from the out-of-sample signal, except for the IBM as the signal has less than 100 point of out-of-sample data. As it can be noticed from Figure 7.13, the plots for the original and predicted signals are very close to each other and at some points they are nearly overlapping. This indicates that DRPNNs are capable of learning the behaviour of chaotic and highly non-linear financial time series data and they can capture the underlying movements in financial markets. Meanwhile, Figure 7.14 depicts the histograms of the prediction errors using RPNNs, which indicates that all the prediction errors approach to zero and demonstrate a bell-shaped of normal distribution. Histograms of the prediction errors for the other four networks; the MLPS, FLNNs, PSNNs, and RPNNs are shown in Appendix 7. The plots in Appendix 7 in most cases also show prediction errors that close to zero.

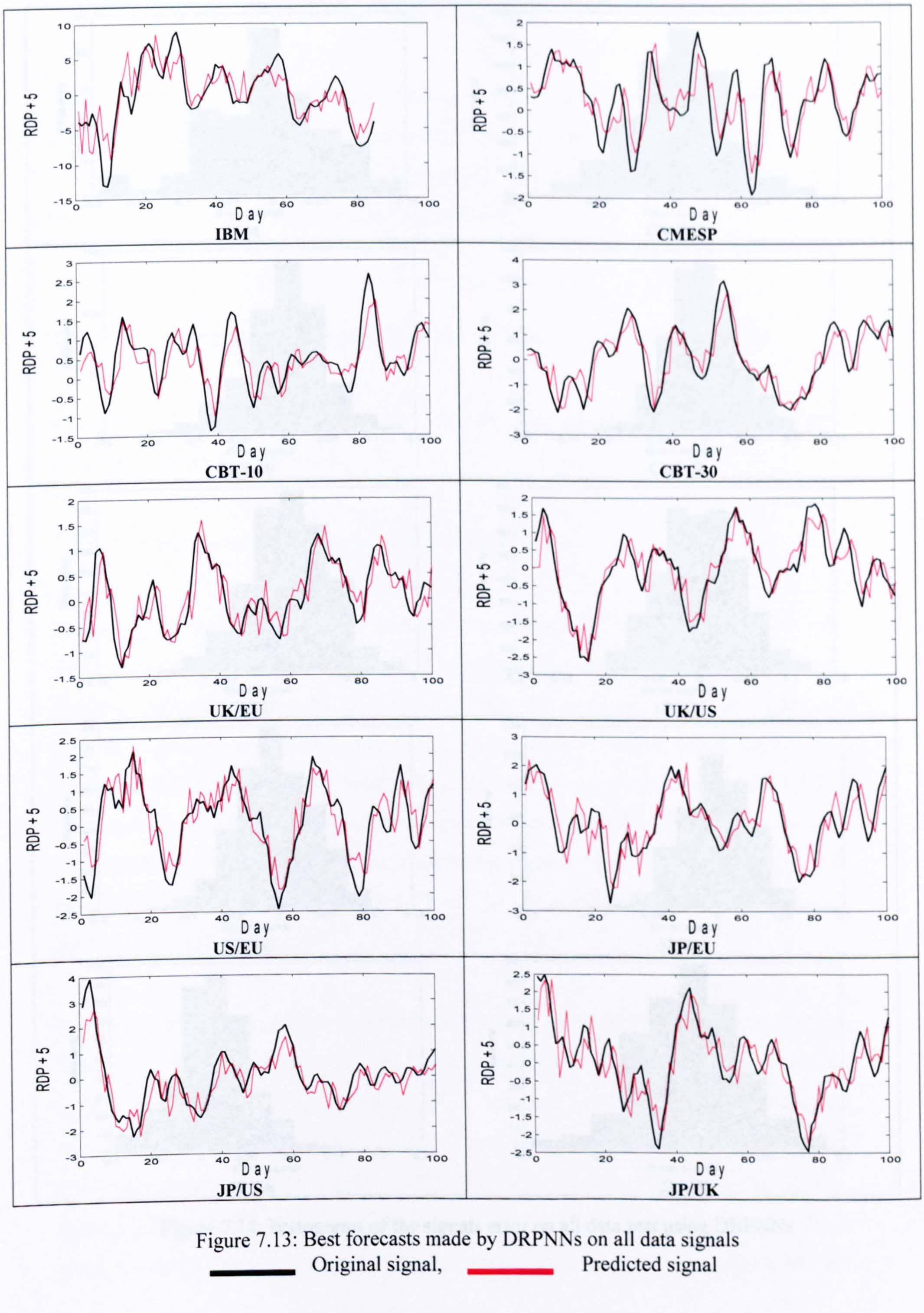


Figure 7.13: Best forecasts made by DRPNNs on all data signals
Original signal, Predicted signal

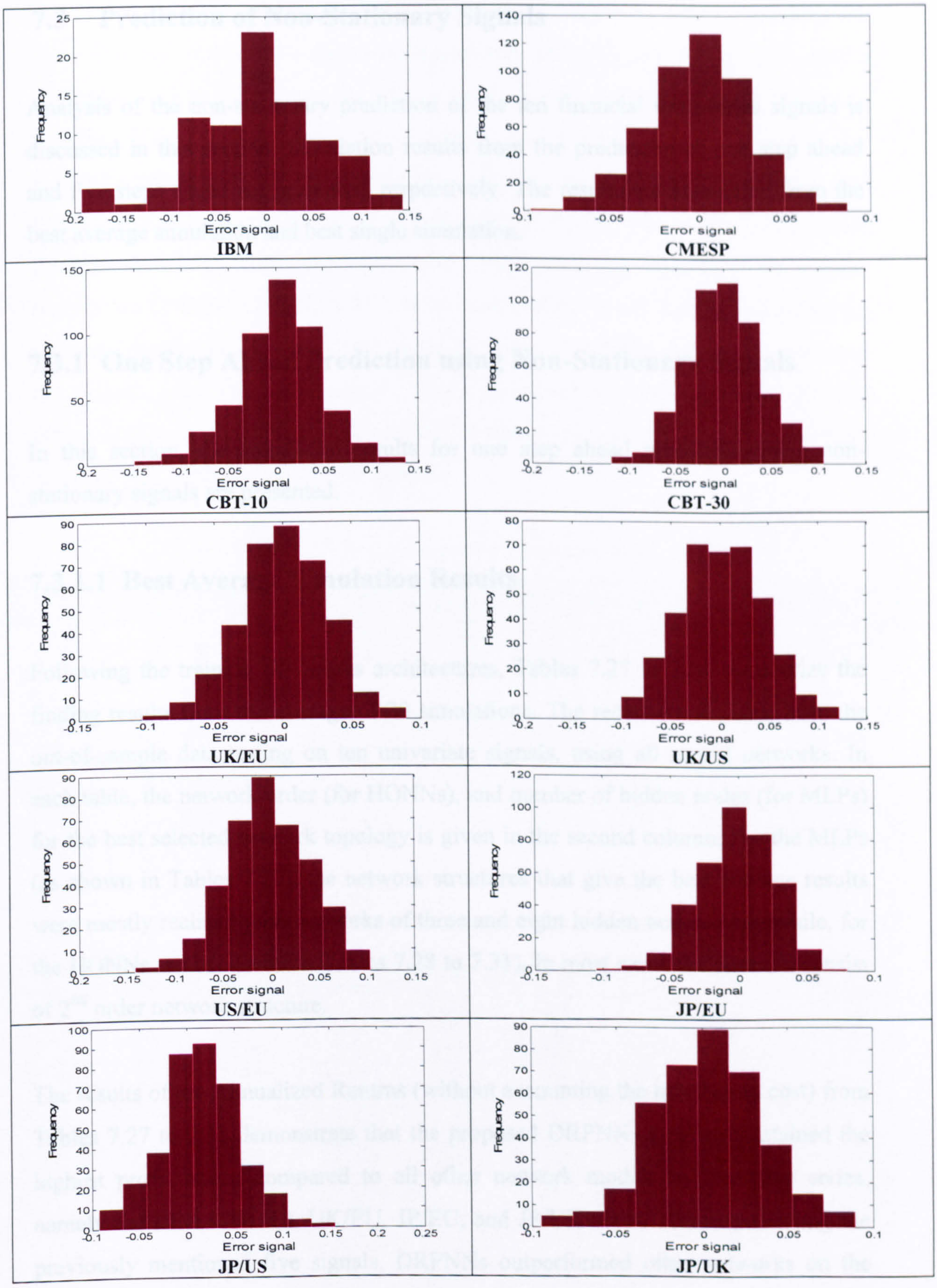


Figure 7.14: Histograms of the signals error on all data sets using DRPNNs

7.3 Prediction of Non-Stationary Signals

Analysis of the non-stationary prediction of the ten financial time-series signals is discussed in this section. Simulation results from the prediction of one step ahead and five steps ahead are presented, respectively. The results are assembled from the best average simulation and best single simulation.

7.3.1 One Step Ahead Prediction using Non-Stationary Signals

In this section, the simulation results for one step ahead prediction using non-stationary signals are presented.

7.3.1.1 Best Average Simulation Results

Following the training of various architectures, Tables 7.27 to 7.31 summarize the finding results from the average of 20 simulations. The results were taken from the out-of-sample data testing on ten univariate signals, using all neural networks. In each table, the network order (for HONNs), and number of hidden nodes (for MLPs) for the best selected network topology is given in the second column. For the MLPs (as shown in Tables 7.27), the network structures that give the best average results were mostly realized with networks of three and eight hidden nodes. Meanwhile, for the HONNs models (refer to Tables 7.28 to 7.31), in most cases the network consist of 2^{nd} order network structure.

The results of the Annualized Returns (without accounting the transaction cost) from Tables 7.27 to 7.31 demonstrate that the proposed DRPNNs profitably attained the highest profit return compared to all other network models in five time series, namely the IBM, CBT-10, UK/EU, JP/EU, and JP/UK signal. When predicting the previously mentioned five signals, DRPNNs outperformed other networks on the average AR by 0.15% to 11.23%. Meanwhile, RPNNs has shown to obtain the best profit return on the CMESP, UK/US, and US/EU signals. Forecasting the CBT-30

and JP/US, FLNNs outperformed other networks on the AR. While evaluating the other financial metrics; the maximum drawdown, volatility and sharpe ratio, results in Tables 7.27 to 7.31 show that the best values were dominant by DRPNNs and RPNNs. When measuring the NMSE, MSE, and SNR, it can be observed that DRPNNs outperform all other networks with the lowest NMSE and MSE, and highest SNR in CBT-10, USEU, JP/US, and JP/UK signals. Meanwhile, MLPs made the highest NMSE, MSE, and lowest SNR when predicting five out of ten signals, namely the CMESP, US/EU, JP/EU, JP/US and JP/UK. In the case of evaluating the Correct Directional Change (CDC), both FLNNs and RPNNs dominantly achieved the highest values in three signals when compared to other networks. FLNNs outperform in the UK/EU, US/EU and JP/US signals, meanwhile RPNNs outperform in the IBM, CMESP, and CBT-30 signals.

Table 7.27: Best average result from the MLPs

Time Series	No.Hidden Nodes	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	3	-4.6122	-20.6499	31.6337	0.1465	0.266283	0.000496	50.63	25.87
CMESP	3	-5.5038	-18.4418	9.2847	0.5930	3.378792	0.008137	48.89	20.82
CBT-10	5	12.1217	-4.1682	6.0213	2.0140	0.027215	0.000167	55.71	35.38
CBT-30	6	2.7592	-8.7013	7.8416	0.3519	0.069532	0.000444	54.7	31.66
UK/EU	8	-5.7302	-12.1393	5.5982	1.0243	0.484508	0.000985	49.72	27.7
UK/US	8	-0.7015	-12.3437	8.2396	0.0852	0.208277	0.000379	51.44	26.17
US/EU	7	0.9642	-8.9431	8.9115	0.1082	2.004158	0.005521	47.22	21.5
JP/EU	3	-2.1583	-10.1293	7.9351	0.2721	12.88734	0.009076	49.89	23.14
JP/US	7	-3.4242	-11.4673	8.6423	0.3963	0.076222	0.000282	48.87	29.52
JP/UK	8	8.0312	-6.6898	7.4047	1.0851	0.384767	0.000393	57.84	32.14

Table 7.28: Best average result from the FLNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	-9.0347	-23.2481	31.5824	0.2870	2.577708	0.004806	50.06	15.67
CMESP	2	-6.3388	-18.9548	9.2881	0.6829	0.03472	0.000084	47.42	39.23
CBT-10	2	9.6093	-5.5482	6.0302	1.5940	0.022697	0.00014	53.64	36.19
CBT-30	2	5.9666	-6.5782	7.8348	0.7619	0.03486	0.000223	54.49	34.83
UK/EU	2	-1.6517	-9.2232	5.5986	0.2961	0.107904	0.000219	52.41	34.06
UK/US	2	-1.2531	-12.5917	8.2401	0.1521	2.228457	0.004054	50.75	15.42
US/EU	2	-0.3741	-10.5707	8.9092	0.0421	0.0916	0.000252	50.04	34.28
JP/EU	5	-0.9701	-8.6543	7.9377	0.1223	0.222269	0.000157	51.53	36.25
JP/US	5	-1.6514	-10.7538	8.6449	0.1911	0.051686	0.000191	49.12	31.18
JP/UK	2	4.7861	-7.4511	7.4123	0.6463	0.152831	0.000156	55.78	36.25

Table 7.29: Best average result from the PSNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	5	-7.5871	-22.3437	31.6257	0.2405	3.817267	0.007117	48.81	16.42
CMESP	5	-6.2391	-20.4375	9.2813	0.6735	1.383479	0.003332	47.89	30.22
CBT-10	2	8.6950	-6.0657	6.0314	1.4428	0.032968	0.000203	52.63	34.62
CBT-30	2	5.6880	-6.5027	7.8356	0.7263	0.086294	0.000551	53.95	31.63
UK/EU	4	-1.3400	-9.2416	5.6010	0.2397	1.690318	0.003435	52.04	29.64
UK/US	4	-0.0728	-13.0318	8.2379	0.0087	1.267532	0.002306	51.6	18.79
US/EU	2	-0.9417	-10.0111	8.9099	0.1057	0.574739	0.001583	48.4	31.2
JP/EU	3	-0.7070	-9.3511	7.9352	0.0891	0.294869	0.000208	51.61	35.64
JP/US	2	-2.9656	-12.5309	8.6432	0.3431	0.059423	0.00022	48.84	30.58
JP/UK	3	5.5543	-7.0358	7.4110	0.7500	0.221689	0.000227	56.24	34.7

Table 7.30: Best average result from the RPNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	0.4641	-17.6616	31.6729	0.0146	1.34883	0.002515	51.59	23.69
CMESP	4	7.8734	-12.4685	9.2720	0.8511	0.146351	0.000352	52.48	33.2
CBT-10	2	11.6105	-4.5959	6.0236	1.9280	0.025674	0.000158	55.3	35.74
CBT-30	3	5.6633	-6.5684	7.8345	0.7235	0.047035	0.0003	54.8	34.45
UK/EU	5	-0.7927	-9.1858	5.6016	0.1414	0.076542	0.000156	52.07	35.72
UK/US	2	1.7961	-10.2990	8.2351	0.2183	0.520431	0.000947	51.36	22.79
US/EU	4	1.0510	-10.0593	8.9042	0.1181	0.173251	0.000477	49.17	32.65
JP/EU	2	-0.8266	-8.1307	7.9364	0.1043	0.174064	0.000123	50.77	37.25
JP/US	5	-4.1463	-13.6355	8.6389	0.4803	0.043948	0.000163	48.54	31.93
JP/UK	3	7.7855	-6.3860	7.4059	1.0518	0.139066	0.000142	57.18	36.62

Table 7.31: Best average result from the DRPNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	2	2.1946	-16.5205	31.6604	0.0696	12.23239	0.022807	51.59	10.07
CMESP	3	6.3732	-13.9709	9.2820	0.6874	0.07396	0.000178	51.79	35.72
CBT-10	3	13.9853	-4.4815	6.0140	2.3263	0.019101	0.000118	54.64	36.91
CBT-30	2	5.0175	-7.4161	7.8362	0.6407	0.043104	0.000275	54.51	34.31
UK/EU	2	-0.2960	-9.1664	5.6023	0.0533	0.083246	0.000169	51.48	35.27
UK/US	3	1.2960	-9.4577	8.2363	0.1575	0.850463	0.001547	50.54	21.15
US/EU	2	0.0837	-8.8003	8.9106	0.0093	0.076884	0.000212	46.83	34.93
JP/EU	2	-0.5587	-9.1767	7.9332	0.0703	0.215175	0.000152	51.52	36.54
JP/US	2	-2.3702	-11.3892	8.6422	0.2743	0.038587	0.000143	48.94	32.46
JP/UK	2	8.7788	-6.5637	7.4034	1.1860	0.10447	0.000107	57.78	37.78

The annualized return achieved in all network models as given in Tables 7.27 to 7.31 is depicted in Figure 7.15. Subsequently, the average number of epochs reached for the prediction of all data signals during the training of the networks is shown in Table 7.32. For the prediction of IBM, UK/EU, JP/EU and JP/US signals, results given in Table 7.32 show that the proposed DRPNNs reveal to use least number of epochs to converge during the training, which is about 1.10 to 375.00 times faster than other networks. RPNNs on the other hand have shown to converge fastest when

used to predict the UK/US signal. For the prediction of CBT-10 and CBT-30, PSNNs appeared to utilize least epochs compared to other network models, while FLNNs have shown to use least epoch when used to learn the CMESP, US/EU, and JP/UK signals. Forecasting the IBM and JP/US signals, both MLPs and FLNNs reached the maximum number of epochs for training the network which was set to 3000. Besides, FLNN also used maximum epochs to learn the UK/US signal. Out of ten signals, MLPs have shown to require larger number of epochs to complete the whole training, except for the prediction of CMESP and UK/US.

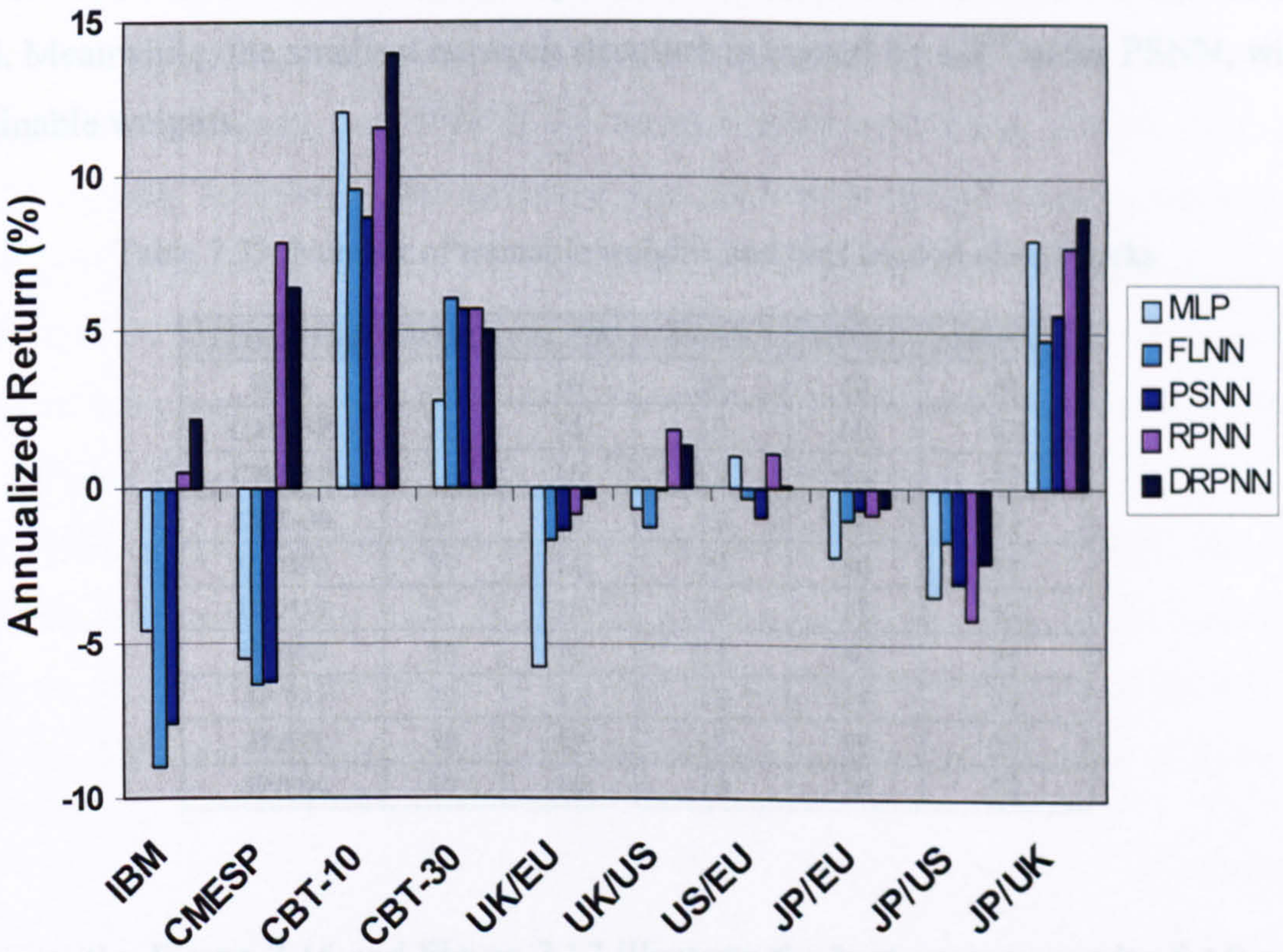


Figure 7.15: Best average annualized return from all network models

Table 7.32: The average maximum epoch reached during training

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	3000	3000	2493	61	8
CMESP	95	19	94	2173	2071
CBT-10	2724	186	98	494	1789
CBT-30	2861	89	38	58	46
UK/EU	2067	24	31	1377	22
UK/US	1443	3000	2759	26	102
US/EU	2353	27	198	249	947
JP/EU	905	20	21	60	19
JP/US	3000	3000	2564	1899	730
JP/UK	2697	125	562	1325	2656

Table 7.33 demonstrates the results of the network size, specifically on the average number of trainable weights and bias utilized in all network models. Evaluating the number of free parameters, it appears that most of the smallest network structures are dominated by the FLNNs and PSNNS. The RPNNs obviously comprise of larger network size when learning the CMESP, UK/EU, US/EU, and JP/US time series. While the training of the MLPs showed that the networks comprise of larger number of trainable weights and bias when used to learn the CBT-30, UK/US, and JP/UK signals. From the tabulated results, the biggest network structure is possessed by a 5^{th} order RPNN, which contains 90 free parameters when used to forecast the JP/US signal. Meanwhile, the smallest network structure is owned by a 2^{nd} order PSNN, with 12 trainable weights.

Table 7.33: Number of trainable weights and bias used in all networks

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	22	16	30	18	21
CMESP	22	16	30	60	42
CBT-10	36	16	12	18	42
CBT-30	43	16	12	36	21
UK/EU	57	16	24	90	21
UK/US	57	16	24	18	42
US/EU	50	16	12	60	21
JP/EU	22	32	18	18	21
JP/US	50	32	12	90	21
JP/UK	57	16	18	36	21

Subsequently, Figure 7.16 and Figure 7.17 illustrate the best average result of AR and NMSE, respectively, tested on unseen data, when used to predict the financial signals. In order to test the modelling capabilities and the stability of all network models, the performance of the networks was assessed with the number of higher order terms increased from 1 to 5 (for HONNs), and the number of hidden nodes increased from 3 to 8 (for MLPs). The plots in Figure 7.16 show that the performance of the proposed network, DRPNNs, steadily continue to increase along with the network growth when used to predict the CMESP, UK/EU, JP/EU, and JP/UK signals. For the prediction of IBM, US/EU, CBT-30, and JP/US signals, the performance of DRPNNs continue to rise when a 2^{nd} order Pi-Sigma unit was added to the networks, and the AR began to drop when the 3^{rd} order Pi-Sigma unit was added. Conversely, for the prediction of

UK/US and CBT-10 signals, the AR for DRPNNs were dropping when a 2nd order Pi-Sigma unit was added to the networks, and the AR began to increase when a 3rd order Pi-Sigma unit was added. In the case of evaluating the performance of RPNNs, the AR for the prediction of CMESP, JP/US, and JP/UK signals keep increasing until network of order five. Meanwhile, for the prediction of the IBM, JP/EU and CBT-30 signals, RPNNs exhibit an increment in the AR, and their performance then start to degrade when they reached network of order three or four. For the remaining of the signals, the plots demonstrate an up and down movement of the AR, except for the CBT-10 series, where the AR kept on decreasing until network of order five. For FLNNs and PSNNs, the networks in some cases reveal decreasing AR (FLNNs for the prediction of IBM, UK/US, CBT-10 signals, and PSNNs for the prediction of US/EU, JP/US, CBT-10, CBT-30 signals). Meanwhile, the networks also showed an up and down movement in the AR performance (FLNNs for the prediction of CMESP, UK/EU, JP/UK, CBT-30 signals, and PSNNs for the prediction of UK/US, UK/EU, JP/EU, JP/UK signals). FLNNs in some cases showed increased AR only after a 5th order network structure were utilized. This can be seen when the networks were used to forecast the US/EU, JP/US, JP/EU signals. For the prediction of IBM and CMESP signals, PSNNs steadily showed an increasing in the AR. Meanwhile, the performances of the MLPs generally exhibit an up and down movement in the AR, except for the prediction of UK/EU in which the AR keep increasing from network structure with hidden node of three to eight.

Following the performance of AR, Figure 7.17 depicts the average performance of the NMSE with increasing networks order or number of hidden nodes. For the prediction of UK/EU, JP/US, JP/EU, JP/UK, CBT-10 signals, the plots for DRPNNs demonstrate continuously decreasing NMSE along with the network growth. Meanwhile, the performance of the DRPNNs when used to predict the CMESP and US/EU signals demonstrate an increment NMSE until evolving 3rd order networks. For the prediction of IBM, the NMSE of DRPNN continued to rise when a 2nd order Pi-Sigma unit was added to the networks, and started to drop when the 3rd order Pi-Sigma unit was added. Conversely, for the prediction of CBT-30 and UK/US signals, the NMSE for DRPNNs were dropping when 2nd order Pi-Sigma unit was added to the networks, and began to increase when the 3rd order Pi-Sigma unit was added.

Forecasting the signals using RPNNs shows that in most cases, the NMSE steadily decreased along with the networks growth. On the whole, PSNNs and FLNNs on the other hand, reveal to perform increasing NMSE. Forecasting the IBM, CMESP, UK/US, US/EU, and JP/US signals, the plots for the MLPs demonstrate a zigzag motion of NMSE. Meanwhile, when predicting the UK/EU, JP/EU, JP/UK, and CBT-10 signals, the performance of MLPs show decreased NMSE, and when the number of hidden nodes was expanded, the NMSE began to rise. Finally, the plot for MLP when used to predict the CBT-30 signal reveals that the NMSE continued to increase with the network size.

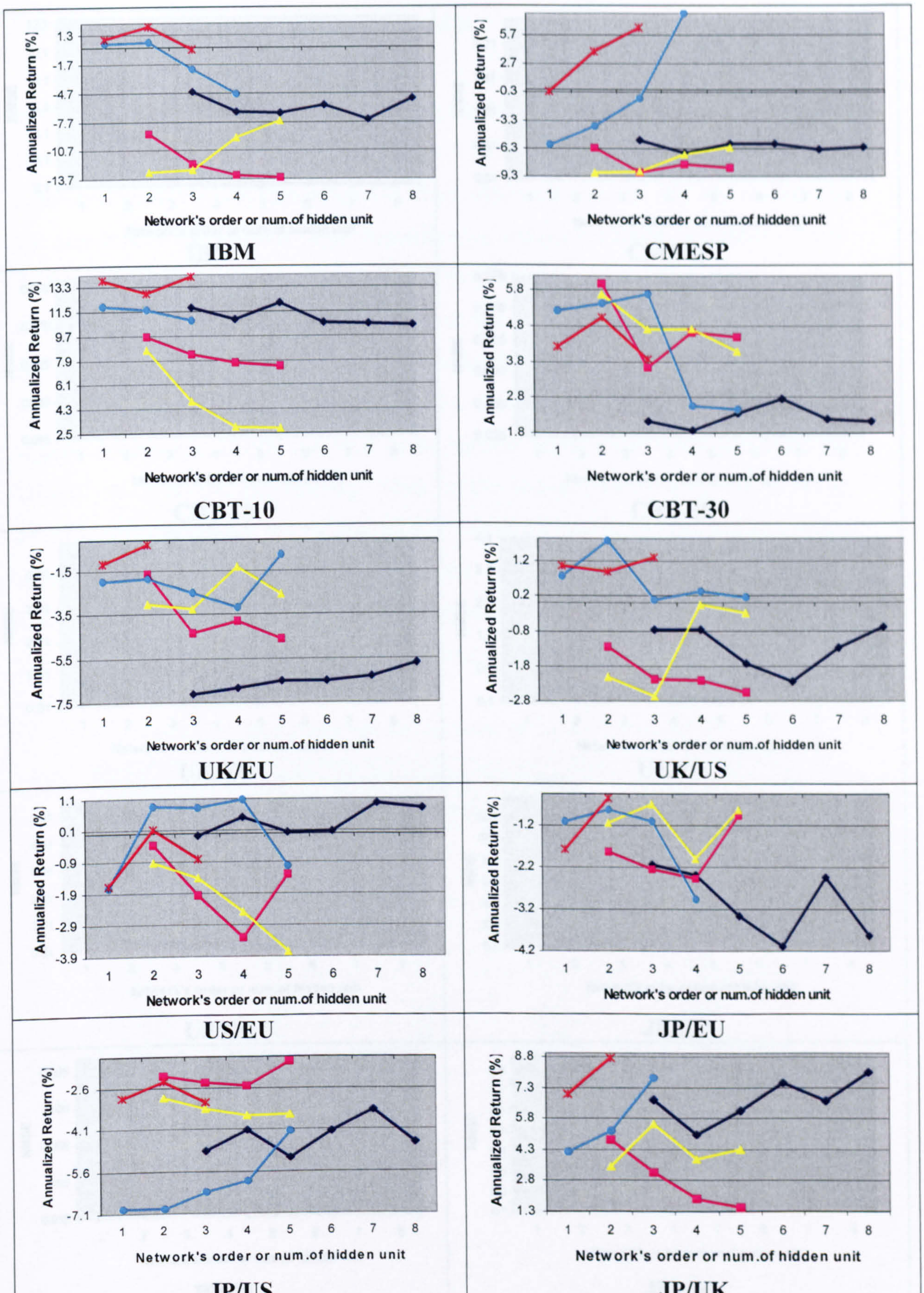


Figure 7.16: Networks' performance on the AR with increasing order / number of hidden nodes

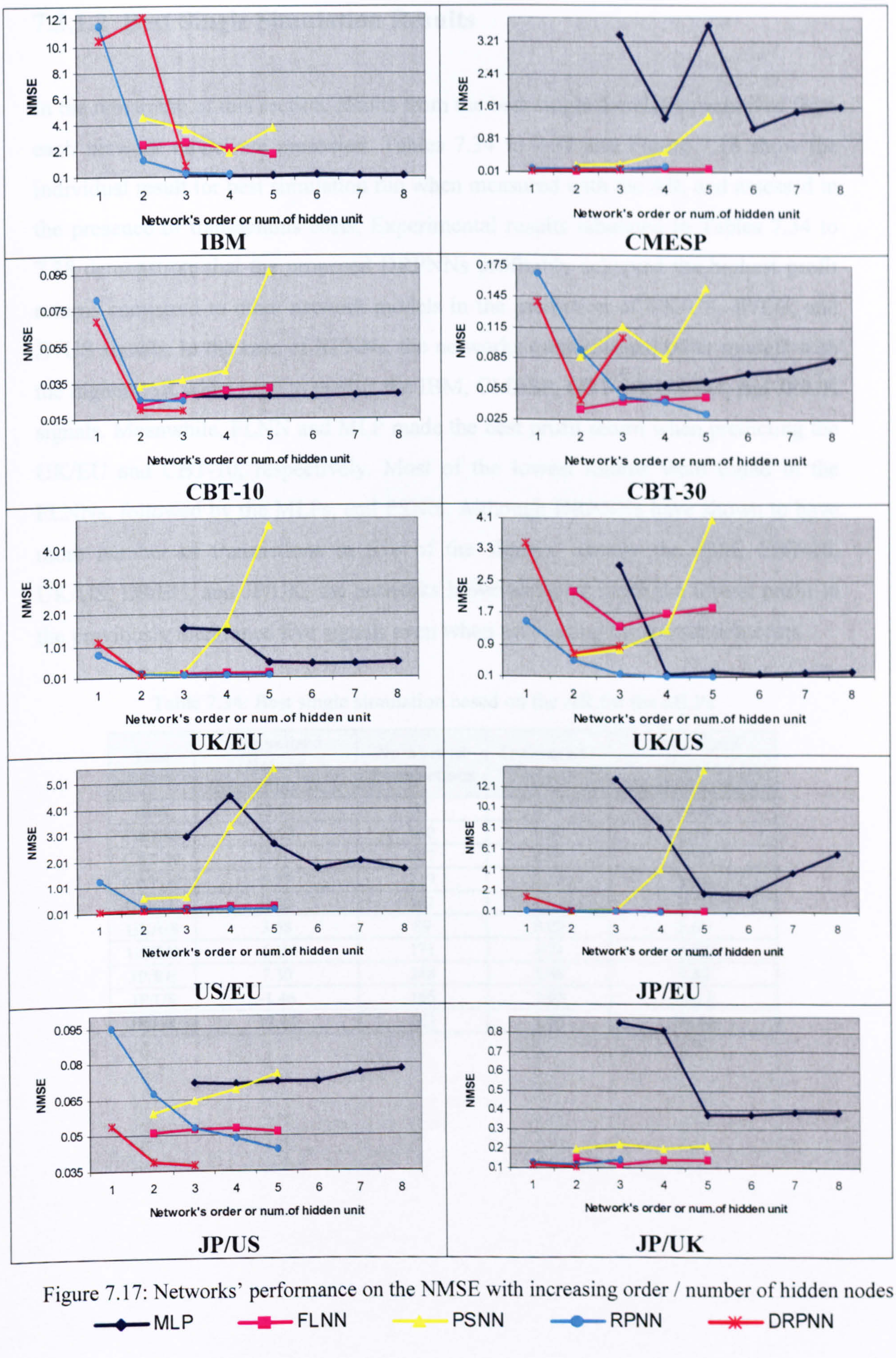


Figure 7.17: Networks' performance on the NMSE with increasing order / number of hidden nodes

7.3.1.2 Best Single Simulation Results

In the remaining of this section, results from the best single simulation achieved from each network model are presented. Tables 7.34 to 7.38 and Figure 7.18 show the individual result for best simulation run when measured with the AR, and assessed in the presence of transactions costs. Experimental results tabulated in Tables 7.34 to 7.38 demonstrate that the proposed DRPNNs profitably achieved the highest profit returns compared to other network models in the prediction of UK/US, JP/EU, and JP/US signals. In the case of RPNNs, the networks outperformed other models with the highest AR when used to predict the IBM, CMESP, CBT-30, US/EU, and JP/UK signals. Meanwhile, FLNN and MLP made the best profit return when predicting the UK/EU and CBT-10, respectively. Most of the lowest returns were found in the FLNNs, followed by the MLPs, and PSNN. Although DRPNNs have shown to have more number of transactions in five of the signals, namely the IBM, CBT-10, UK/US, US/EU, and JP/UK, the networks however never made the lowest profit in the previously mentioned five signals even when accounting the transaction costs.

Table 7.34: Best single simulation based on the AR for the MLPs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	13.55	37	0.37	13.18
CMESP	10.23	360	3.60	6.63
CBT-10	17.02	207	2.07	14.95
CBT-30	5.77	219	2.19	3.58
UK/EU	0.82	161	1.61	-0.79
UK/US	3.58	89	0.89	2.69
US/EU	6.19	171	1.71	4.48
JP/EU	7.30	248	2.48	4.82
JP/US	-1.46	165	1.65	-3.11
JP/UK	12.13	163	1.63	10.50

Table 7.35: Best single simulation based on the AR for the FLNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	4.37	43	0.43	3.94
CMESP	1.11	138	1.38	-0.27
CBT-10	14.75	168	1.68	13.07
CBT-30	12.46	142	1.42	11.04
UK/EU	10.95	148	1.48	9.47
UK/US	3.31	187	1.87	1.44
11US/EU	5.56	175	1.75	3.81
JP/EU	3.09	155	1.55	1.54
JP/US	0.85	169	1.69	-0.84
JP/UK	10.37	104	1.04	9.33

Table 7.36: Best single simulation based on the AR for the PSNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	2.31	31	0.31	2.00
CMESP	5.46	180	1.80	3.66
CBT-10	16.71	210	2.10	14.61
CBT-30	14.19	153	1.53	12.66
UK/EU	7.36	138	1.38	5.98
UK/US	9.46	181	1.81	7.66
US/EU	8.08	99	0.99	7.09
JP/EU	8.44	141	1.41	7.03
JP/US	1.17	190	1.90	-0.73
JP/UK	10.56	177	1.77	8.79

Table 7.37: Best single simulation based on the AR for the RPNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	18.34	54	0.54	17.80
CMESP	20.73	188	1.88	18.85
CBT-10	17.53	251	2.51	15.02
CBT-30	15.68	182	1.82	13.86
UK/EU	9.72	189	1.89	7.83
UK/US	9.30	153	1.53	7.77
US/EU	12.71	102	1.02	11.69
JP/EU	6.51	103	1.03	5.48
JP/US	1.88	176	1.76	0.12
JP/UK	14.33	177	1.77	12.56

Table 7.38: Best single simulation based on the AR for the DRPNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	16.23	59	0.59	15.64
CMESP	17.44	233	2.33	15.11
CBT-10	17.48	256	2.56	14.92
CBT-30	12.39	160	1.60	10.79
UK/EU	9.62	188	1.88	7.74
UK/US	12.49	238	2.38	10.11
US/EU	6.43	179	1.79	4.64
JP/EU	16.23	145	1.45	14.78
JP/US	5.07	184	1.84	3.23
JP/UK	12.95	178	1.78	11.17

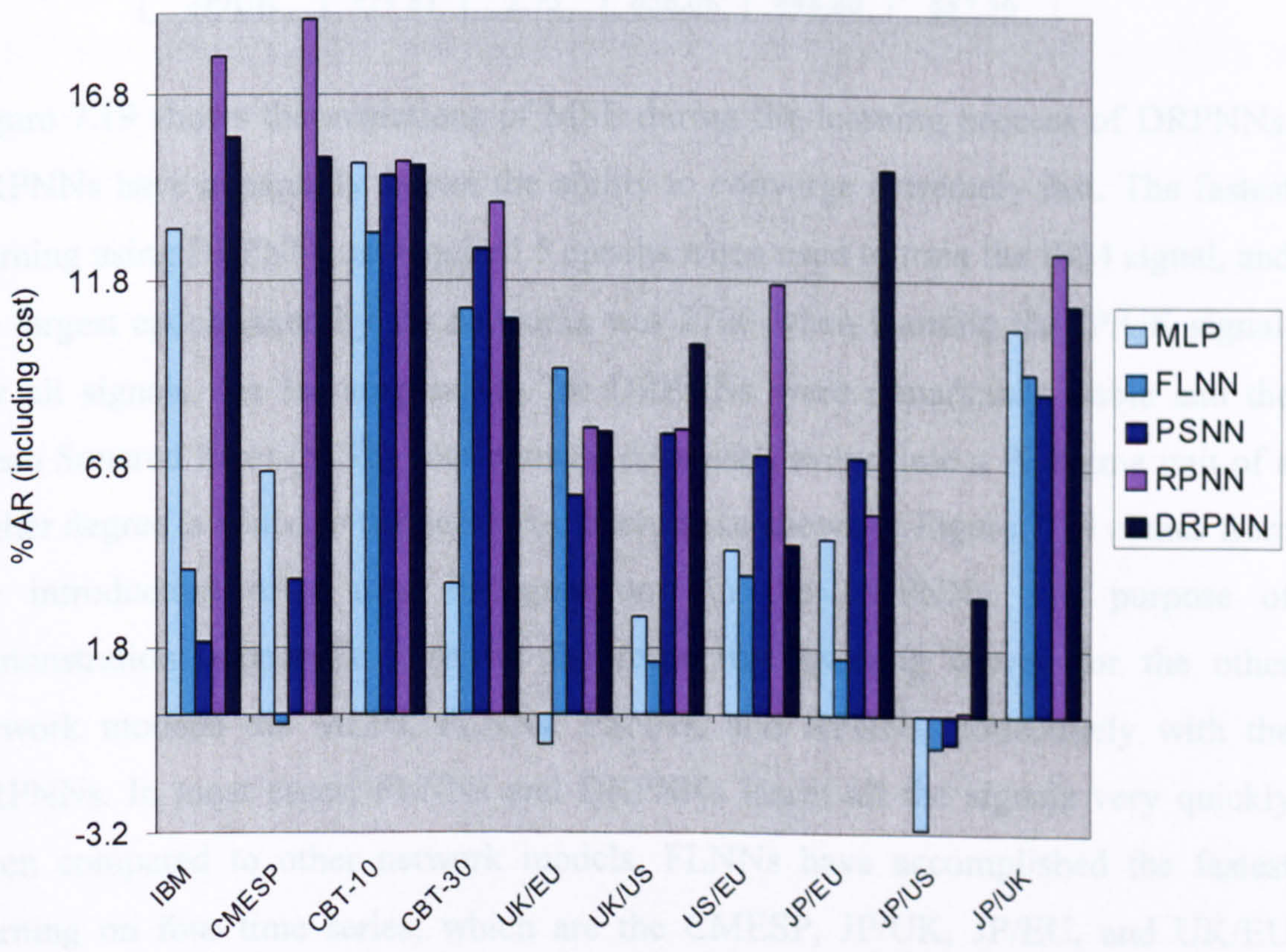


Figure 7.18: Best AR (including the transaction cost)

Table 7.39 shows the amount of CPU time utilized by all neural networks during the training of various signals. Results from the table demonstrated that FLNNs broadly used the least CPU time when compared to other neural networks in all signals, except for the training of IBM and UK/US signals. The networks appeared to outperform other neural networks with a speed of 1.29 to 322.68 faster in CPU time.

RPNNs took the quickest time when training the IBM and UK/US signals. Meanwhile, most of the longest CPU times were found in MLPs, that is when training the IBM, CBT-30, UK/EU, US/EU, and JP/UK signals.

Table 7.39: CPU time usage for training each neural network

Predictor	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	161.92	104.27	153.05	4.61	4.97
CMESP	33.20	4.42	5.69	57.38	603.89
CBT-10	563.73	29.02	68.13	153.19	289.59
CBT-30	869.25	4.17	13.39	10.92	17.53
UK/EU	520.59	4.05	4.70	66.36	12.02
UK/US	611.55	395.36	642.53	8.98	57.77
US/EU	728.47	3.45	7.64	29.39	193.14
JP/EU	347.19	2.72	4.00	7.98	12.80
JP/US	690.28	387.39	597.66	671.97	619.37
JP/UK	713.83	2.75	658.08	448.80	887.36

Figure 7.19 shows the evolutions of MSE during the learning process of DRPNNs. DRPNNs have apparently shown the ability to converge extremely fast. The fastest learning using DRPNN just required 5 epochs when used to train the IBM signal, and the largest epoch taken by the networks was 2700 when learning the JP/UK signal. For all signals, the learning curves for DRPNNs were remarkably stable and the Mean Squared Error (MSE) continuously decreased every time a Pi-Sigma unit of a higher degree is added to the networks. Each spike shown in Figure 7.19 comes from the introduction of a new Pi-Sigma unit in the DRPNNs. For purpose of demonstration, Appendix 8 shows the respective learning curves for the other network models; the MLPs, FLNNs, PSNNs, and RPNNs, collectively with the DRPNNs. In most cases, FLNNs and DRPNNs learnt all the signals very quickly when compared to other network models. FLNNs have accomplished the fastest learning on four time series, which are the CMESP, JP/UK, JP/EU, and UK/EU signals. Meanwhile, DRPNNs converged fastest when used to predict the IBM, JP/US, and UK/US signals. Apart from the prediction of the CMESP and UK/US signals, MLPs appeared to utilize the largest epochs during the training, and the networks revealed to reach maximum number of pre-determined epoch when learning the IBM, CBT-10, JP/UK, and JP/US signals.

The best prediction on all signals using the proposed DRPNNs is depicted in Figure 7.20. The plots show the first 100 data points from the unseen data, except for the

IBM as the signal has less than 100 point of unseen data. For demonstration purpose, the applicable plots for the other network models are shown in Appendix 9. Notice that when compared to the stationary signals, the plots in Figure 7.20 and Appendix 9 indicate that for some financial time series, the non-stationary signals are harder to predict. This can be viewed at some data points, in which the original and predicted signals are quite distant from each other. Meanwhile, Figure 7.21 presents the histograms of the prediction errors using DRPNNs, which signifies that most of the prediction errors are close to zero, except for the prediction of the IBM, UK/EU, and UK/US signals. The pertinent histograms of the other four networks; the MLPS, FLNNs, PSNNs, and RPNNs are shown in Appendix 10. Histograms in Appendix 10 reveal that MLPs and PSNNs in most cases demonstrate prediction errors that far off zero. This suggests that the non-stationary time series signals are harder to predict, as compared to the previously presented stationary signals.

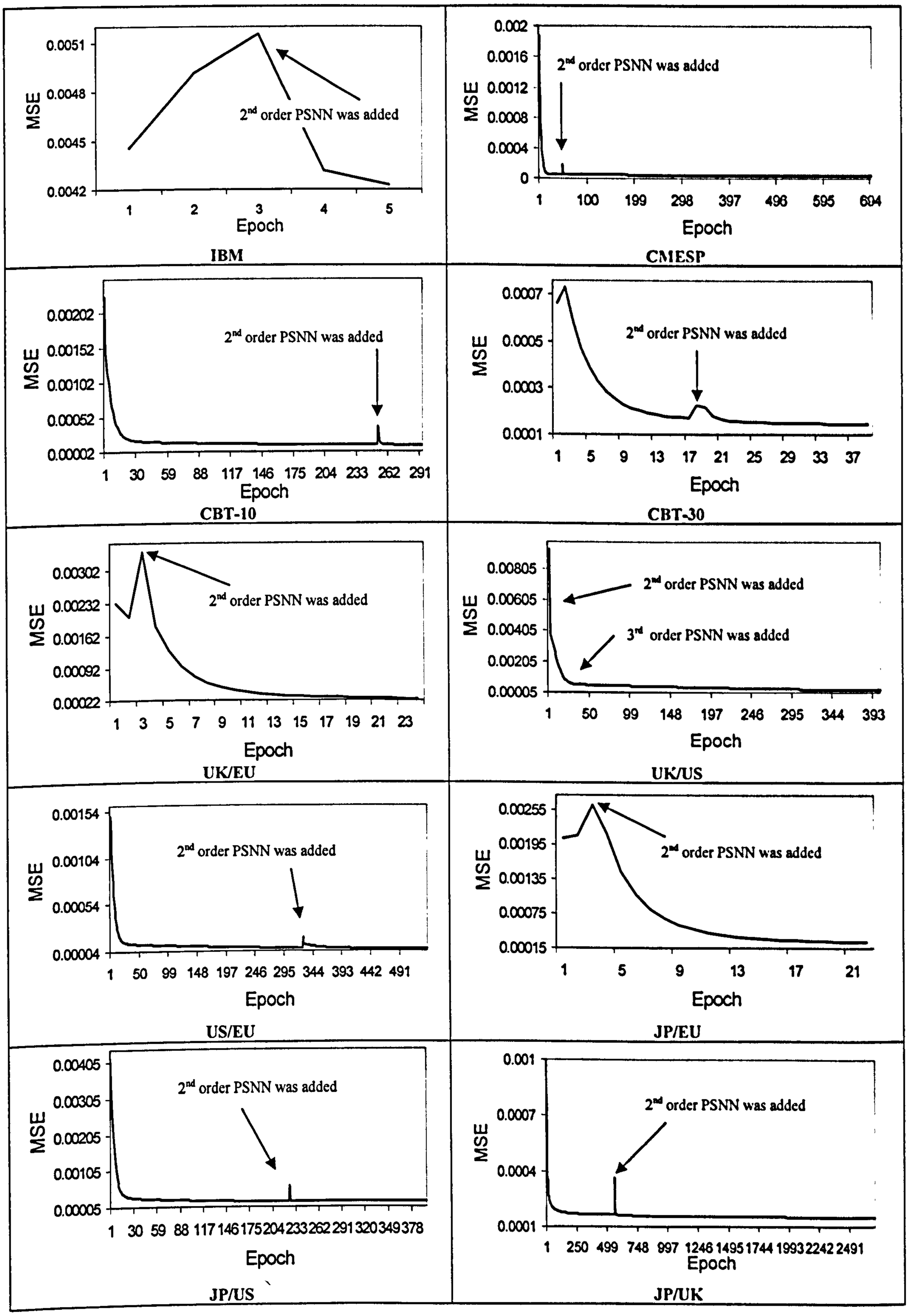


Figure 7.19: Learning curves for the prediction of all signals using DRPNNs

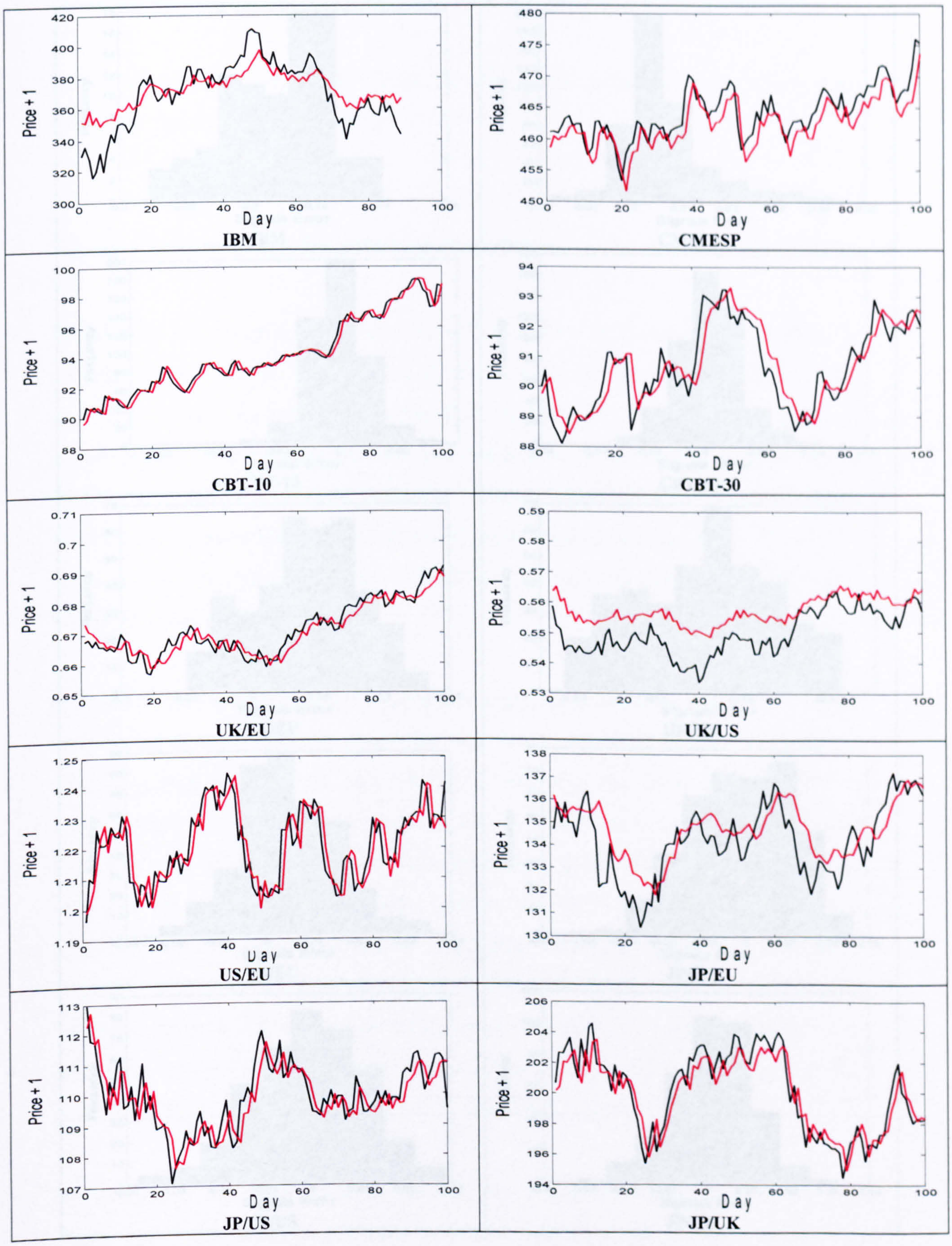


Figure 7.20: Best forecasts made by DRPNNs on all data signals

Original signal, Predicted signal

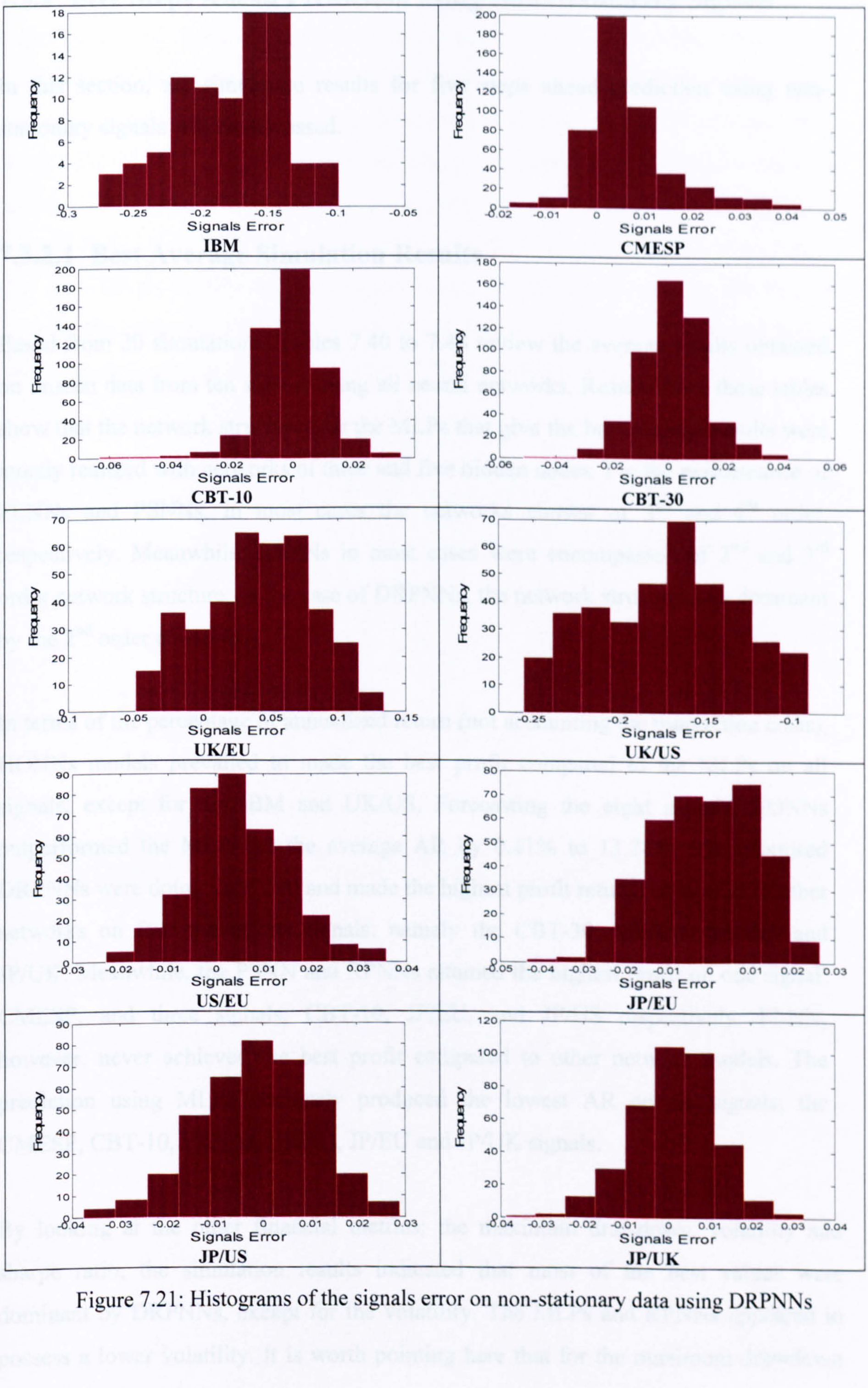


Figure 7.21: Histograms of the signals error on non-stationary data using DRPNNs

7.3.2 Five Steps Ahead Prediction using Non-Stationary Signals

In this section, the simulation results for five steps ahead prediction using non-stationary signals will be discussed.

7.3.2.1 Best Average Simulation Results

Based from 20 simulations, Tables 7.40 to 7.44 review the average results obtained on unseen data from ten signals using all neural networks. Results from these tables show that the network structures for the MLPs that give the best average results were mostly realized with networks of three and five hidden nodes. For the performance of FLNNs and PSNNs, in most cases the networks consist of 2^{nd} and 4^{th} order, respectively. Meanwhile, RPNNs in most cases were encompassed of 2^{nd} and 3^{rd} order network structure. In the case of DRPNNs, the network structure was dominant by the 2^{nd} order networks.

In terms of the percentage of annualized return (not accounting the transaction costs), HONNs models prevailed to made the best profit compared to the MLPs on all signals, except for the IBM and UK/US. Forecasting the eight signals, HONNs outperformed the MLPs on the average AR by 0.41% to 13.28%. The proposed DRPNNs were doing very well and made the highest profit returns compared to other networks on four out of ten signals; namely the CBT-30, UK/EU, US/EU, and JP/UK. Meanwhile, the PSNN and RPNNs attained the highest profit on one signal; CMESP, and three signals; CBT-10, JP/EU, and JP/US respectively. FLNN, however, never achieved the best profit compared to other network models. The prediction using MLPs obviously produced the lowest AR on six signals; the CMESP, CBT-10, UK/EU, US/EU, JP/EU and JP/UK signals.

By looking at the other financial metrics; the maximum drawdown, volatility and sharpe ratio, the simulation results indicated that most of the best values were dominant by DRPNNs, except for the volatility. The MLPs and RPNNs appeared to possess a lower volatility. It is worth pointing here that for the maximum drawdown

and sharpe ratio, a bigger value is preferable. Meanwhile for volatility, a lower value is desirable. For all financial metric evaluations, the MLPs revealed to perform worst in most cases of the signal predictions. DRPNNs also attained the highest values in four out of ten signals when assessed with the correct directional change (CDC), that is the IBM, CBT-10, CBT-30, and JP/UK signals. In the same measure, MLPs made the lowest CDC when predicting four of the signals, namely the CMESP, CBT-10, CBT-30, and JP/EU signals. When measuring the NMSE, MSE, and SNR, it can be noticed that all HONNs models outperform the MLPs in all signals. Among the HONNs models, the proposed DRPNNs achieved the lowest NMSE and MSE, and highest SNR in five signals; namely the CMESP, CBT-30, UK/US, JP/US and JP/UK.

Table 7.40: Best average result from the MLPs

Time Series	No.Hidden Nodes	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	4	4.6310	-17.0223	31.7003	0.1463	1.4847	0.0027	51.55	18.38
CMESP	4	-4.3568	-23.9475	9.2943	-0.4683	3.4799	0.0084	52.57	19.74
CBT-10	3	-11.3984	-22.0317	5.9958	-1.9031	0.1135	0.0007	50.34	29.22
CBT-30	8	0.9245	-8.2450	7.8388	0.1179	0.8775	0.0056	53.26	20.61
UK/EU	5	-2.4510	-8.5185	5.6097	-0.4372	4.3038	0.0087	53.39	18.51
UK/US	5	12.2974	-5.5978	8.1566	1.5084	2.8735	0.0052	53.62	14.50
US/EU	5	-0.4046	-12.9163	8.8697	-0.0456	5.0802	0.0140	51.26	16.88
JP/EU	3	-1.4164	-11.3317	7.9151	-0.1792	18.6406	0.0132	50.93	19.55
JP/US	8	8.2792	-9.1225	8.5967	0.9633	0.4653	0.0017	52.09	21.69
JP/UK	3	4.6096	-9.1185	7.3800	0.6250	1.5511	0.0016	54.39	27.66

Table 7.41: Best average result from the FLNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	4	2.2919	-19.4881	31.7203	0.0724	2.0686	0.0038	49.71	16.73
CMESP	4	-3.7535	-20.5846	9.3020	0.4038	0.1160	0.0003	52.58	34.24
CBT-10	4	-3.3257	-12.3002	6.0157	0.5531	0.1105	0.0007	52.39	29.34
CBT-30	5	-0.2145	-11.9279	7.8362	0.0271	0.6259	0.0040	53.57	22.40
UK/EU	5	2.9488	-6.0166	5.6078	0.5264	0.3452	0.0007	52.72	29.02
UK/US	2	7.1974	-5.4384	8.1846	0.8795	4.4686	0.0081	52.50	12.36
US/EU	3	1.8762	-10.5181	8.8645	0.2118	0.3311	0.0009	50.83	28.92
JP/EU	2	-1.0040	-10.7201	7.9115	0.1274	0.4961	0.0004	52.31	32.63
JP/US	2	7.5879	-11.8206	8.6003	0.8823	0.2102	0.0008	51.88	25.19
JP/UK	2	5.6456	-8.6591	7.3792	0.7653	0.4797	0.0005	55.21	31.15

Table 7.42: Best average result from the PSNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	3	2.1275	-18.8953	31.7177	0.0673	1.7093	0.0031	50.46	19.33
CMESP	2	-3.2481	-21.0706	9.3019	0.3495	0.1320	0.0003	52.75	34.48
CBT-10	5	-1.8877	-11.1463	6.0152	0.3144	0.4937	0.0030	52.75	26.80
CBT-30	2	0.2121	-10.3497	7.8380	0.0271	0.4089	0.0026	53.69	23.93
UK/EU	4	1.6594	-5.9605	5.6095	0.2961	1.4054	0.0029	53.52	24.67
UK/US	3	10.6332	-5.5241	8.1657	1.3032	4.5271	0.0082	53.41	20.88
US/EU	4	2.1340	-11.0979	8.8608	0.2420	4.9430	0.0136	51.83	19.68
JP/EU	4	-0.7323	-11.1450	7.9083	0.0922	8.5022	0.0060	52.12	29.56
JP/US	4	7.3261	-10.1964	8.6003	0.8521	0.5764	0.0021	52.82	20.73
JP/UK	3	5.1359	-5.5934	7.3792	0.6963	0.5587	0.0006	54.22	30.51

Table 7.43: Best average result from the RPNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	3	3.3807	-17.9568	31.6989	0.1069	1.2244	0.0022	51.95	19.29
CMESP	2	-3.2925	-20.3323	9.3006	0.3549	0.1012	0.0002	52.67	34.43
CBT-10	3	1.8808	-8.8009	6.0132	0.3127	13.9260	0.0847	54.06	9.51
CBT-30	3	0.1378	-9.5628	7.8370	0.0176	0.1638	0.0010	53.38	28.80
UK/EU	3	2.3788	-6.4716	5.6091	0.4242	0.2546	0.0005	53.94	30.31
UK/US	2	10.7459	-5.9207	8.1669	1.3163	0.2576	0.0005	53.07	24.77
US/EU	2	1.8078	-10.2540	8.8671	0.2041	9.6383	0.0265	50.69	21.82
JP/EU	5	1.6069	-10.7718	7.9071	0.2038	0.4865	0.0003	52.84	32.76
JP/US	2	9.1770	-8.1107	8.5912	1.0686	0.8842	0.0033	52.92	19.23
JP/UK	4	6.9296	-7.7458	7.3719	0.9411	0.5145	0.0005	54.87	30.85

Table 7.44: Best average result from the DRPNNs

Time Series	Network's Order	Annualized Return	Maximum Drawdown	Volatility	Sharpe Ratio	NMSE	MSE	CDC	SNR (dB)
IBM	3	4.6259	-19.9992	31.6221	0.1465	2.3335	0.0042	52.64	18.05
CMESP	3	-3.2542	-21.5630	9.2958	0.3510	0.0907	0.0002	52.70	35.29
CBT-10	3	0.5473	-10.1067	6.0127	0.0915	1.3583	0.0083	54.10	21.93
CBT-30	2	0.9651	-10.0597	7.8379	0.1232	0.1168	0.0007	53.75	29.46
UK/EU	2	4.3449	-7.1003	5.6074	0.7752	0.3945	0.0008	53.74	28.68
UK/US	3	8.8454	-6.2374	8.1735	1.0836	0.2359	0.0004	52.97	25.19
US/EU	2	2.1552	-10.2798	8.8620	0.2450	0.6360	0.0018	51.79	25.81
JP/EU	2	1.1228	-9.6084	7.9073	0.1427	0.5149	0.0004	52.51	32.52
JP/US	2	7.3246	-9.5584	8.5999	0.8520	0.1554	0.0006	52.90	26.43
JP/UK	2	7.1044	-8.3385	7.3726	0.9643	0.4190	0.0004	55.22	31.74

For demonstration purpose, the annualized return achieved in all network models (as given in Tables 7.40 to 7.44) is depicted in Figure 7.22. Meanwhile, the average number of epochs reached for the prediction of all data signals during the training of the networks is shown in Table 7.45. Networks that converged fastest are likely to disperse in all HONNs models. Apart from the prediction of IBM, JP/EU and JP/US, HONNs revealed to use less number of epochs to converge on all data, which is 1.24

to 155 times faster than the MLPs. Each DRPNNs, RPNNs, and FLNNs have shown to require the least number of epochs to converge on three out of ten signals, whereas PSNN only converged quickest on one signal. Of all the ten signals, MLPs appeared to utilize more epochs to complete the whole training, except for the prediction of UK/US, JP/EU, and JP/US.

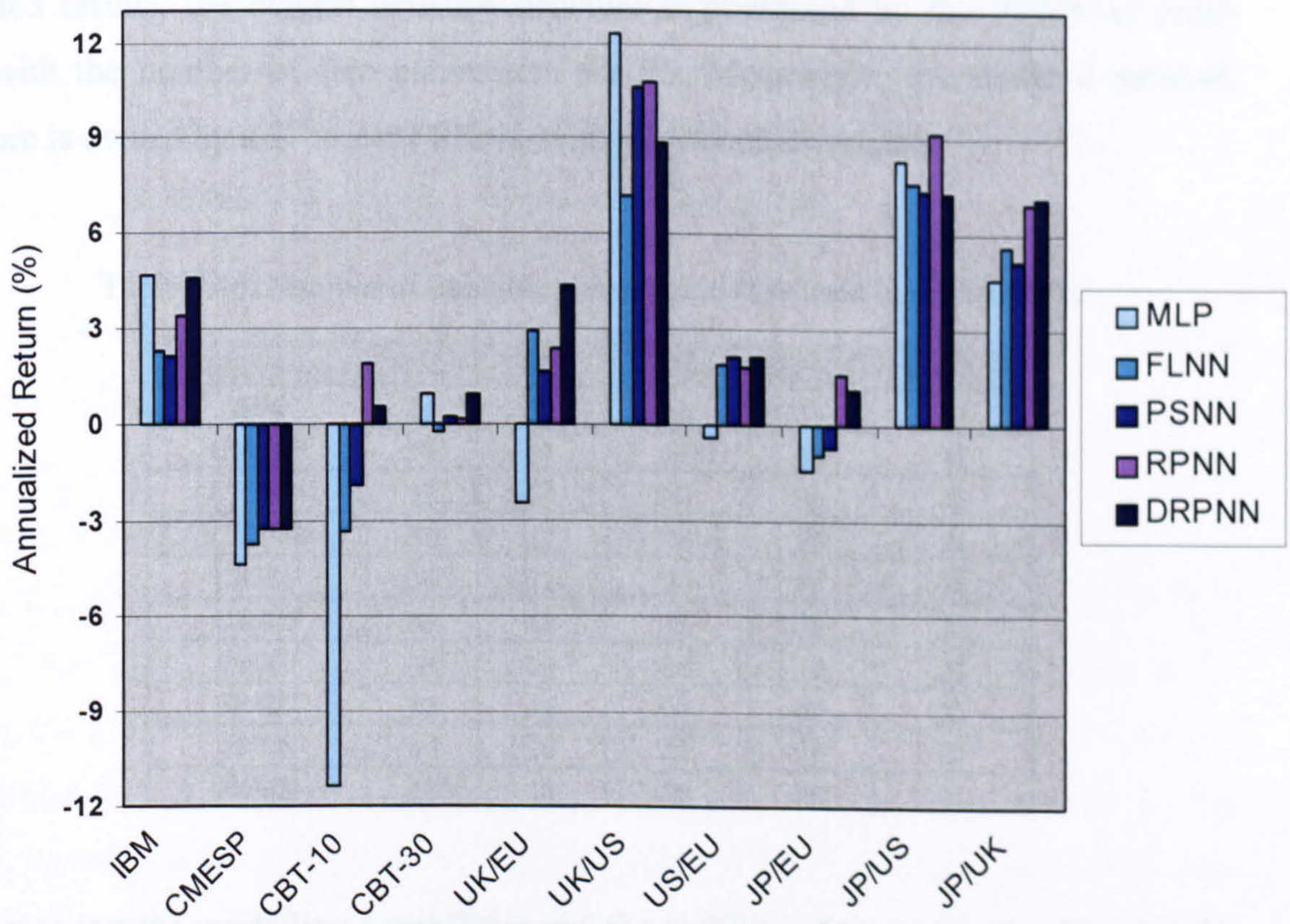


Figure 7.22: Best average annualized return from all network models.

Table 7.45: The average maximum epoch reached during training

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	3000	3000	3000	128	70
CMESP	104	19	34	51	20
CBT-10	2524	116	40	72	84
CBT-30	2520	21	479	293	35
UK/EU	1040	42	544	91	10
UK/US	945	3000	739	140	189
US/EU	1554	28	241	10	16
JP/EU	937	19	47	1149	24
JP/US	2301	2610	755	124	415
JP/UK	2490	2006	1374	1496	176

Table 7.46 demonstrates the results on the number of trainable weights and bias utilized in all network models calculated from the network structures given in Tables 7.40 to 7.44. After examining the number of free parameters, it appears that most of the smallest network structures are prevailing by the FLNNs, followed by PSNNs. The MLPs comprise of larger number of trainable weights to learn five of the financial time series; that is the CBT-30, UK/EU, UK/US, US/EU, and JP/US. From the tabulated results, the biggest network structure is possessed by the RPNN of order five, with the number of free parameters are 90. Meanwhile, the smallest network structure is owned by a 2^{nd} order PSNNs, with 12 trainable weights.

Table 7.46: Number of trainable weights and bias used in all networks

Time Series	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	29	31	18	36	42
CMESP	29	31	12	18	42
CBT-10	22	31	30	36	42
CBT-30	57	32	12	36	21
UK/EU	36	32	24	36	21
UK/US	36	16	18	36	21
US/EU	36	26	24	18	21
JP/EU	22	16	24	90	21
JP/US	57	16	24	18	21
JP/UK	22	16	18	60	21

In order to test the modelling capabilities and the stability of the networks, Figure 7.23 and Figure 7.24 illustrate the best average result of AR and NMSE, respectively, tested on unseen data, when used to predict the financial signals. The performance of the networks was evaluated with the number of higher order terms increased from 1 to 5 (for RPNNs and DRPNNS), and from 2 to 5 (for FLNNs and PSNNs), and number of hidden nodes increased from 3 to 8 (for MLP). The plots in Figure 7.23 indicate that the proposed network, DRPNNs, generally learned the data steadily. The performance of the networks in all signals showed an increment in the AR along with the network growth except for three cases, namely the prediction of JP/US, JP/UK, and CBT-30; in which the AR began to drop when the 3^{rd} order Pi-Sigma unit was added to the networks. In most cases, the plots for RPNNs and PSNNs show a rise in the AR, and in most of the cases when the networks reached order higher than two, the performance start to degrade, and usually the AR will not rise up again. This can be

seen when the PSNNs were used to predict the IBM, UK/US, UK/EU, US/EU, JP/US, JP/UK signals, and when RPNNs predicting the IBM, CMESP, UK/US, US/EU, JP/US, and CBT-30 signals. In the case of evaluating the performance of FLNNs and MLPs, most of the plots mostly demonstrate an up and down movement, namely when FLNNs predicting the CMESP, US/EU, JP/US, CBT-30, UK/EU signals, and when MLPs predicting the IBM, UK/US, UK/EU, JP/US, and JP/UK signals. This may indicate that there is no clear pattern whether the profit is going up or down when the network order or number of hidden nodes in the networks were appended. Nevertheless, MLP when used to predict the CBT-30 signal generated a continuously increasing profit with the increment number of hidden nodes.

Figure 7.24 demonstrates the average performance of the NMSE with increasing networks order or number of hidden nodes. Using all the signals, DRPNNs exhibit a stable performance with a significantly decreased NMSE along with the network growth. Meanwhile, the performance of RPNNs when forecasting the UK/US and JP/EU signals show a decreasing NMSE, and conversely, in the US/EU and JP/UK signals, the NMSE keep rising along the network growth. For the remaining of the signals, the plots for RPNNs demonstrate a zigzag movement. In most cases, the plots for PSNNs and FLNNs show a rise in the NMSE, except for the prediction of the JP/UK when using the PSNN, and for the prediction of IBM, UK/US, and JP/US when using the FLNNs. On the contrary, the plots for the MLPs revealed to show a zigzag movement on the NMSE, and in certain cases, the NMSE keep on increasing, except for the prediction of CMESP.

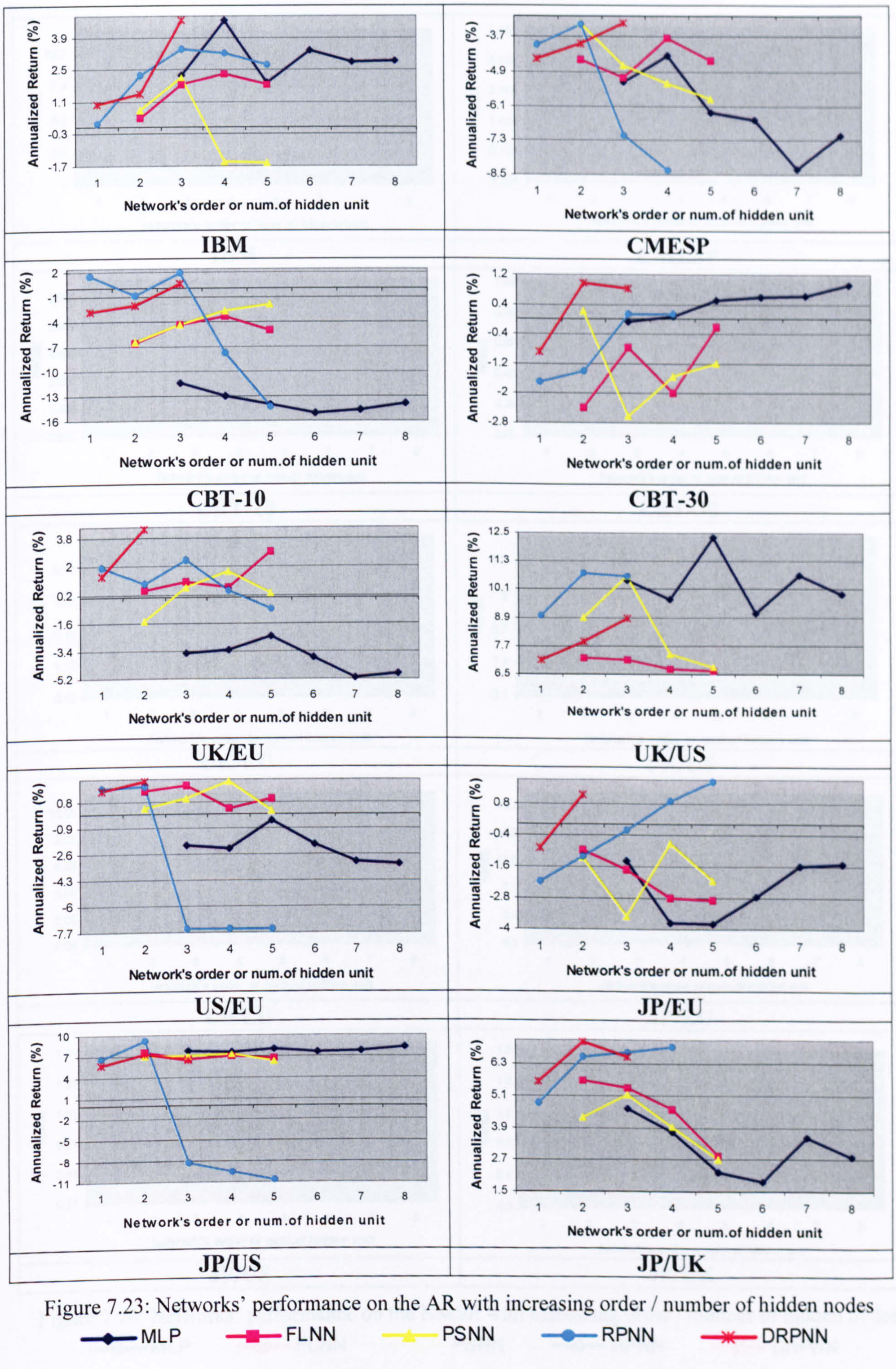


Figure 7.23: Networks' performance on the AR with increasing order / number of hidden nodes

MLP FLNN PSNN RPNN DRPNN

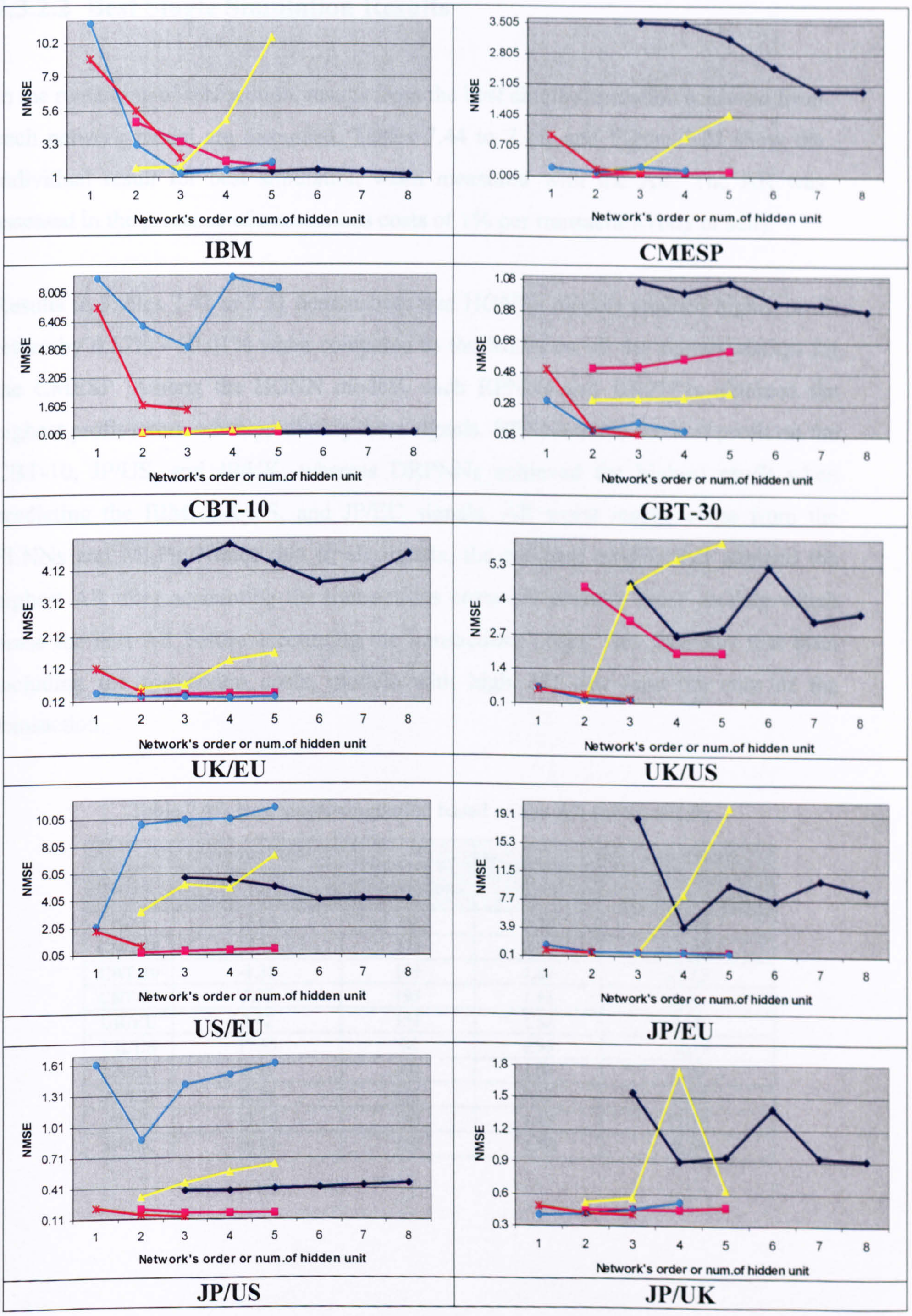


Figure 7.24: Networks' performance on the NMSE with increasing order / number of hidden nodes

MLP FLNN PSNN RPNN DRPNN

7.3.2.2 Best Single Simulation Results

In the remaining of this section, results from the best single simulation achieved from each network model are discussed. Tables 7.44 to 7.48, and Figure 7.25 show the individual result for best simulation when measured with the AR. The AR was assessed in the presence of transactions costs of 1% per transaction (buy or sell).

Results in Tables 7.47 to 7.51 demonstrate that HONNs models attained higher profit return by 0.17% - 19.01% when compared to the MLPs on all the signals, except for the CMESP. Among the HONN models, each RPNNs and DRPNNs obtained the highest profit return when predicting three signals. RPNNs made the best profit on the CBT-10, JP/US, and JP/UK, whereas DRPNNs achieved the highest profit when predicting the IBM, UK/US, and JP/EU signals. All worst results come from the FLNNs and MLPs. Notice that in all signals, the network models that attained the highest AR after accounting the transactions costs are actually those models which made the best AR before accounting the transactions costs. This indicates that even including the transaction costs, models with high AR can save the cost of the transaction.

Table 7.47: Best single simulation based on the AR for the MLPs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	16.03	29	0.29	15.74
CMESP	18.18	270	2.70	15.48
CBT-10	-1.36	189	1.89	-3.25
CBT-30	5.08	193	1.93	3.15
UK/EU	2.42	153	1.53	0.89
UK/US	17.55	96	0.96	16.59
US/EU	5.83	104	1.04	4.79
JP/EU	6.76	113	1.13	5.63
JP/US	13.19	143	1.43	11.76
JP/UK	10.94	158	1.58	9.36

Table 7.48: Best single simulation based on the AR for the FLNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	10.29	35	0.35	9.93
CMESP	7.01	143	1.43	5.58
CBT-10	4.26	121	1.21	3.05
CBT-30	12.35	284	2.84	9.51
UK/EU	11.78	116	1.16	10.62
UK/US	10.11	161	1.61	8.50
11US/EU	11.89	91	0.91	10.98
JP/EU	8.42	131	1.31	7.11
JP/US	10.43	147	1.47	8.96
JP/UK	9.05	165	1.65	7.40

Table 7.49: Best single simulation based on the AR for the PSNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	12.93	25	0.25	12.68
CMESP	9.91	168	1.68	8.22
CBT-10	4.62	179	1.79	2.83
CBT-30	4.94	134	1.34	3.60
UK/EU	7.84	127	1.27	6.58
UK/US	16.24	154	1.54	14.70
US/EU	15.91	126	1.26	14.65
JP/EU	15.38	144	1.44	13.94
JP/US	10.94	156	1.56	9.38
JP/UK	11.56	201	2.01	9.55

Table 7.50: Best single simulation based on the AR for the RPNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	20.49	46	0.46	20.03
CMESP	7.90	139	1.39	6.51
CBT-10	18.34	258	2.58	15.76
CBT-30	8.02	214	2.14	5.88
UK/EU	8.23	151	1.51	6.72
UK/US	16.43	136	1.36	15.07
US/EU	13.40	208	2.08	11.32
JP/EU	15.56	138	1.38	14.18
JP/US	13.38	131	1.31	12.07
JP/UK	13.23	150	1.50	11.73

Table 7.51: Best single simulation based on the AR for the DRPNNs

Time Series	Annualized Return (excluding TC)	Number of Transactions	Transaction Cost	Annualized Return (including TC)
IBM	22.15	59	0.59	21.56
CMESP	12.49	260	2.60	9.89
CBT-10	14.70	257	2.57	12.13
CBT-30	10.04	268	2.68	7.36
UK/EU	9.92	187	1.87	8.05
UK/US	17.66	91	0.91	16.75
US/EU	14.12	187	1.87	12.25
JP/EU	16.93	162	1.62	15.31
JP/US	11.75	95	0.95	10.80
JP/UK	12.94	148	1.48	11.46

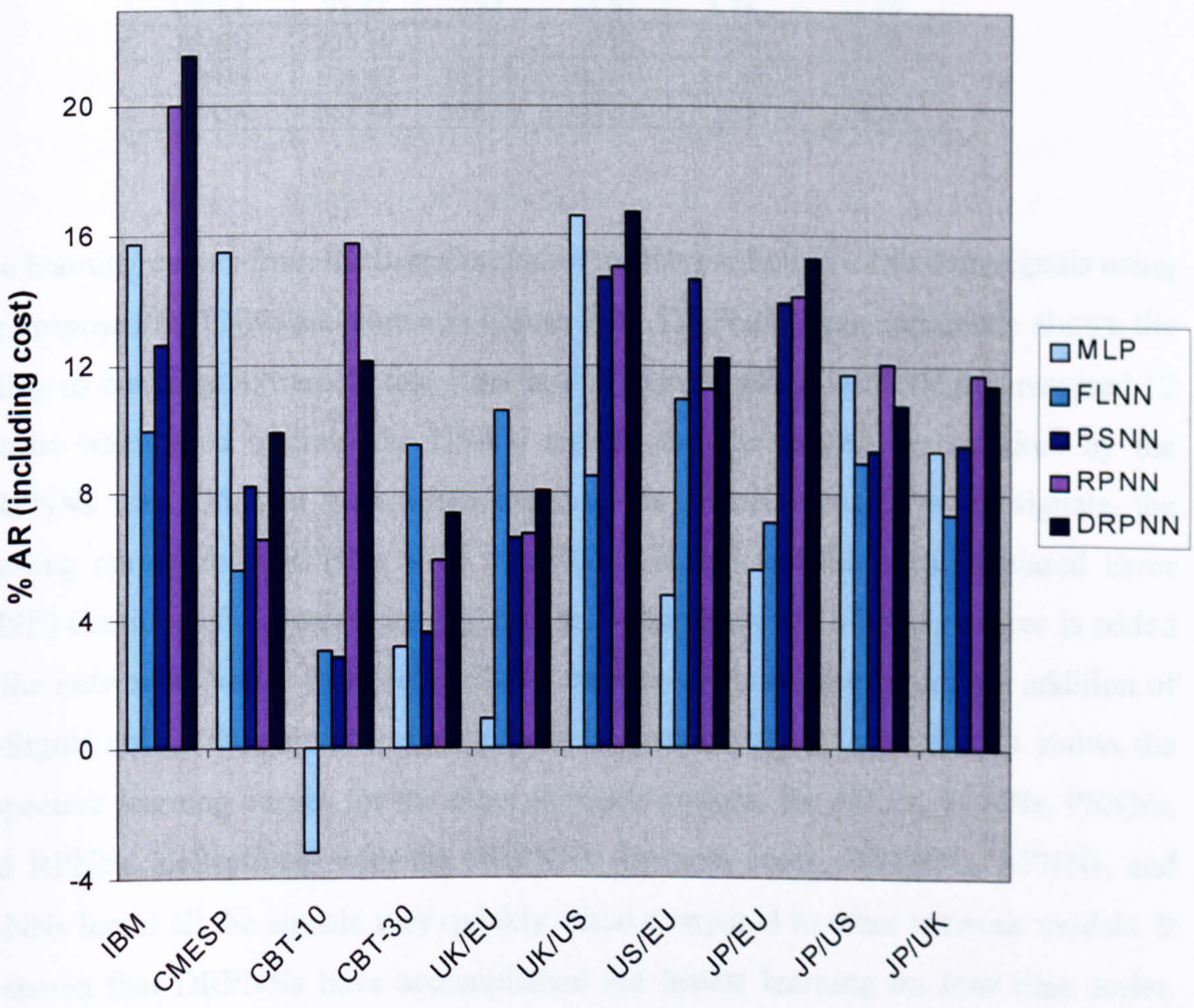


Figure 7.25: Best AR (including the transaction cost)

Results for CPU time for the training of non-stationary signals, forecasting five steps ahead is presented in Table 7.52. FLNNs demonstrate the fastest CPU time in most of the learning, namely when used to train the CMESP, CBT-10, CBT-30, UK/EU, US/EU, and JP/EU signals, which is 1.01 to 77.28 time faster than other neural

networks. This is followed by DRPNNs when learning the UK/US, JP/US and JP/UK signals with 1.28 to 15.90 times faster then other network models. Results for the MLPs reveal that the network took the longest time to learn all the signals, except for the UK/US.

Table 7.52: CPU time usage for training each neural network

Predictor	MLP	FLNN	PSNN	RPNN	DRPNN
IBM	156.64	95.44	133.98	5.37	28.55
CMESP	30.27	3.58	5.77	10.45	13.41
CBT-10	591.23	24.53	24.75	25.23	61.38
CBT-30	253.89	4.13	9.55	37.61	28.58
UK/EU	180.52	2.86	8.16	15.17	6.38
UK/US	245.11	372.36	56.23	53.28	27.86
US/EU	93.42	2.94	15.53	3.78	11.16
JP/EU	230.30	2.98	18.81	21.61	15.94
JP/US	674.69	389.36	213.92	54.50	42.42
JP/UK	667.44	389.33	337.72	356.57	125.86

The learning curves from the best simulation for the prediction of all data signals using the proposed DRPNNs are shown in Figure 7.26. DRPNNs have apparently shown the ability to converge extremely fast. The fastest learning using DRPNN just required 12 epochs when used to train the US/EU signal, and the largest epoch taken by the DRPNNs was 436, that were when learning the JP/UK signal. For all signals, the learning curves for DRPNNs were remarkably stable and the Mean Squared Error (MSE) continuously decreased every time a Pi-Sigma unit of a higher degree is added to the networks. Notice that each spike in the curve was resulted from the addition of Pi-Sigma unit in the networks. For purpose of demonstration, Appendix 11 shows the respective learning curves for the other network models; the MLPs, FLNNs, PSNNs, and RPNNs, collectively with the DRPNNs. In most cases, DRPNNs, RPNNs, and FLNNs learnt all the signals very quickly when compared to other network models. It is shown that DRPNNs have accomplished the fastest learning on four time series, which is when used to predict the IBM, JP/UK, JP/US, and UK/EU signals. The RPNNs have made the fastest learning on three signals; the CBT-10, US/EU, and UK/US. Meanwhile the FLNNs converged fastest on the prediction of CMESP, CBT-30, and JP/EU signals. Of all networks, MLPs appeared to utilize more epoch in most of the signals, apart from the CMESP, JP/UK, and UK/US.

Following the learning curve plots, Figure 7.27 shows the best prediction on all signals using the proposed DRPNNs. The plots were taken from the first 100 data points from the unseen part of the data, except for the IBM as the signal has less than 100 point of out-of-sample data. For demonstration purpose, the relevant plots for the other network models are shown in Appendix 12. By looking at the plots in Figure 7.27 and Appendix 12, it can be spotted that at some points, the original and predicted signals are a bit distant from each other. This can suggest that the non-stationary time series signals are harder to predict, as compared to the previously presented stationary signals. Meanwhile, Figure 7.28 presents the histograms of the prediction errors using DRPNNs, which demonstrates that most of the signals error are not approaching zero, even though in some cases, the histograms show a normal distribution. Further, Appendix 13 depicts the histograms of the prediction errors for the other four networks; the MLPS, FLNNs, PSNNs, and RPNNs, which also indicates that most of the prediction errors for these networks are far off zero, while relatively few of them tend to one extreme or the other. When compared to the prediction error from stationary signals, histogram plots in Figure 7.28 and Appendix 13 suggest that the non-stationary time series signals are more difficult to predict.

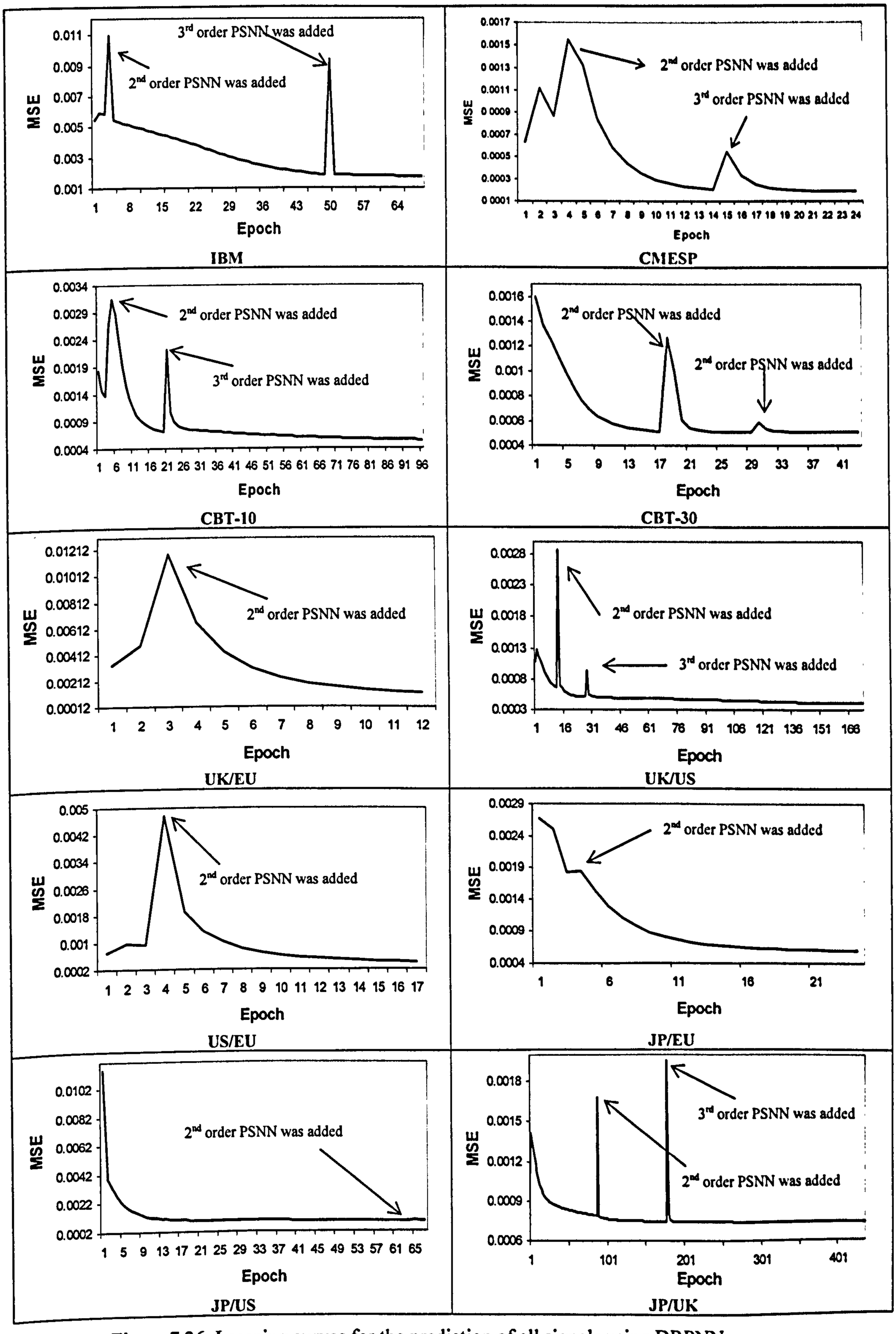


Figure 7.26: Learning curves for the prediction of all signals using DRPNNs

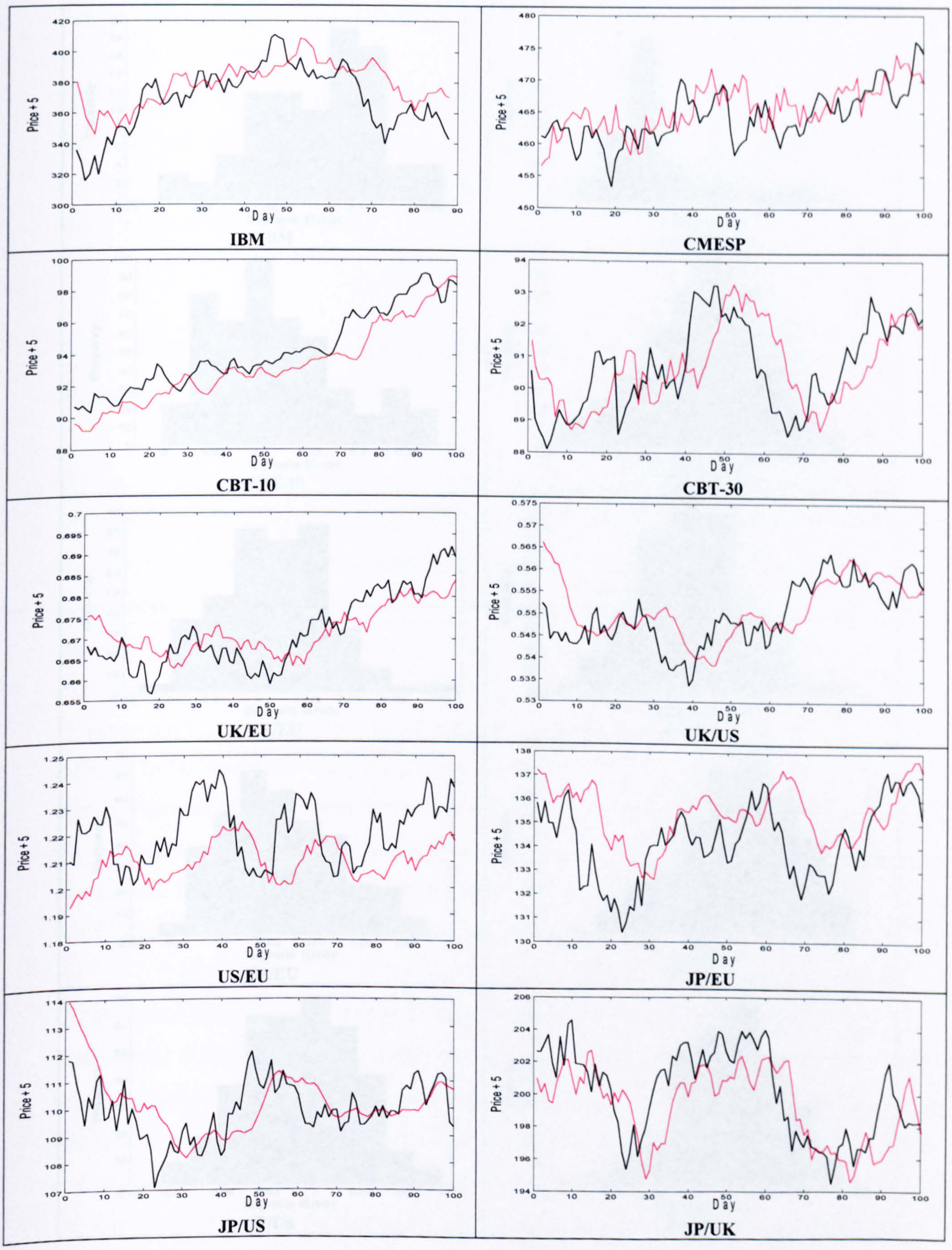


Figure 7.27: Best forecasts made by DRPNN on all data signals
Original signal, Predicted signal

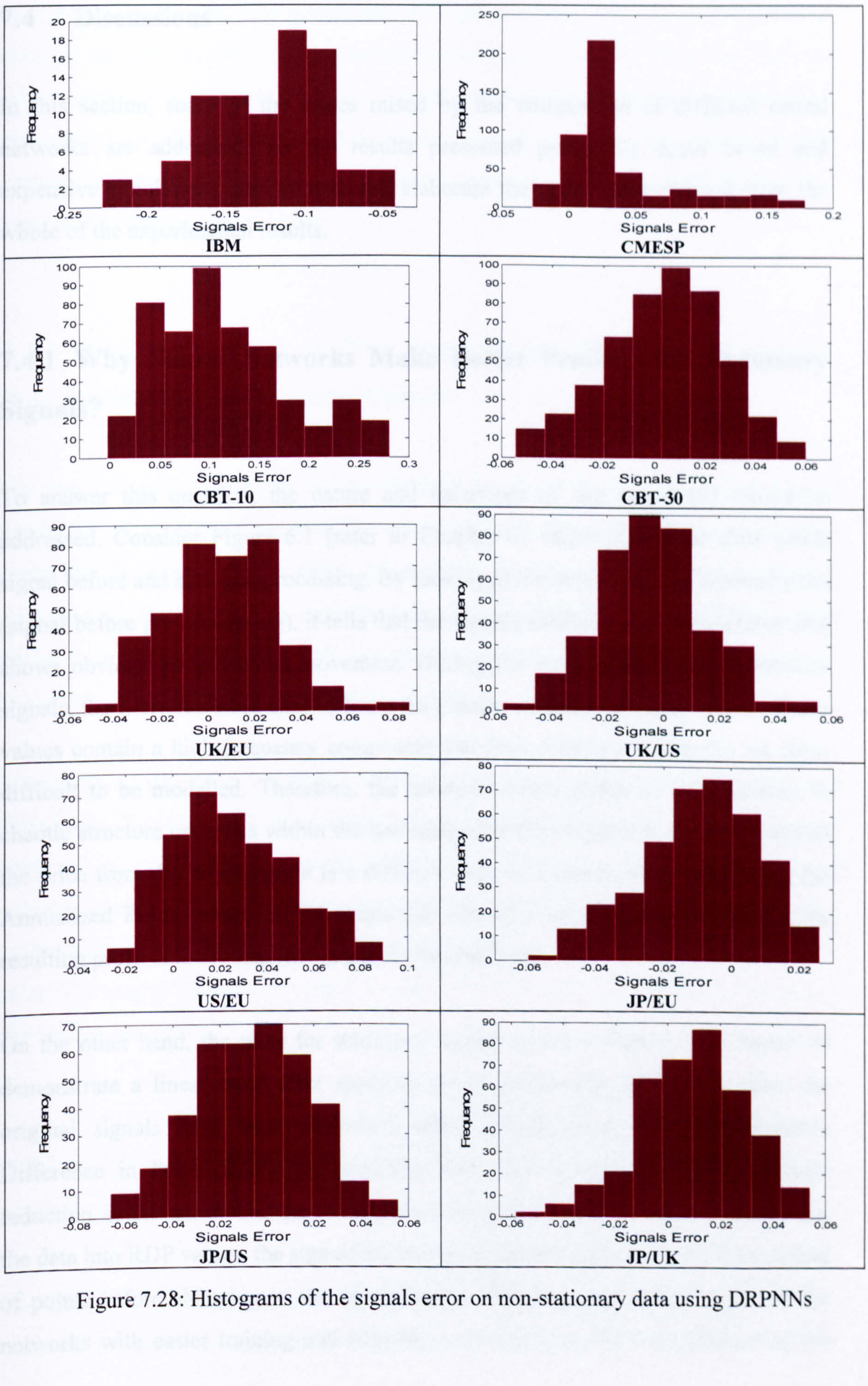


Figure 7.28: Histograms of the signals error on non-stationary data using DRPNNs

7.4 Discussions

In this section, some of the issues raised by the comparison of different neural networks are addressed. As the results presented previously cover broad and expensive simulations, this section will elaborate the observation derived from the whole of the experimental results.

7.4.1 Why Neural Networks Make Better Profits with Stationary Signals?

To answer this question, the nature and behaviour of the data itself should be addressed. Consider Figure 6.1 (refer to Chapter 6) which shows the time series signal before and after pre-processing. By looking at the non-stationary forecast plots (signal before pre-processing), it tells that the signals exhibit a very strong trend and shows obvious up and down movement. During the training of the non-stationary signals, the networks were used to learn the precise values of the daily prices. These values contain a high-frequency component and their relative magnitudes are more difficult to be modelled. Therefore, the networks often unable to respond well to chaotic structure underlies within the non-stationary data. Hence, to correctly predict the price from day to day point is a difficult task. As a result, when calculating the Annualized Return based on the magnitude size of correct directional change, the resulting profit is likely unpromising and unsatisfactory.

On the other hand, the plots for stationary signals (refer to Figure 6.1, Chapter 6) demonstrate a linear trend after applying the pre-processing technique. Since the original signals have been smoothed before transforming them into Relative Difference in Price (RDP), the resulting stationary signals demonstrate a huge reduction in the trends and day to day variations. Furthermore, when transforming the data into RDP values, the sign of the resulting signal remain the same for a period of points before changing to the opposite sign. The stationary signals provide the networks with easier training and help the networks to capture the essence of the

background movement. The assumption that pre-processing the signal before using them in the networks will lead to better forecasting seems to hold for all the time series used in this research work. As a result, the potential profit that the networks can achieve by correctly predicting the changes increases. Thus, the result presented previously support the theoretical concept of non-stationary signal is much harder to predict than the stationary signals.

7.4.2 Why Forecasting of Five Steps Ahead Make Better Profits than Forecasting of One Step Ahead?

For non-stationary signals, it is very difficult to obtain direct relation between the prediction of one step ahead and five steps ahead, as the results do not show which prediction is better than the other. As a matter of fact, the prediction of the non-stationary signals is about to learn the mapping of the precise values of daily prices. Probably because of this, the profit gained through forecasting either one step ahead or five steps ahead was likely did not show a big different.

Meanwhile, there is an obvious difference in terms of the profit gained when forecasting the one step ahead stationary signal (RDP+1) and five steps ahead stationary signal (RDP+5). In nature, the size of changes from point to point in the original series varies significantly. The magnitude of RDP of point x to point $x+5$, is comparatively larger than that of x to point $x+1$. Therefore, forecasting the RDP+5 provide a greater chances of making better profit if the networks correctly predicting the sign of the change.

7.4.3 Why some Neural Networks with Good AR Produce High NMSE and Low SNR?

The simulation results showed that in some cases some neural networks attained high profit return, but at the same time they produce slightly high NMSE and lower SNR.

This does not reflect the significant profitable value offered by the networks. This is because the SNR and NMSE are calculated based on the squared error; therefore, if a model has a low NMSE, the SNR will be high. Conversely, if the NMSE is high, the SNR will drop. On the other hand, the AR is a scaled calculation of the observed change in the time series value, when the sign of the change is correctly predicted. Hence, it is worth noting that seeking optimal forecasting in terms of NMSE is not the aim of this research work, as explained previously in chapter 6.

7.4.4 How do DRPNNs and RPNNs Compare?

DRPNNs in most cases are more capable of giving higher profit return when compared to the RPNNs for both stationary and non-stationary data. When considering the other performance measures, DRPNNs on the whole consistently gave better results than the RPNNs, with the exception for Correct Directional Change (CDC) for the prediction of one step ahead using non-stationary signals, in which RPNNs offered better results than DRPNNs.

The networks are also compared in terms of the networks' complexity. 'Complexity' will be defined as the number of adaptive weights used by the neural network. While there are other measures of complexity, such as the training algorithm, weights are a good complexity measure. As in the case when all other properties of two neural networks are equal, a network which has less weights will usually execute faster and require less time for training. Such properties are especially desirable in real-time systems.

In most cases, DRPNNs were found to have a reduced complexity compared to the RPNNs when predicting the one step ahead of non-stationary signals and five steps ahead of stationary signals. RPNNs on the other hand have less number of adaptive weights when dealing with one step ahead prediction of stationary signals. For the prediction of five steps ahead of non-stationary signals, both networks fairly prevailing to have smaller network size. For both predictions of one step and five steps ahead using both stationary and non-stationary signals, the smaller number of

weights utilized by the DRPNNs in most of the cases led to significantly faster training-simulation of these networks as well as reduced memory requirements when compared to the RPNNs.

7.4.5 How do HONNs and MLPs Compare?

When comparing the results given by all HONNs models (DRPNNs, RPNNs, PSNNs, and FLNNs) and the MLPs, HONNs models gave broadly the best average results on the profit return. Forecasting the one step ahead using stationary and non-stationary signals, HONNs significantly outperformed the MLPs in all time series used. When using both versions of the signals to forecast the five steps ahead, HONNs demonstrate better profit than the MLPs in all time series, apart from the IBM and UK/US signals. Further, evaluating the networks with other performance measures shows that HONNs on the whole offered more promising results than the MLPs for both versions of signals and forecast horizons.

In both predictions of one step and five steps ahead using both stationary and non-stationary signals, all the smallest network size was dominant and shared by HONN models, with the exception for five steps ahead prediction of CBT-10 non-stationary signal, in which the MLP comprised of the smallest network size. With the reduction in networks complexity in HONNs, all the fastest convergence presented in the experimental results was dominant and shared by HONNs models. Nevertheless, MLPs in all cases apparently have never converged fastest during their training simulations when compared to the HONNs.

The significant performance of HONNs in comparison to the MLPs is not surprising. HONNs are computationally efficient nonlinear networks and they are capable of complex nonlinear mapping between the input and output. The use of higher order terms allows them to increase their information capacity and expand their input space into higher dimensional space. Besides, HONNs structures are single layered of learnable weights and thus allow the networks to advance faster toward convergence. Reductions in the training time and number of free parameters help to avoid the

problems of over-fitting, resulting in the enhancement of networks performance and make them superior to the MLPs.

7.4.6 Why DRPNNs Made the Best Profit than Other Network Models?

Simulation results demonstrate that all the neural networks models used in this research work were potentially profitable, however, in most cases the proposed DRPNN is by far the most beneficial as money-making predictor. The use of DRPNN in financial time series prediction shows that the proposed network provides promising tool to time series forecasting. The general property making the DRPNN interesting and potentially useful in financial prediction are as follows. The networks manifests highly nonlinear dynamical behaviour induced by the recurrent feedback, therefore leads to a better input-output mapping and a better forecast. With the recurrent connection, the network outputs depend not only on the initial values of external inputs, but also on the entire history of the system inputs. Therefore, the DRPNN is provided with memory which gives the network the ability of retaining information to be used later. The superior performance of DRPNN is also attributed to the natural mechanism for incremental network growth, therefore giving the network a very well regulated structure and smaller network size which led to and network robustness. The presence of higher order terms in the network equipped the DRPNN with the ability to forecast the upcoming trends in financial time series signals. The DRPNN is guaranteed to exhibit a unique equilibrium state, as the stability convergence of the network was applied during their training to ensure that the network always posses a stable condition. The network can robustly process the underlying dynamics of a non-stationary environment with a vast speed in convergence time. A noteworthy advantage of DRPNN is the fact that there is no requirement to select the order of the networks as in PSNN and FLNN, or the number of hidden units as in MLP.

DRPNN has shown its advantages in forecasting both stationary and non-stationary signals. Simulation results have provided additional evidence supporting the

application of DRPNN to financial time series prediction, thus suggesting that the model is good at making profit.

7.5 Chapter Summary

This chapter presents extensive simulation results of five neural networks architectures; namely the MLPs, FLNNs, PSNNs, RPNNs, and DRPNNs. Two sets of simulations are shown in this thesis, stationary and non-stationary prediction of ten financial time series to forecast the one step and the five steps ahead predictions. Networks performance was evaluated using five financial metrics, and four statistical and signal processing metrics. In the next chapter, the main conclusions of this research work are presented.

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

8.1 Introduction

This chapter summaries the final conclusions that can be derived from this research study. Matters such as different network types, financial time series used, pre-processing method, performance measure, and finding results are drawn. The novelty and contribution of this research work and possible research directions for used in the future are discussed.

8.2 Main Conclusions Derived from this Research Study

This thesis investigates extensively various types of Higher Order Neural Networks (HONNs) as non-linear prediction models and the idea of using higher-order terms within the neural network structure. Three HONNs models were explored in this research work, namely the Functional Link Neural Network (FLNN), the Pi-Sigma Neural Network (PSNN), and the Ridge polynomial Neural Network (RPNN). The learning algorithms and their use in real world applications were presented. Their strengths and capabilities in input-output mappings, on various kinds of problems ranging from signal prediction, pattern recognition, time series forecasting, data classification, and more were discussed. The networks are computationally efficient and are capable of complex nonlinear mapping between their input and output pattern space. The use of higher order terms allows the networks to expand their input space into higher dimensional space where linearly separable is possible.

In order to represent a dynamic system, the functionality and architecture of the ordinary feedforward RPNN were extended. Accordingly, a novel recurrent neural network architecture called Dynamic Ridge Polynomial Neural Network (DRPNN) was proposed. A learning algorithm for DRPNN was also derived to tune the free

parameters in the network. Subsequently, the stability and the convergence of the proposed network were implemented to ensure having steady and fixed output.

This thesis elaborated the fundamentals of financial time series prediction, addressing the difficulties and comparing neural networks and traditional forecasting approaches, particularly to the prediction of financial market. A review on literature detailing the practical applications of neural networks in financial time series prediction was shown. The design of neural networks to successfully predict financial time series was presented. More attention on the design stage was given to the pre-processing method to reduce the trend and embedded noise, as financial time series exhibit dynamic behaviour over time. A step by step methodology of constructing the network forecasting models, as well as the generation of input-output pattern, the specification of parameters, and performance measures used was discussed.

The networks have been trained and tested on ten financial time series signals. The results from extensive simulations were accordingly presented, collectively from four HONNs models; the FLNN, PSNN, RPNN, and the proposed DRPNN. Stationary and non-stationary versions of ten financial time series were used to forecast the one step ahead and five steps ahead predictions. The simulation results for the prediction of all data signals using four various feedforward networks (MLP, FLNN, PSNN, and RPNN) and recurrent network (DRPNN) were compared. Analysis on profit return, network convergence, training epochs, performance of networks with increasing order or number of hidden nodes, transaction cost, CPU time, and learning curves were given. Networks' performance was evaluated using five financial metrics, and four statistical and signal processing metrics. It is worth to point out that in financial forecasting application, an important question for any forecasting model is how the model stands up in an actual trading scenario. In this research work, the questions have been tackled by using and testing the networks with the actual trading signals. Furthermore, the performance of the networks models also evaluated with a real trading evaluation, such as the Annualized Return, Maximum Drawdown, Annualized Volatility and Sharpe Ratio.

Further, experimental results showed that HONNs in most cases always perform better in terms of producing higher profit return against the MLP. In addition to generating profitable return value, which is a desirable property in nonlinear financial time series prediction, HONNs also used smaller number of epochs during the training in comparison to the MLPs. Among all HONN models, the proposed DRPNN showed an improvement in approximating the financial time series and the networks are capable in outperforming the MLP and other HONN models. DRPNN, on the whole offers significant advantages over other network models; including such increment in profit return, reduction in network complexity, faster learning, and smaller prediction error. In summary, HONNs and especially DRPNN are one of the promises for the future in financial time series prediction. They offer an ability to perform tasks outside the scope of traditional techniques. They can recognize patterns within large datasets and then generalize those patterns into recommended courses of action.

8.3 Novelty and Research's Contribution

Despite of being successfully applied in signal prediction, pattern recognition, data classification, function approximation, and data classification, there was very little research being carried out in financial time series prediction using HONNs. The construction of the proposed DRPNN is the main contribution and novelty in this research work. The unique characteristic of DRPNN which combine the properties of Higher Order Neural Network and Recurrent Neural Network, make it suitable and useful for financial time series forecasting. In fact the way in which the data was pre-processed and presented to the HONNs and the proposed network is a novel application indeed.

The forecasting and trend prediction results using the proposed DRPNN are promising and certainly warrant further research and analysis. Given the finding results, the current study should be a great for many practitioners and managers of multinational corporations as it suggest DRPNN can be effectively used as financial forecasting tool. Further, based on the good performance of DRPNN, corporations

can devise more effective business strategy to improve their financial positions and efficacious precautionary measures to reduce potential currency risk. The use of DRPNN in financial time series prediction demonstrated that the proposed network is potentially useful for technical trading to forecast daily financial time series data.

DRPNN offers some significant advantages. The merit of DRPNN, as compared to the feedforward RPNN is its increased inherited nonlinearity which resulted from the use of recurrent neural networks architecture, giving it an advantage when dealing with financial time series forecasting. A considerable profitable value does exist in the DRPNN when compared to other neural networks and the network can make contributions to the maximization of returns. The network demonstrated a vast speed in convergence time and comprises of smaller number of network size, therefore showing a reduction in network's complexity.

8.4 Future Research Directions

The method which was proposed and presented in this research work can be viewed as starting points for future research direction, since the potential of DRPNN, especially with respect to financial time series prediction is by far not fully exploited yet. More research is needed with the use of DRPNN to give a more general account of their abilities beyond the financial time series domain. Based on the conclusions of this thesis and other research which is currently ongoing in this area, the following continuations of this research work are suggested.

Use of evolutionary computation - It should be emphasized that RPNN and DRPNN is not without problem. The main intricacy when using the networks is to find the suitable parameters for successively adding a higher degree of Pi-Sigma unit in the networks. Training the networks can be a quite expensive procedure, as it is difficult to know the best combination of learning parameters; the learning rate n, threshold r, dec_n, and dec_r. With respect to this deficiency which causing to a trial and error approach, it might be worthwhile to consider how Genetic Algorithm (GA) (Koza, 1992) can be used to automatically generating and finding suitable parameters

for the networks. Evolutionary computation has been successfully used to automatically develop neural network structures, weights adaptation and learning parameter setting; rather than the user doing this task experimentally. However, research relating to time series prediction with HONNs and evolutionary computation remains largely unexplored. GA has proven to be capable at finding near optimal solutions for problems which have extremely large search spaces. In this way, GA searches promising areas of the solution space by evolving a population of rules that tends to become more adept at solving the problem in successive generations. By implementing a GA approach to the learning parameter selection, it is expected to an improvement in the training process and therefore leads to better forecasting performance of the networks.

Improved trading strategies - As there is no perfect forecasting technique, trading profit is ensured by a good trading strategy which taking a full advantage of a good forecasting method. The transformations from prediction into market actions are obtained by specifying the trading strategies; a set of rules to buy and sell currency futures. The investigation of trading strategies forms an important part of research for people working in economics. Any sensible trading strategy should somehow restrict the number of trading transactions because of the incidence of transaction cost. More trading strategies are needed in addition to the one explored in this research work. There are some good trading strategy can be referred to, such as (Fieldsend and Singh, 2005; Dunis and Huang, 2002; Hamm and Brorsen, 2000; Tenti, 1996).

Multivariate time series - This research work has focuses on the univariate time series, which is data from the single time series to be forecasted. The ever more global nature of the world's financial markets necessitates the inclusion of more global knowledge into neural networks design. Multivariate series can look at the interdependence between several time series. Therefore, the use of multivariate series would be advantageous, since some dependent market depends on other global markets and thus the inclusion of these series will potentially improve neural networks forecasting performance (Zhang, 2003).

Market Timing Hypothesis - It would be worthwhile endeavour to consider a methodology which provides a measure of the economic value of a forecasting model. A market timing test can be used in order to give more candid examinations of the financial time series forecasts generated by different network architectures. Henriksson and Merton (1981) have developed a framework for analyzing the statistical significance of the correlation between forecasts and actual values of returns on stocks. The Henriksson–Merton market timing test is essentially a test of the directional forecasting accuracy of a model. Directional accuracy has been shown to be highly correlated with actual trading profits and a good indicator of the economic value of a forecasting model. This hypothesis test is a practical test to observe whether the network models have an economically significant value in predicting the financial time series signals. This would be very useful to evaluate the probability of the network's profit and to justify whether the networks are able for making money out of its predictions.

Testing and evaluation with datasets from different application areas - This research has evaluated the capabilities of HONNs on the prediction of financial time series data. In light of the DRPNNs' results giving a higher AR in most of the cases, an evaluation of the network using data with different properties, from different fields would be very useful. One useful application is preventing undesirable events by forecasting the event, identifying the circumstances preceding the event, and taking corrective action so the event can be avoided. Another application is forecasting undesirable, yet unavoidable events. The sunspots data series, which is data counting dark patches on the sun and is related to the solar storms, shows an eleven-year cycle of solar maximum activity (Plummer, 2000), and if accurately modelled, can forecast the severity of future activity. While solar activity is unavoidable, its impact can be lessened with appropriate forecasting and proactive action.

8.5 Chapter Summary

This research work underlines an important contribution of the proposed DRPNN; namely their elegant ability to approximate nonlinear financial time series. This superior property hold by the DRPNN could promise more powerful applications in many other real world problems. Hence, it is anticipated that HONNs, and particularly DRPNN, can be used as an alternative or supplemental method for predicting financial variables and thus justified the potential use of these models by practitioners. To conclude, DRPNN is a promising ***intelligent computational technology*** that potentially can be a challenging tool for future research.

REFERENCES:

[1] Abraham, A., Nath, B., and Mahanti, P. K. (2001) Hybrid Intelligent Systems for Stock Market Analysis, *Lecture Notes in Computer Science*, Vol. 2074, pp. 337-345.

[2] Allen, F. and Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics 51*, pp. 245-271

[3] Artyomov, E, and Pecht, O.Y (2005) Modified High-order neural Network for invariant pattern recognition. *Pattern Recognition Letters*, 26, pp. 843-851.

[4] Atiya, A.F. (1988). Learning on a general network. *In Proc. IEEE Conf. Neural Information Processing Systems*, 1987.

[5] Atiya, A.F. (2000). New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence. *IEEE Transaction on Neural Networks*, Vol. 11, No. 3, pp.697-709.

[6] Beale, R. and Jackson, T. (1990). *Neural Computing: An Introduction.* Bristol: Hilger.

[7] Berardi, V.L. (2003). An Empirical Investigation of Bias and Variance in Time Series Forecasting: Modeling Considertions and Error Evaluation. *IEEE Transactions on Neural Networks*, Vol.14, No.2, pp. 668-679.

[8] Box, G.E.P., Jenkins, G.M. and Reinsel, G.C. (1994) *Time Series Analysis, Forecasting and Control*, 3rd edn. Prentice-Hall.

[9] Cass, R. and Radl, B. (1996) Adaptive Process Optimization Using Functional-Link Networks and Evolutionary Algorithm. *Control Eng. Practice*, Vol. 4, No. 11, pp. 1579-1584.

[10] Cao, L. and Tay, F.E.H. (2001) Financial Forecasting Using Vector Machines. *In Neural Computing and Application.* Vol. 10, pp 184-192.

[11] Cao, L. J. and Tay, F. E. H. (2003). Support Vector Machine with Adaptive Parameters in Financial time Series Forecasting. *IEEE Transactions on Neural Networks*, Vol. 14, No. 6. pp. 1506-1518.

[12] Castiglione, F. (2000). Forecasting price increments using an artificial Neural Network. *Adv. Complex Systems*, 1, pp. 1-12.

[13] Chan, M.C., Wong, C.C., and Lam, C.C. (2000). Financial Time Series Forecasting by Neural Network Using Conjugate Gradient Learning Algorithm and Multiple Linear Regression Weight Initialization. Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong.

[14] Chang, L.C. Chang, F.J. and Chiang, Y.M. (2004) A two-step-ahead recurrent neural network for stream-flow forecasting. *Hydrol. Process. 18*, pp.81-92.

[15] Chen, A.S., and Leung, M.T. (2005) Performance Evaluation of Neural Network Architectures: The case of Predicting Foreign Exchange Correlations. *Journal of Forecasting*, J. Forecast. 24. pp. 403-420.

[16] Chen, A.S. and Leung, M.T. (2004) Regression Neural Network for error correction in foreign exchange forecasting and trading. *Computers & Operations Research 31*, pp. 1049-1068.

[17] Cheng, W., Wagner, L., and Lin, C.H. (1996). *Forecasting the 30-year U.S. Treasury Bond with a System of Neural Networks*. Finance & technology Publishing, Inc. USA.

[18] Chenoweth, T. and Obradovic, Z., (1995). An Explicit Feature Selection Strategy for Predictive Models of the S&P 500 Index. *Journal of Computational Intelligence in Finance, Finance & Technology Publishing*, vol. 3, no. 6, pp. 14-21.

[19] Chow, M.Y., and Teeter, J. (1997). Reduced-Order Functional Link Neural Network for HVAC Thermal System Identification and Modeling. Proceedings of 1997 International Conference on Neural Networks, Houston, TX.

[20] Connor, J.T., Martin, R.D, and Atlas, L.E. (1994) Recurrent Neural Networks and Robust Time Series Prediction. *IEEE Transactions on Neural Networks*, Vol. 5. No. 2.pp. 240-254.

[21] Cybenko, G. (1989) Approximation by superposition of a sigmoidal function. *Math. Contr., Signals, Sys.*, vol 2, pp. 303-314.

[22] *Datastream International Limited ALL RIGHTS RESERVED/* Datastream database, [machine-readable data] London, Thompson Financial Limited [producer and distributor], retrieved November 14, 2005.

[23] Dorffner, G. (1996) Neural Networks for Time Series Processing. *Neural Network World*, vol. 6. pp. 447-468.

[24] Duch, W. and Jankowski, N. (1999) Survey of Neural Transfer Function. *Neural Computing Surveys 2*, pp. 163-212.

[25] Dunis, C.L., Harris, A., Leong, S., and Nacaskul, P. (1999). Optimising intraday trading models with genetic algorithms. *Neural Network World*, 9, No.3, pp.193-224

[26] Dunis, C.L. and Huang, X. (2002). Forecasting and trading currency volatility: An application of Recurrent Neural Regression and model combination. *Journal of Forecasting*, J. Forecast. 21, 317-354.

[27] Dunis, C, L. and Williams, M. (2002) Modeling and trading the UER/USD exchange rate : Do Neural Network models perform better?. *In derivatives Use, Trading and Regulation*, Vol. No. 8, 3, pages 211-239.

[28] Durbin, R. and Rumelhart, D. E. (1989) Product Units: A Computationally Powerful and Biologically Plausible Extension to Back-propagation Networks. *Neural Computation*, vol. 1, pp. 133-142.

[29] Durbin, R. and Rumelhart, D. E. (1990) Product units with trainable exponents and multilayer networks. In F. Fogelman Soulie, and J. Herault, (Ed.) *Neurocomputing: Algorithms, Architecture and Applications*, NATO ASI Series, vol. F68, (Springer-Verlag), pp. 15-26.

[30] Elman, J. L. (1990) Finding structure in time. *Cognitive Sci. 14*, pp. 179-211.

[31] Estudillo, A.M., Estudillo, F.M., Martinez, C.H., and Pedrajas, N.G. (2006) Evolutionary product unit based neural networks for regression. *Neural Networks 19*, pp. 477-486.

[32] Fernandez-Rodriguez, F., Gonzalez-Martel, C., and Sosvilla-Rivero, S. (2000). On the profitability of technical trading rules based on artificial neural networks: Evidence from the Madrid stock market. *Economics Letters 69*, pp. 89-94.

[33] Fieldsend, J.E. and Singh, S, (2005). Pareto Evolutionary Neural Networks. *IEEE Transactions on Neural Networks*, Vol. 16, No. 2, pp. 338-354.

[34] Franses, P. H. (1998). Forecasting Exchange Rates Using Neural Networks for Technical Trading Rules. *Studies in Nonlinear Dynamics and Econometrics*, 2(4), pp. 109-114.

[35] Fraser, N., (1998). *Introduction to Neural Networks*. Available from: http://vv.carleton.ca/~neil/neural/neuron.html.

[36] Garcia, R. and Gencay, R. (2000). Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometric*, 94, pp. 93-115.

[37] Ghosh, J. and Shin, Y. (1992). Efficient Higher-order Neural Networks for Function Approximation and Classification. *Int. J. Neural Systems*, vol. 3, no. 4, pp. 323-350.

[38] Gilde, C. (1996). *Time series analysis and prediction using Recurrent Gated Experts*. Master Thesis, Dept. of Computer Science, University of Skovde.

[39] Giles, C. L., Griffin, R. D., and Maxwell, T., (1998) Encoding Geometric Invariances in HONN. *American Institute of Physics*, pp. 310-309.

[40] Giles, C. L., Lawrence, S, and Tsoi, A. C. (2001). Noisy Time Series Prediction using Recurrent Neural Networks and Grammatical Inference. Machine Learning 44(1/2), pp. 161-183.

[41] Giles, C. L. and Maxwell, T. (1987) Learning, invariance and generalization in high-order neural networks. *In Applied Optics*, vol. 26, no. 23, Optical Society of America, Washington D. C., pp.4972-4978.

[42] Gradojevic, N. and Yang, J. (2000). The application of artificial neural networks to exchange rate forecasting: The role of market microstructure variables. Bank of Canada Working Paper 2000-23. Financial Markets Department, Bank of Canada, Ontario.

[43] Guler, M. and Sahin E. (1994). A New Higher-Order Binary-Input Neural Unit: Learning and Generalizing Effectively via Using Minimal Number of Monomials. Third Turkish Symposium on Artificial Intelligence and Neural Networks Proceedings, Middle East Technical University, Ankara. pp. 51-60.

[44] Hamm, L. and Brorsen, B.W. (2000). Trading futures markets based on signals from a neural network. *Applied Economics Letters*, 7, pp. 137-140.

[45] Hartmann, D. and Pierdzioch, C. (2006). International Equity Flows and the Predictability of U.S. Stock Returns. MPRA Paper No. 562. University Library of Munich, Germany. Available from: http://mpra.ub.uni-muenchen.de/562/01/MPRA_paper_562.pdf

[46] Haykin, S. (1999) *Neural Networks. A comprehensive foundation*. Second Edition, Prentice-Hall, Inc., New Jersey.

[47] Hellström, T., and Holmström, K. (1998) Predicting the Stock Market. Technical Report IMA-TOM-1997-07, Center of Mathematical Modeling, Department of Mathematics and Physics, Mälardalen University, Västeras, Sweden. Available from: http://www.cs.umu.se/~thomash/reports/pred.pdf

[48] Henriksson, R.D. and Merton R.C. (1981). On the Market Timing and Investment Performance of Managed Portfolios II - Statistical Procedures for Evaluating Forecasting Skills. *Journal of Business*. 54, pp. 513-533.

[49] Ho, S.L., Xie, M. and Goh, T.N. (2002) A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction. *Computers & Industrial Engineering*, 42, pp. 371-375.

[50] Hornik, K., Stinchcombe, M. and White, H. (1989) Multilayer feedforward networks are universal approximators. *Neural networks*, vol. 2, pp. 359-366.

[51] Husken, M. and Stagge, P. (2003) Recurrent neural networks for time series classification. *Neurocomputing*, 50, pp. 223-235.

[52] Hussain, A. J., Ghazali, R. and Al-Jumeily, D. (2006-a). Dynamic Ridge polynomial neural network for financial time series prediction. *IEEE International conference on Innovation in Information Technology*, IIT06, Dubai.

[53] Hussain, A. J., Knowles, A., Lisboa, P., El-Deredy, W, & Al-Jumeily, D. (2006-b). Polynomial Pipelined Neural Network and Its Application to Financial Time Series Prediction. *Lecture Notes in Artificial Intelligence*, vol. 4304, pp. 597-606.

[54] Hussain, A. J. and Liatsis, P. (2002) Recurrent pi-sigma networks for DPCM image coding. *Neurocomputing*, 55, pp. 363-382.

[55] Hyndman, R.J. (n.d.): *Time Series Data Library*. Accessed on September 2005, downloaded from: http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/.

[56] Ismail, A. and Engelbrecht, A.P. (2002) Pruning Product Unit Neural Networks. *IEEE international joint conference on neural networks*. Honolulu, Hawai.

[57] Ivakhnenko, A. G., (1971) Polynomial Theory of Complex Systems. *IEEE transactions on Systems, Man, and Cybernetics,* SMC-1, 4, pp. 364-378.

[58] Jordan, M. I. (1986) Serial order: A parallel distributed processing approach. Institute for Cognitive Science report 8604, Institute for Cognitive Science, University of California, San Diego.

[59] Kaastra, I. and Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing 10*, pp. 215-236.

[60] Kaita, T., Tomita, S. and Yamanaka, J. (2002) On a Higher-order Neural Network for Distortion Invariant Pattern Recognition. *Pattern Recognition Letters 23*, pp 977-984.

[61] Karnavas, Y.L., and Papadopoulos, D.P. (2004) Excitation Control of a Synchronous Machine using Polynomial Neural Networks. *Journal of ELECTRICAL ENGINEERING*, Vol. 55, No. 7-8, pp. 169-179.

[62] Kim, S. S. (1998) Time delay recurrent neural network for temporal correlations and prediction. *Neurocomputing*, 20, pp. 253-263.

[63] Kimura, M. and Nakano, R. (2000) Systems and Computers in Japan. Vol. 31, No. 4, 2000, pp.77-86.

[64] Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[65] Knowles. A. C., (2005). *Higher-Order and Pipelined Neural Networks for Time Series Prediction of Currency Exchange Rates*. A Master Thesis, Liverpool John Moores University, UK.

[66] Kuan, C. M. and Liu, T. (1995). Forecasting exchange rates using feedforward and recurrent neural networks. *Journal of Applied Economics*, 10, pp. 347-364.

[67] Kuan, C.M. (1989). *Estimation of neural network models*. Ph.D. Thesis, University of California, San Diego.

[68] Kuo, C. and Reitsch, A. (1995-96). Neural networks vs Conventional Methods of Forecasting. *The journal of business forecasting*. Winter 1995-96, pp.17-22.

[69] Lawrence, S. and Giles, C. L. (2000) Overfitting and Neural Networks: Conjugate Gradient and Backpropagation. *International Joint Conference on Neural Networks*, Como, Italy, July 24–27, IEEE Computer Society, Los Alamitos, CA, pp. 114–119.

[70] Leerink, L. R., Giles, C. L., Horne, B. G. and Jabri, M. A. (1995) Learning with product units. In G. Tesaro, D. Touretzky, and T. Leen (Ed.) *Advances in Neural Information Processing Systems 7*, pp. 537-544, MIT Press, Cambridge, MA.

[71] Lendasse, A., Bodt, E.D., Wertz, V. and Verleysen, M. (2000). Non-linear financial time series forecasting – Application to the Bel 20 stock market index. *European Journal of Economic and Social Systems*, 14 N 1, pp.81-91.

[72] Leung, M. T., Chen, A. S. and Daouk, H. (2000) Forecasting exchange rates using general regression neural networks. *Computers & Operations Research 27*, pp. 1093-1110.

[73] Liatsis P., Hussain A. J. (1999) Nonlinear one-dimensional DPCM image prediction using polynomial neural networks. *In Proc. SPIE - Applications of Artificial Neural Networks in Image Processing IV*, pages 58-68, San Jose, California.

[74] Magdon-Ismail, M., Nicholson, A., Abu-Mostafa, Y.S. (1998). Financial Market: Very Noisy Information Processing. *Proceedings of the IEEE*, Vol. 86, No.11, pp. 2184-2195.

[75] Masters, T. (1993). *Practical Neural Network Recipes in C++*. Academic Press, New York.

[76] Merton R.C. (1981). On Market Timing and Investment Performance, I: An Equilibrium Theory of Value for Market Forecasts. *Journal of Business*. 54, no. 3, pp. 363-406.

[77] Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge.

[78] Mirea, L. and Marcu, T. (2002) System identification using Functional-Link Neural Networks with dynamic structure. *15th Triennial World Congress*, Barcelona, Spain.

[79] Moody, J. (1995). Economic Forecasting: Challenges and neural Network Solutions. *In Proceedings of the International Symposium on Artificial Neural Networks*, Hsinchu, Taiwan.

[80] Moody, S. L., Wise, S. P., Pellegrino, G. and Zipser, D. (1998) A Model That Accounts for Activity in Primate Frontal Cortex during a Delayed Matching-to-Sample Task. *The Journal of Neuroscience*, 18(1), pp: 399–410.

[81] Mozer, M.C. (1989) A Focused Backpropagation Algorithm for Temporal Pattern Recognition. *Complex System*, 3, pp 349-381.

[82] Neville, R.S., Stonham, T.J., and Glover, R.J. (2000) Partially pre-calculated weights for the backpropagation learning regime and high accuracy function mapping using continuous input RAM-based sigma-pi nets. *Neural Networks 13*, pp. 91-110.

[83] Nikolaev, N. Y. (2006). *CIS 311 Neural Networks*. Department of Computing, Goldsmiths College, University of London [online]. Available from: http://homepages.gold.ac.uk/nikolaev/cis311.htm

[84] Nikolaev N. Y., Iba, H. (2003). Learning Polynomial Feedforward Neural Networks by Genetic Programming and Backpropagation. *IEEE Transactions on Neural Networks*, Vol. 14, No. 2, pp.337-350.

[85] O'Connor, J.J and Robertson, E.F. (2000). Paramesvara, *MacTutor History of Mathematics archive*. Available from: http://www.answers.com/topic/mean-value-theorem?cat=technology

[86] Oh, S. K., Pedrycz, W., and Park, B. J. (2003). Polynomial neural networks architecture: analysis and design. *Computer and Electrical Engineering 29*, pp.703-725.

[87] Omlin, C. and Giles, C. L. (1996). Constructing Deterministic Finite-State Automata in Recurrent Neural Networks. *Journal of the ACM (JACM)*, Volume 43, Issue 6, pp. 937 - 972

[88] Pao, Y.H. (1989). *Adaptive pattern recognition and neural networks*. Addison-Wesley, USA. ISBN: 0-201012584-6.

[89] Park, S., Smith, M.J.T., and Mersereau, R.M. (2000). Target recognition based on directional filter banks and higher-Order Neural Networks. *Digital Signal Processing 10*, pp. 297-308.

[90] Patra, J.C., Bos, A.V.D. (2000). Modeling of an intelligent pressure sensor using functional link artificial neural networks. *ISA Transactions 39*, pp. 15-27.

[91] Patra, J.C., Pal, R.N. (1995). A functional link artificial neural network for adaptive channel equalization. *Signal Processing 43*. pp. 181-195.

[92] Patterson, D. W. (1996). *Artificial Neural Networks: Theory and Applications*. Prentice Hall, Singapore.

[93] Pau, Y.H, Phillips S.M (1995). The Functional Link Net and learning optimal control. *Neurocomputing* 9, pp 149-164.

[94] Plummer, E. A. (2000). *Time series forecasting with feed-forward neural networks: Guidelines and limitations*. Master of Science in Computer Science, University of Wyoming. Available from: http://www.karlbranting.net/papers/plummer/Paper_7_12_00.htm

[95] Reyes, J.R., Yu, W. and Poznyak, A.S. (2000). Passivation and Control of Partially Known SISO Nonlinear Systems via Dynamic Neural Networks. *Mathematical Problems in Engineering Volume 6*, pp. 61-83.

[96] Richmond, K. (2002). *Estimating Articulatory Parameters from the Acoustic Speech Signal*. PhD thesis, The Centre for Speech Technology Research, Edinburgh University

[97] Rumelhart, D.E., Hinton, G.E., and Williams, G.E. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Ed.), *Parallel distributed processing*, vol.1 (pp. 318-362). The MIT Press.

[98] Sanzogni, L., Chan, R., and Bonner, R. F. (2000). Perceptrons with polynomial post-processing. *J. EXPT. THEOR. ARTIF. INTELL. 12*, pp. 57-68.

[99] Sarle, W. S. (2002). *AI FAQ/neural Nets*. Available from: http://www.faqs.org/faqs/ai-faq/neural-nets/, SAS Institute Inc., USA, 2002.

[100] Schmitt, M (2001-a). On the complexity of computing and learning with multiplicative neural networks. *Neural Computation, 14*, pp. 241-301.

[101] Schmitt, M. (2001-b). Product Unit Neural Networks with Constant Depth and Superlinear VC Dimension. *Proceedings of the International Conference on Artificial Neural Networks ICANN 2001*, Lecture Notes in Computer Science, volume 2130, pages 253-258, Springer-Verlag, Berlin.

[102] Schwaerzel, R. (1996). *Improving the prediction accuracy of financial time series by using multi-neural network systems and enhanced data preprocessing*. Thesis, Master of Science, The University of Texas at San Antonio.

[103] Senjyu, T. (2002). One-Hour-Ahead Load Forecasting using Neural Network. *IEEE Transactions on Power Systems*, Vol.17, No.1, pp. 113-118.

[104] Shachmurove, Y. and Witkowska, D. (2000). *Utilizing Artificial Neural Network model to predict stock markets*. CARESS Working Paper, Series No. 00-11, Pennsylvania: University of Pennsylvania, Center for Analytic Research in Economics and the Social Sciences.

[105] Shieh, J. S., Chou, C. F. Huang, S. J. and Kao, M. C. (2004). Intracranial pressure model in intensive care unit using a simple recurrent neural network through time. *Neurocomputing 57*, pp. 239-256.

[106] Shin, Y. and Ghosh, J. (1992). Approximation of Multivariate Functions using Ridge Polynomial Networks. Proc. IJCNN, Baltimore. pp. II: 380-385.

[107] Shin, Y. and Ghosh, J. (1991-a). Realization of Boolean Functions using Binary Pi-Sigma Networks. In Kumara & Shin, (Ed.), *Intelligent Engineering Systems Through Artificial Neural Networks, Dagli*, (pp. 205-210). ASME Press.

[108] Shin, Y. and Ghosh, J. (1991-b). The Pi-Sigma Networks: An Efficient Higher-order Neural Network for Pattern Classification and Function Approximation. *Proceedings of International Joint Conference on Neural Networks, Vol.1*, pp.13-18.

[109] Shin, Y. and Ghosh, J. (1995). Ridge Polynomial Networks. *IEEE Transactions on Neural Networks*, Vol.6, No.3, pp.610-622.

[110] Shin, Y., Ghosh, J. and Samani, D. (1992). Computationally Efficient Invariant Pattern Classification with Higher-order Pi-sigma Networks. In Burke and Shin, (Ed), *Intelligent Engineering Systems Through Artificial Neural Networks-II, Dagli*, (pp. 379-384). ASME Press.

[111] Sierra, A., Mac J.A. and Corbacho, F. (2001). Evolution of functional link networks. In *Transactions on Evolutionary Computation*. Vol. 5, No. 1. Pages 54-65.

[112] Sitte, R. and Sitte, J. (2000). Analysis of the Predictive Ability of Time Delay Neural Networks Applied to the S&P 500 Time Series. *IEEE Transaction on Systems, Man, and Cybernetics-part C: Applications and Reviews*, vol. 30, No. 4. pp. 568-572.

[113] Steil, J.J. (2006). Online stability of backpropagation-decorrelation recurrent learning. *Neurocomputing 69*. pp.642-650.

[114] Steil, J.J. (2005). Stability of backpropagation-decorrelation efficient O(N) recurrent learning. ESANN'2005 proceedings - European Symposium on Artificial Neural Networks, Belgium. ISBN 2-930307-05-6, pp. 43-48.

[115] Tawfik, H. and Liatsis, P. (1997). Prediction of non-linear time-series using Higher-Order Neural Networks. *Proceeding IWSSIP'97 Conference*, Poznan, Poland.

[116] Tenorio, M.F. and Lee, W.T. (1990). Self-organizing network for optimum supervised learning. *IEEE Transactions on Neural Networks, vol.1*, no.1, pp. 100-110.

[117] Tenti, P. (1996). Forecasting Foreign Exchange Rates Using Recurrent Neural Networks. *Applied Intelligence, 10*: pp. 567-581.

[118] Tertois, S.,Glaunec, A.L., and Vaucher, G. (2002). Compensating the non linear distortions of an OFDM signal using neural networks. In P. Liatsis (Ed.) *Recent Trends in Multimedia*

Information Processing, proceedings of IWSSIP'02, (pp. 484-488). World Scientific, ISBN 981-238-243-7.

[119] Thimm, G., (1995). Optimization of high order Perceptron. Swiss federal Institute of Technology (EPFL).

[120] Thimm, G and Fiesler, E. (1994). High Order and Multilayer Perceptron Initialization. Institute Dalle Molle D'Intelligence Artificial Perceptive (IDIAP) Technical Report, 97-04.

[121] Thimm, G and Fiesler, E. (1997). Optimal Setting of Weights, Learning Rate, and Gain. Institute Dalle Molle D'Intelligence Artificial Perceptive (IDIAP) Research Report, 97-04.

[122] Thomas, J. and Sycara, K. (1999). Integrating Genetic Algorithms and Text Learning for Financial Prediction. *Proceedings of the GECCO-2000 Workshop on Data Mining with Evolutionary Algorithms*.

[123] Thomason, M. (1998). The practitioner method. *Journal of Computational Intelligence in Finance*, vol. 6, no. 1, pp. 43-44.

[124] Thomason, M (1999-a). The practitioner method and tools. *Journal of Computational Intelligence in Finance*, vol. 7, no. 3, pp. 36-45.

[125] Thomason, M. (1999-b). The practitioner method and tools. *Journal of Computational Intelligence in Finance*, vol. 7, no. 4, pp. 35-45.

[126] UNIVERSITY OF NORTH CAROLINA AT CHARLOTTE. Department of computer Science. (2002). *Neural Networks*. Available from: http://www.sis.uncc.edu/~mirsad/itcs6265/NN.ppt#1

[127] Voutriaridis, C., Boutalis, Y. S. and Mertzios, G. (2003). Ridge Polynomial Networks in pattern recognition. EC-VIP-MC 2003, *4th EURASIP Conference focused on Video / Image Processing and Multimedia Communications*, Croatia. pp. 519-524.

[128] Walczak, S. (2001). An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems*, Vol.17, No. 4, pp. 203-222.

[129] Wang, Z. Fang, J., Liu, X. (2006). Global stability of stochastic high-order neural networks with discrete and distributed delays. *Chaos, Solitons and Fractals*. doi:10.1016/j.chaos.2006.06.063

[130] Williams, R.J., Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation, 1*. pp. 270-280.

[131] Yao, J. and Tan, C. L. (2000). A case study on neural networks to perform technical forecasting of forex. *Neurocomputing 34*, pp. 79-98.

[132] Yao, J., Poh, H. and Jasic, T. (1996). Foreign exchange rates forecasting with neural networks. *Proceedings of the International Conference on Neural Information Processing*, Hong Kong, pp. 754-759.

[133] Yao, J. and Tan, C. L. (2001). Guidelines for Financial Forecasting with Neural Networks. *Proceedings of International Conference on Neural Information Processing*, Shanghai, China, pp. 757-761.

[134] Yonghong, C., Yaolin, J., and Jianxue, X. (2003). Dynamic properties and a new learning mechanism in higher order neural networks. *Neurocomputing* 50, pp. 17-30.

[135] Yu, W. and Morales, A. (2005). Neural Networks for the Optimization of Crude Oil Blending. *International Journal of Neural Systems, Vol. 15*, No. 5, pp. 377-389.

[136] Yumlu, S., Gurgen, F.S., and Okay, N. (2005). A comparison of global, recurrent and smoothed-piecewise neural models for Istanbul stock exchange (ISE) prediction. *Pattern Recognition Letterts 26*, pp.2093-2103.

[137] Zaknich, A. (2003). *Neural Networks for Intelligent Signal Processing*. World scientific Publishing Co. Pte. Ltd.

[138] Zekić, M. (1998). Neural Network Applications in Stock Market Predictions - A Methodology Analysis. In Aurer, B., Logažar, R.,& Varaždin (Ed.) *Proceedings of the 9th International Conference on Information and Intelligent Systems '98*, (pp. 255-263), UDC: 007:681.3:651(082)

[139] Zhang, G.P. (2003). *Neural Networks in business forecasting*. Hershey, Pa, Idea.

[140] Zhang, G., Patuwo, B.E., and Hu, M.Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting, 14*, pp. 35-62.

[141] Zhang, Y.Q. and Chan, L.W. (2000). Fourier recurrent Networks for Time series Prediction. *Proceedings of International Conference on Neural Information Processing*, ICONIP 2000, Taejon, Korea, pp. 576-582.

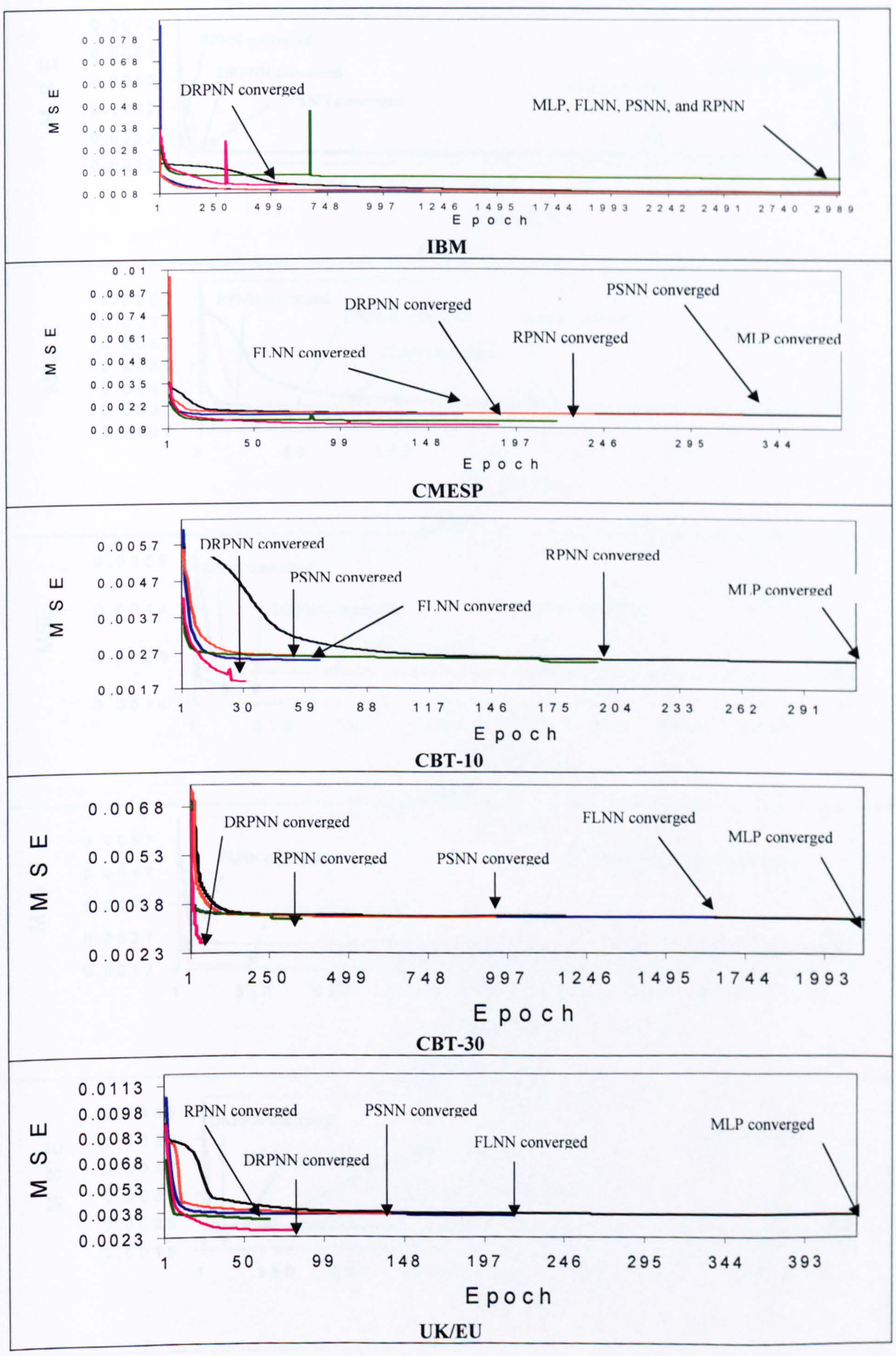
[142] Zumbach, G.O., Pictet, O.V. and Masutti, O., (2001). Genetic Programming with Syntactic Restrictions Applied to Financial Volatility Forecasting. Olsen & Associates Working Paper No. GOZ.2000-07-28. Available at SSRN: http://ssrn.com/abstract=269189 or DOI: 10.2139/ssrn.269189

APPENDICES

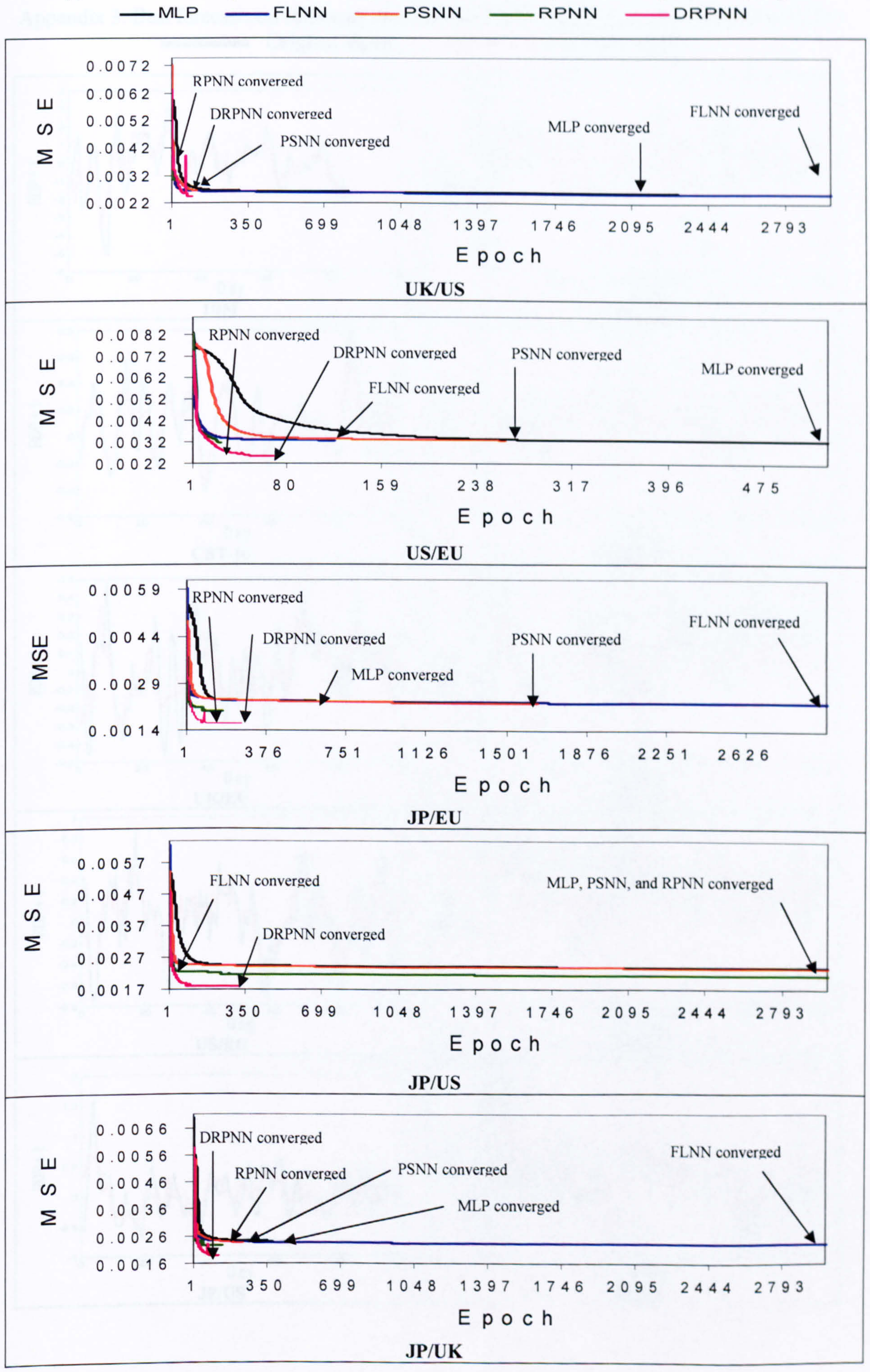
Appendix 1: Gantt Chart

	Year 1 (2004-2005)												Year 2 (2005-2006)												Year 3 (2006-2007)											
	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Research & Literature review																																				
Neural Networks																																				
Higher Order Neural Networks																																				
Functional Link Neural Network																																				
Pi-Sigma Neural Network																																				
Ridge Polynomial Neural Network																																				
Recurrent Neural Networks																																				
Financial Time Series Prediction																																				
Development																																				
Multilayer Perceptron																																				
Functional Link Neural Network																																				
Pi-Sigma Neural Network																																				
Ridge Polynomial Neural Network																																				
Dynamic Ridge Polynomial Neural Network																																				
Testing																																				
Multilayer Perceptron																																				
Functional Link Neural Network																																				
Pi-Sigma Neural Network																																				
Ridge Polynomial Neural Network																																				
Dynamic Ridge Polynomial Neural Network																																				
Analysis on Results																																				
Writing Up																																				
Thesis																																				
Transfer Report																																				
Conference Papers and Journals																																				

Appendix 2: Learning curves for the prediction of one step ahead using stationary signals

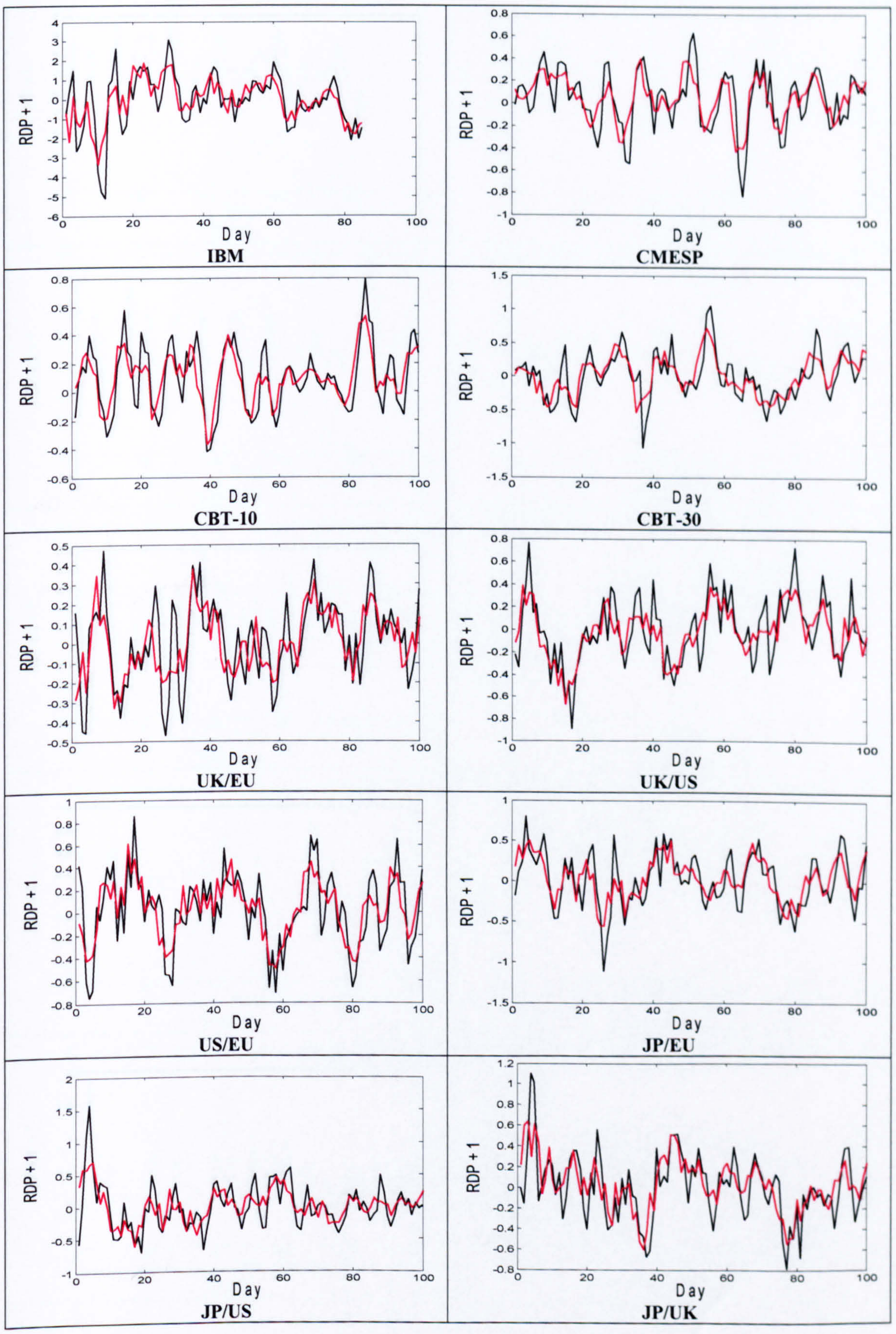


Appendix 2: Learning curves for the prediction of one step ahead using stationary signals



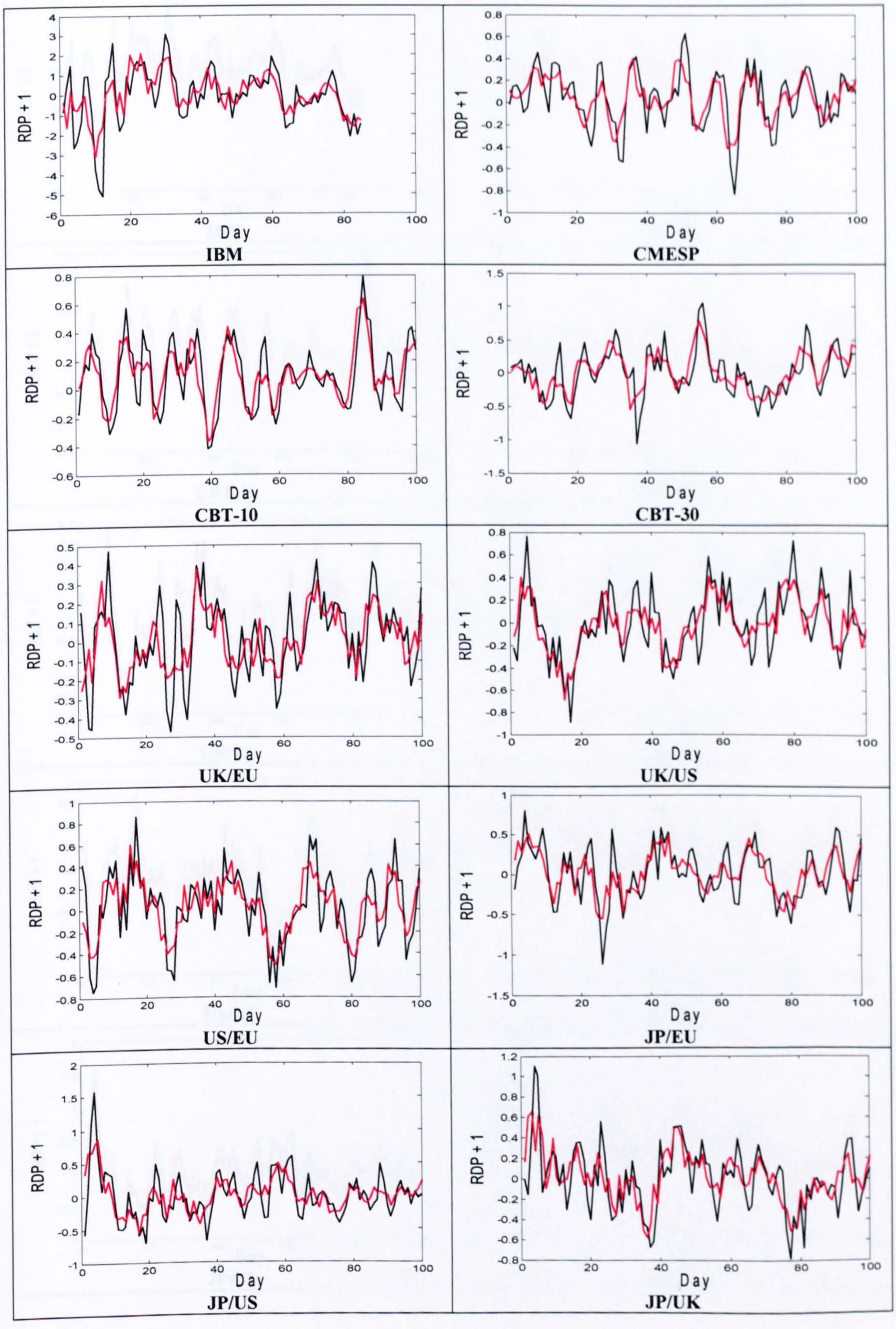
Appendix 3: Best forecasts on stationary data for the prediction of one step ahead using MLPs

Original signal, Predicted signal



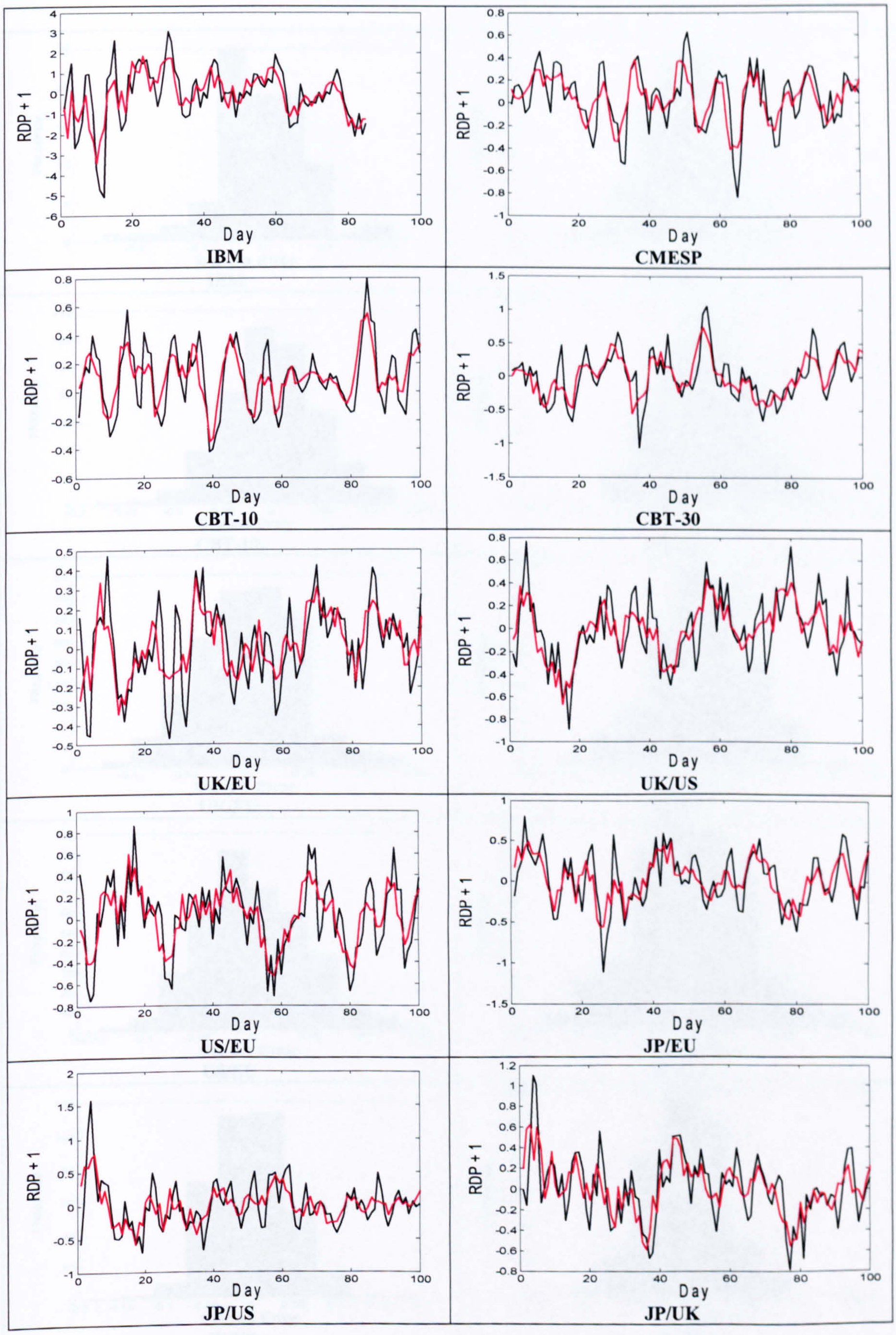
Appendix 3: Best forecasts on stationary data for the prediction of one step ahead using FLNNs

Original signal, Predicted signal

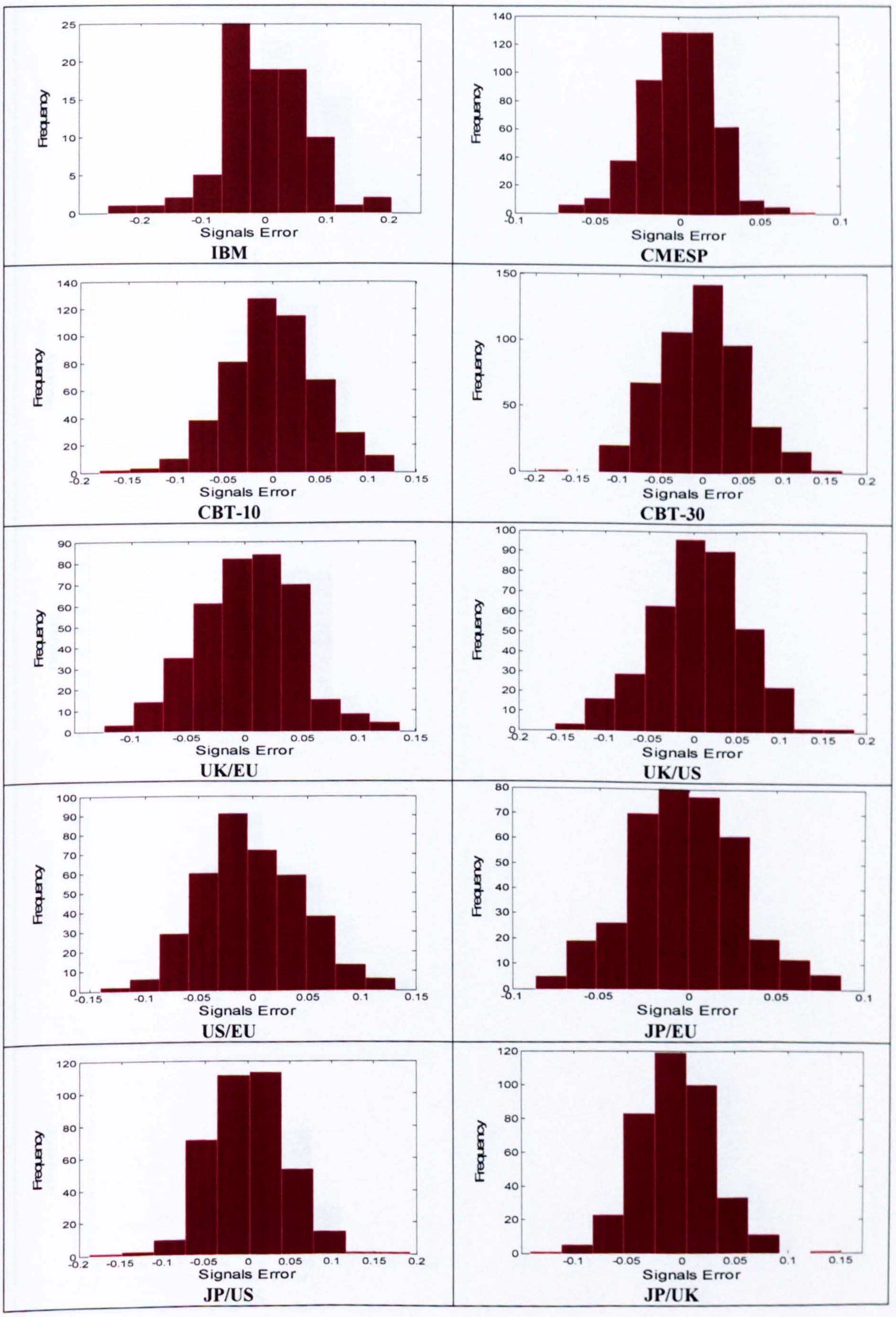


Appendix 3: Best forecasts on stationary data for the prediction of one step ahead using RPNNs

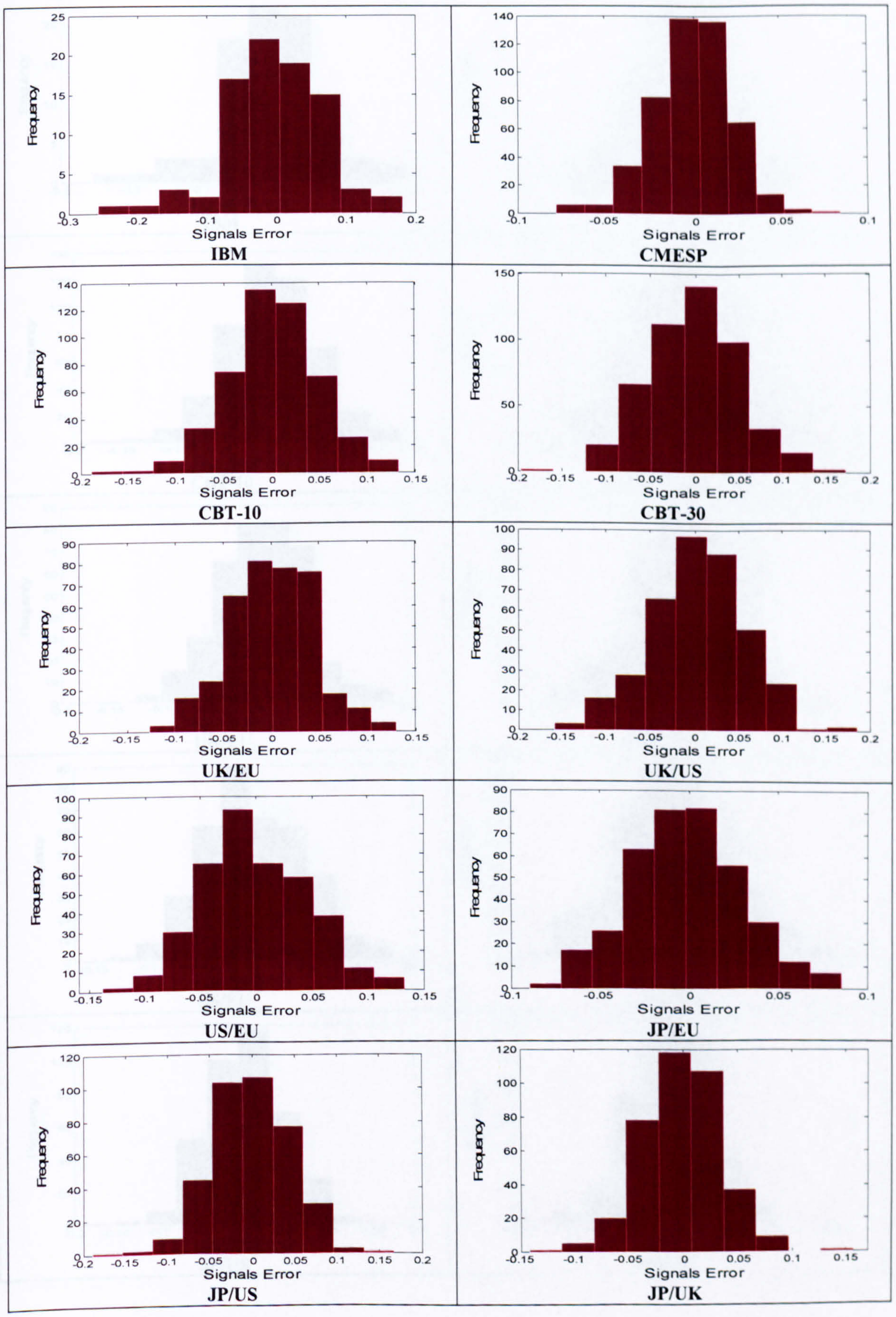
Original signal, Predicted signal



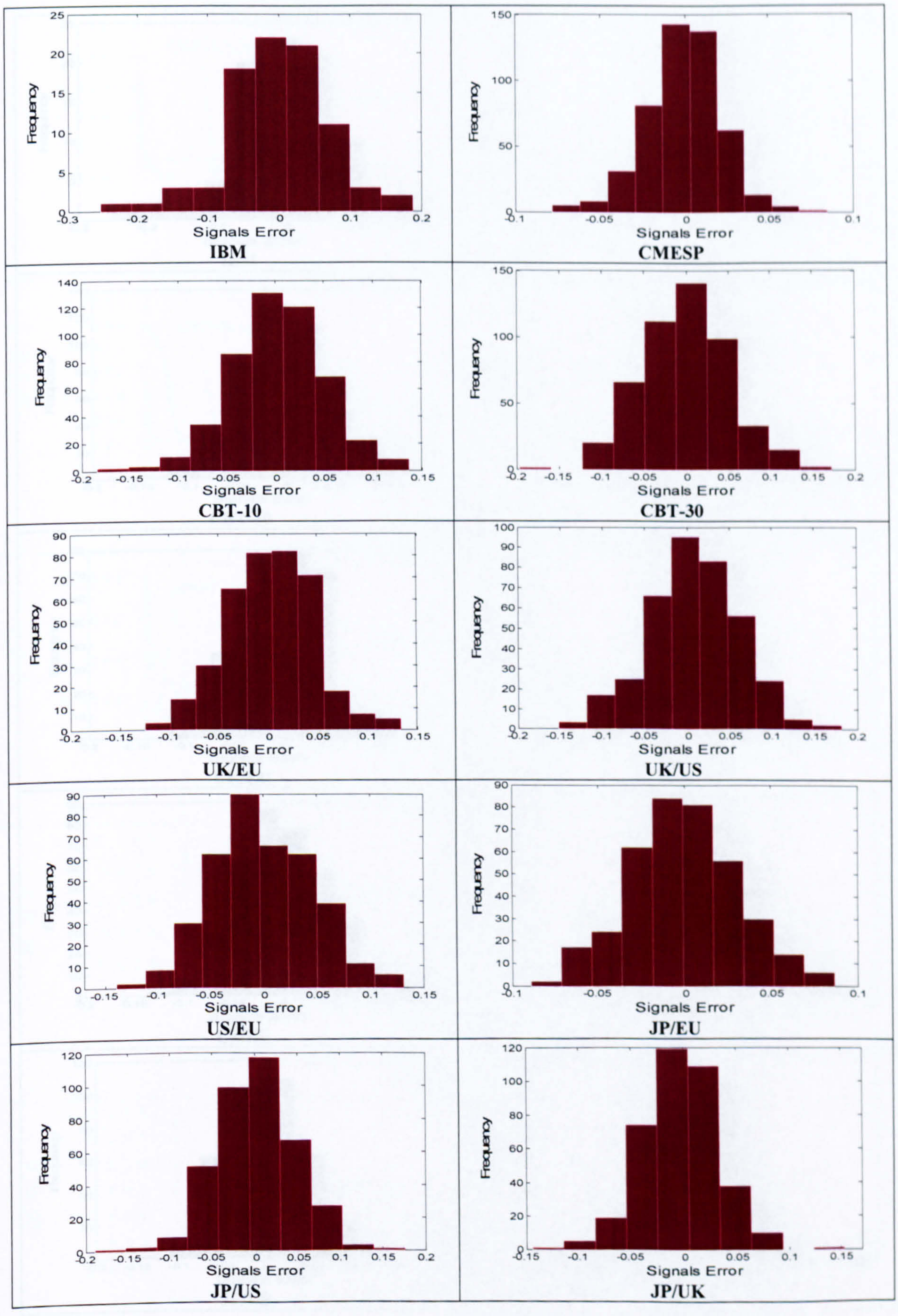
Appendix 4: Histograms of the signals error on stationary data for the prediction of one step ahead using MLPs



Appendix 4: Histograms of the signals error on stationary data for the prediction of one step ahead using FLNNs

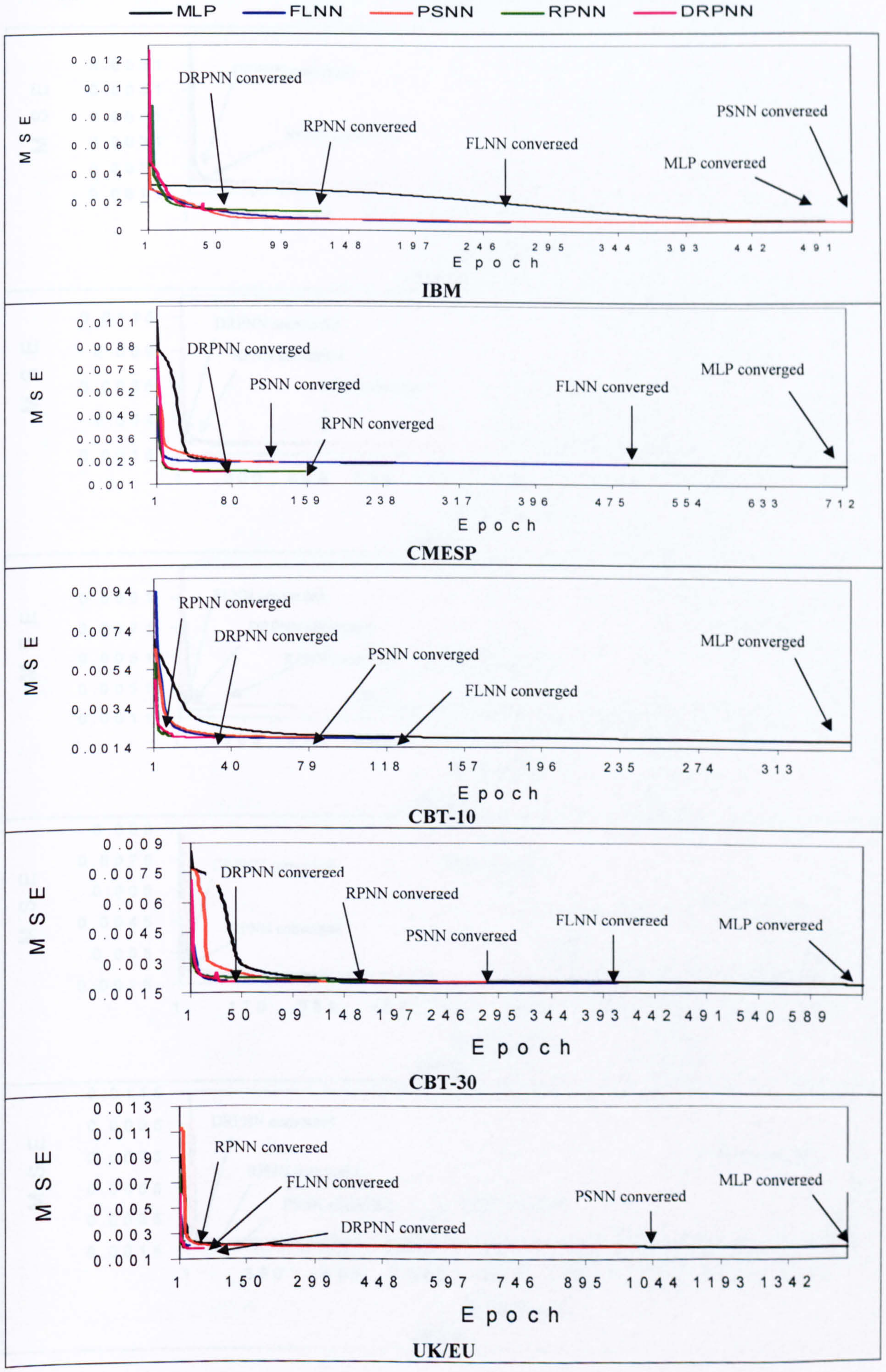


Appendix 4: Histograms of the signals error on stationary data for the prediction of one step ahead using PSNNs

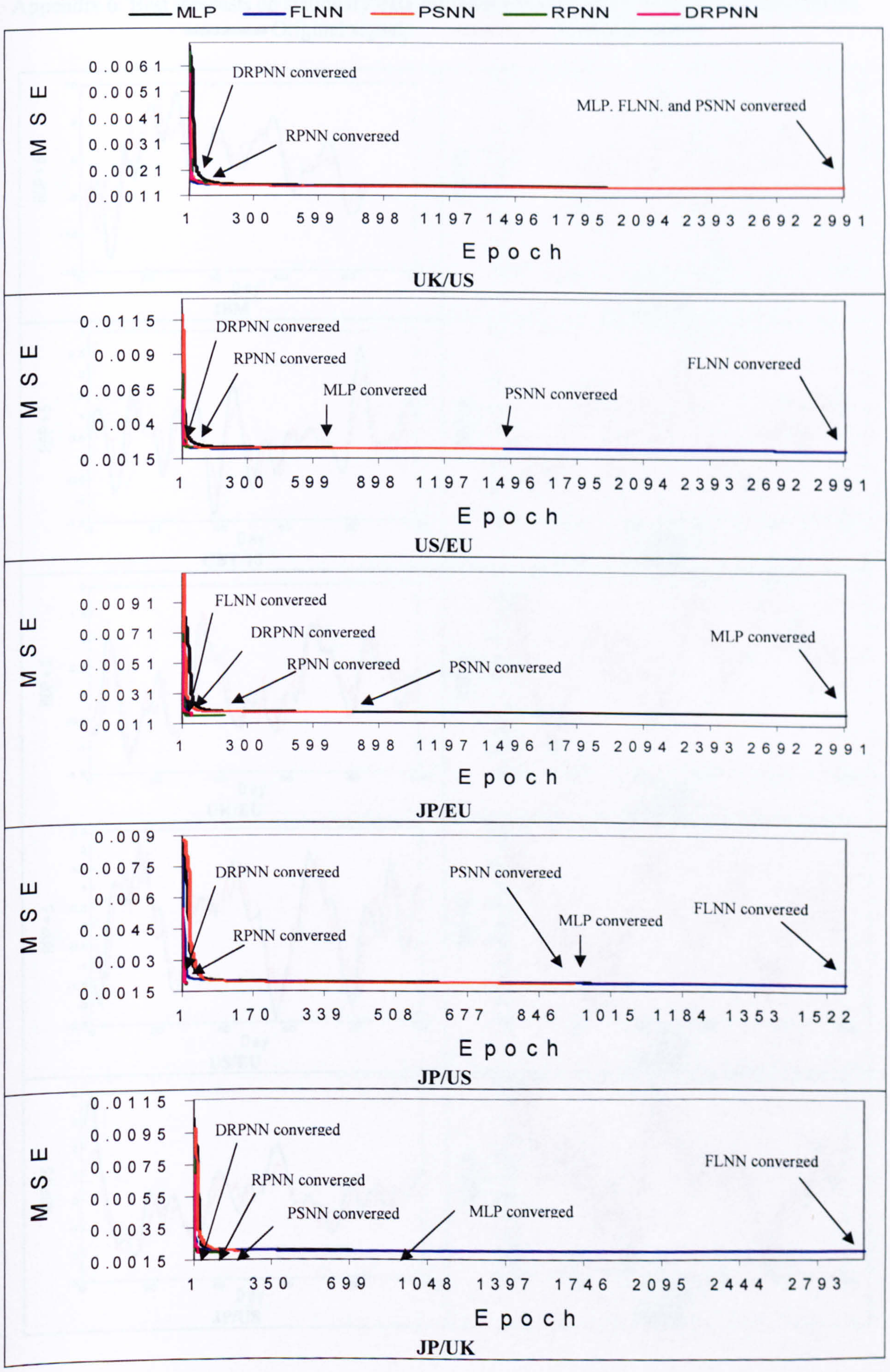


Appendix 4: Histograms of the signals error on stationary data for the prediction of one step ahead using RPNNs

Appendix 5: Learning curves for the prediction of five steps ahead using stationary signals

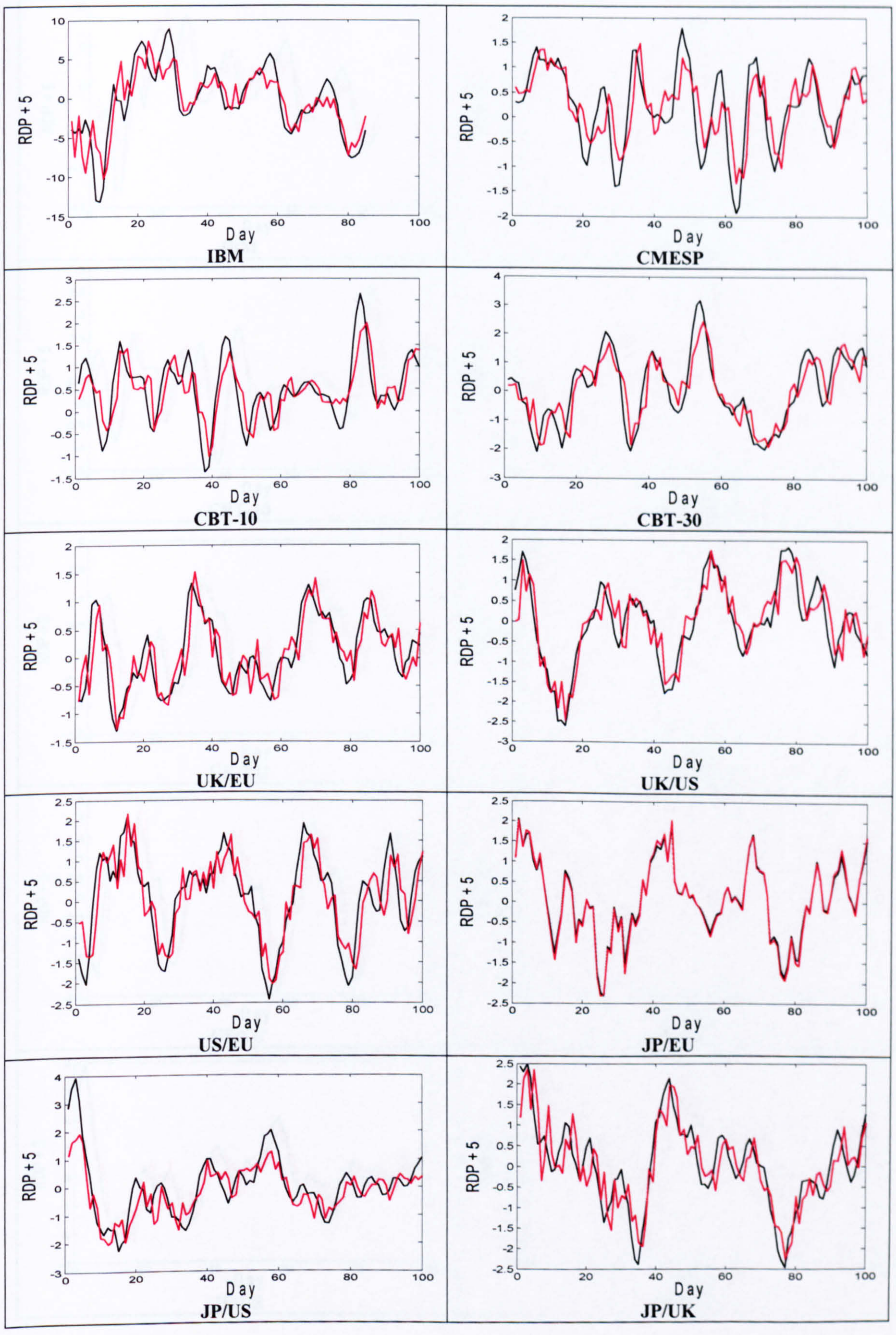


Appendix 5: Learning curves for the prediction of five steps ahead using stationary signals



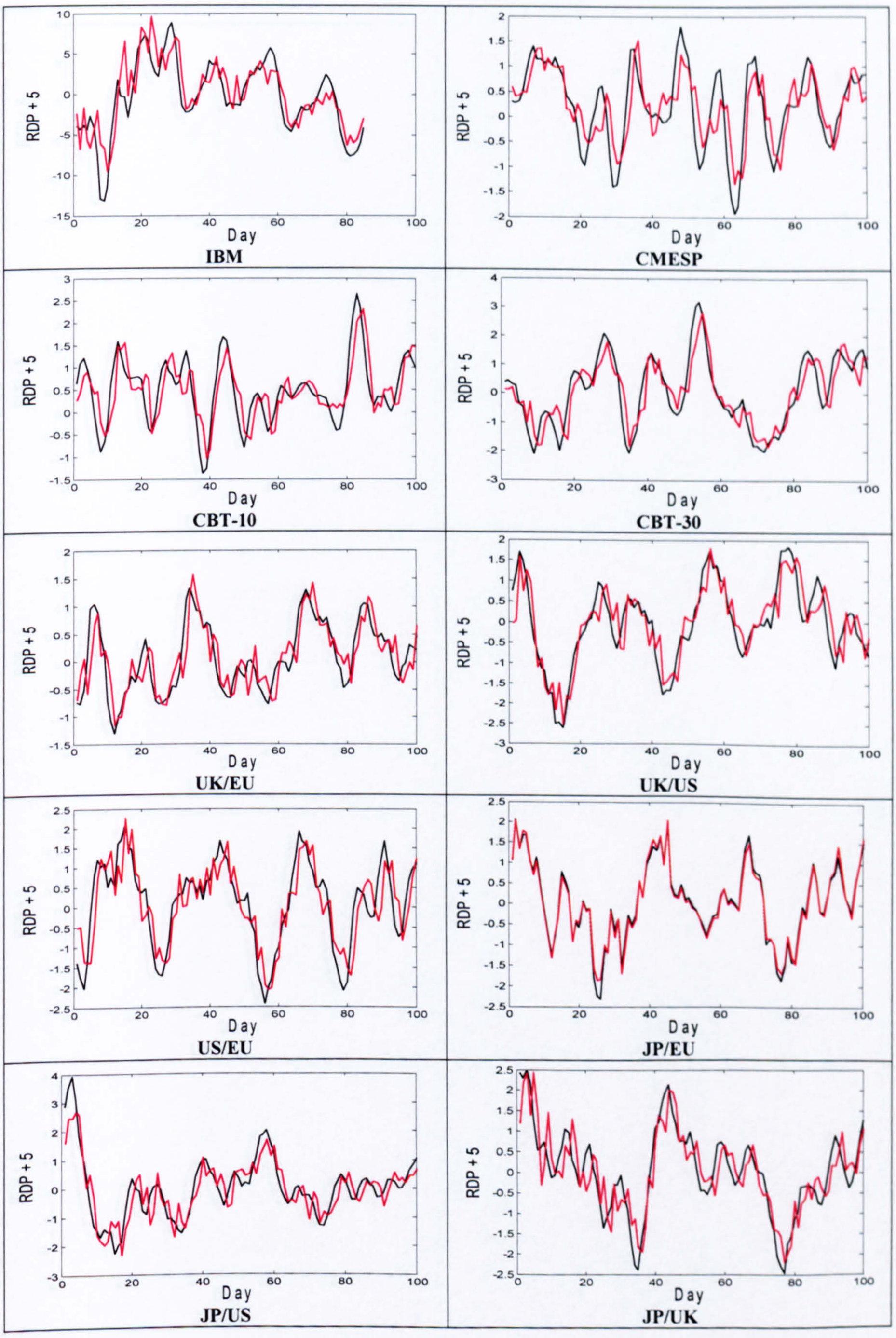
Appendix 6: Best forecasts on stationary data for the prediction of five steps ahead using MLPs

Original signal, Predicted signal



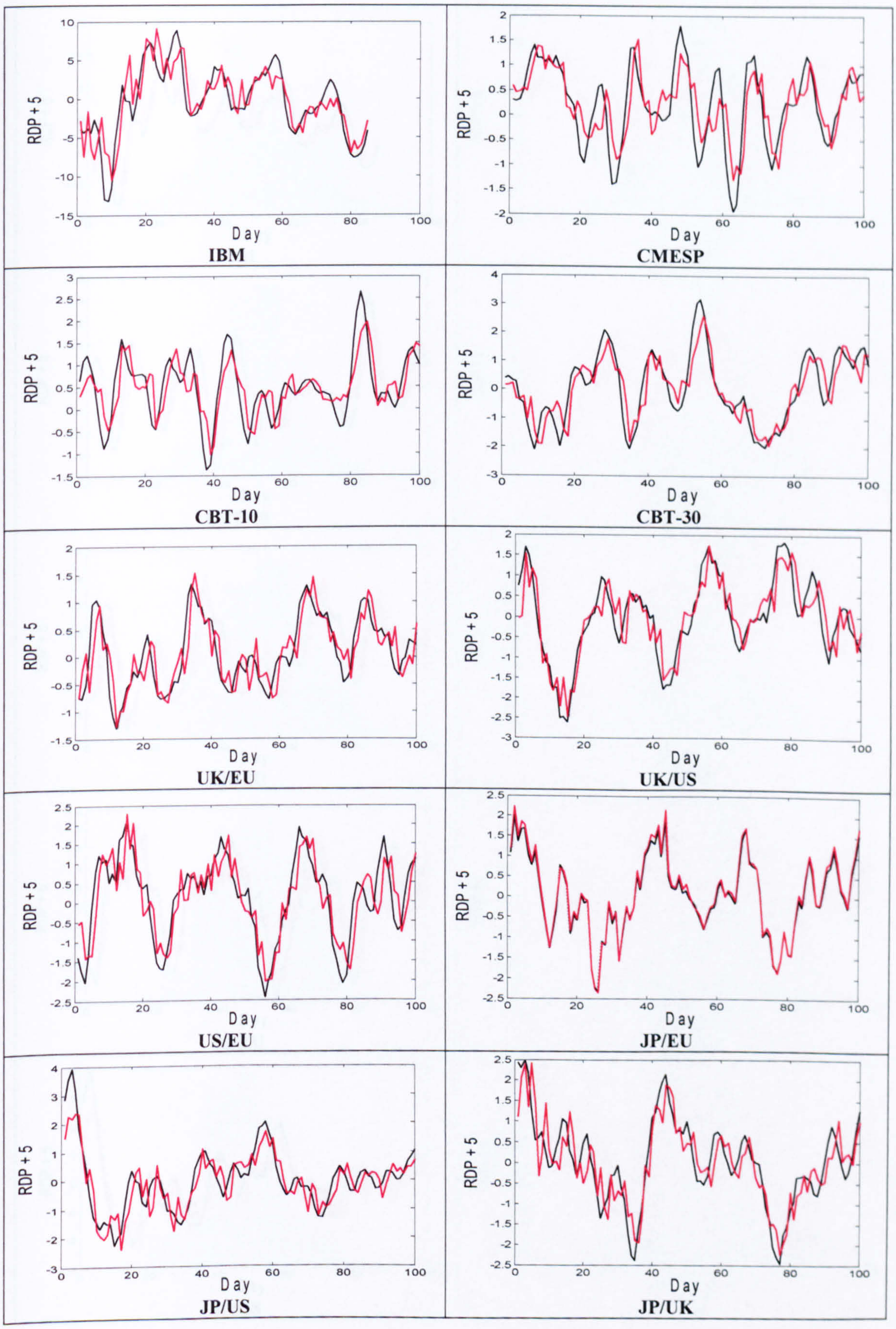
Appendix 6: Best forecasts on stationary data for the prediction of five steps ahead using FLNNs

Original signal, Predicted signal



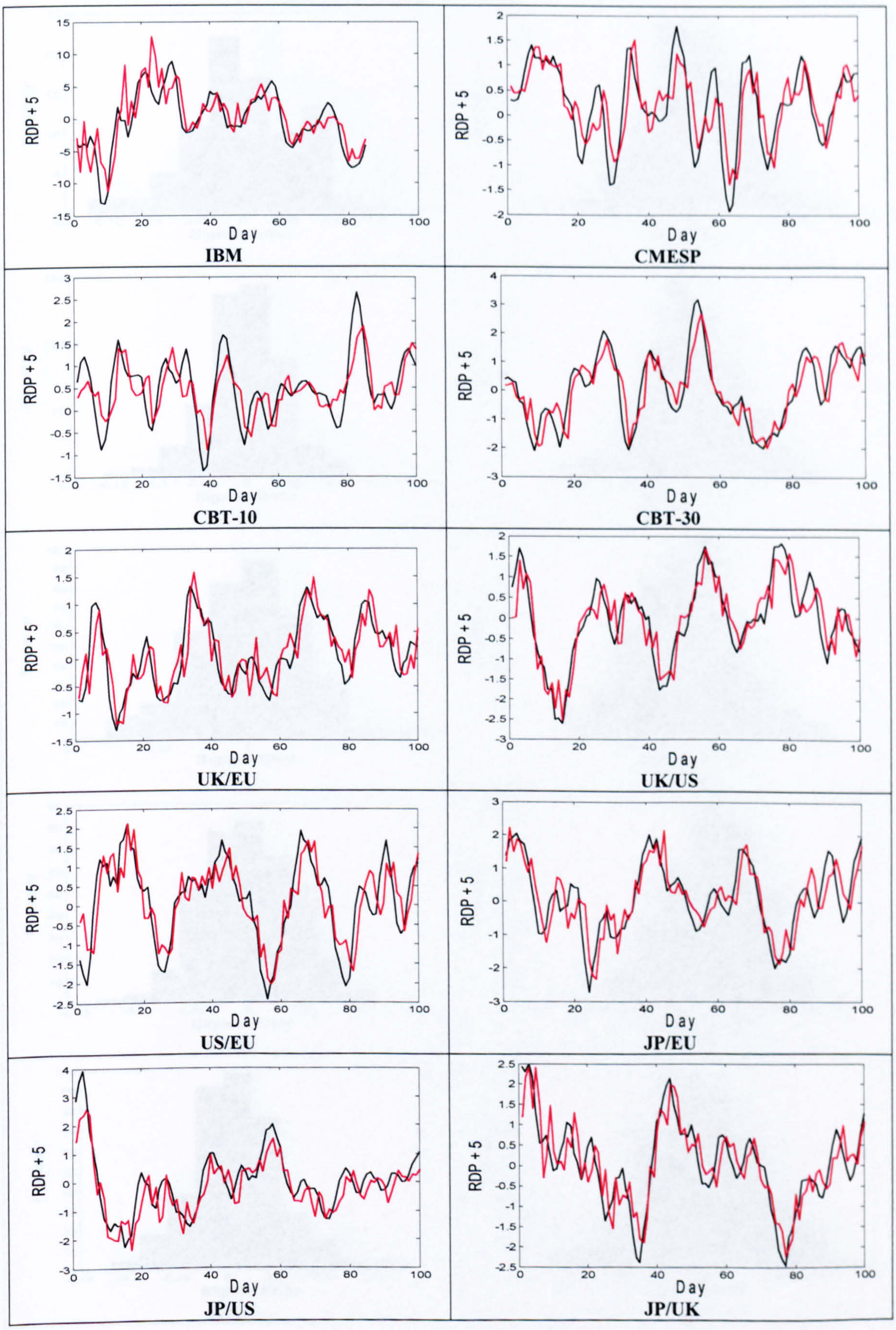
Appendix 6: Best forecasts on stationary data for the prediction of five steps ahead using PSNNs

Original signal, Predicted signal

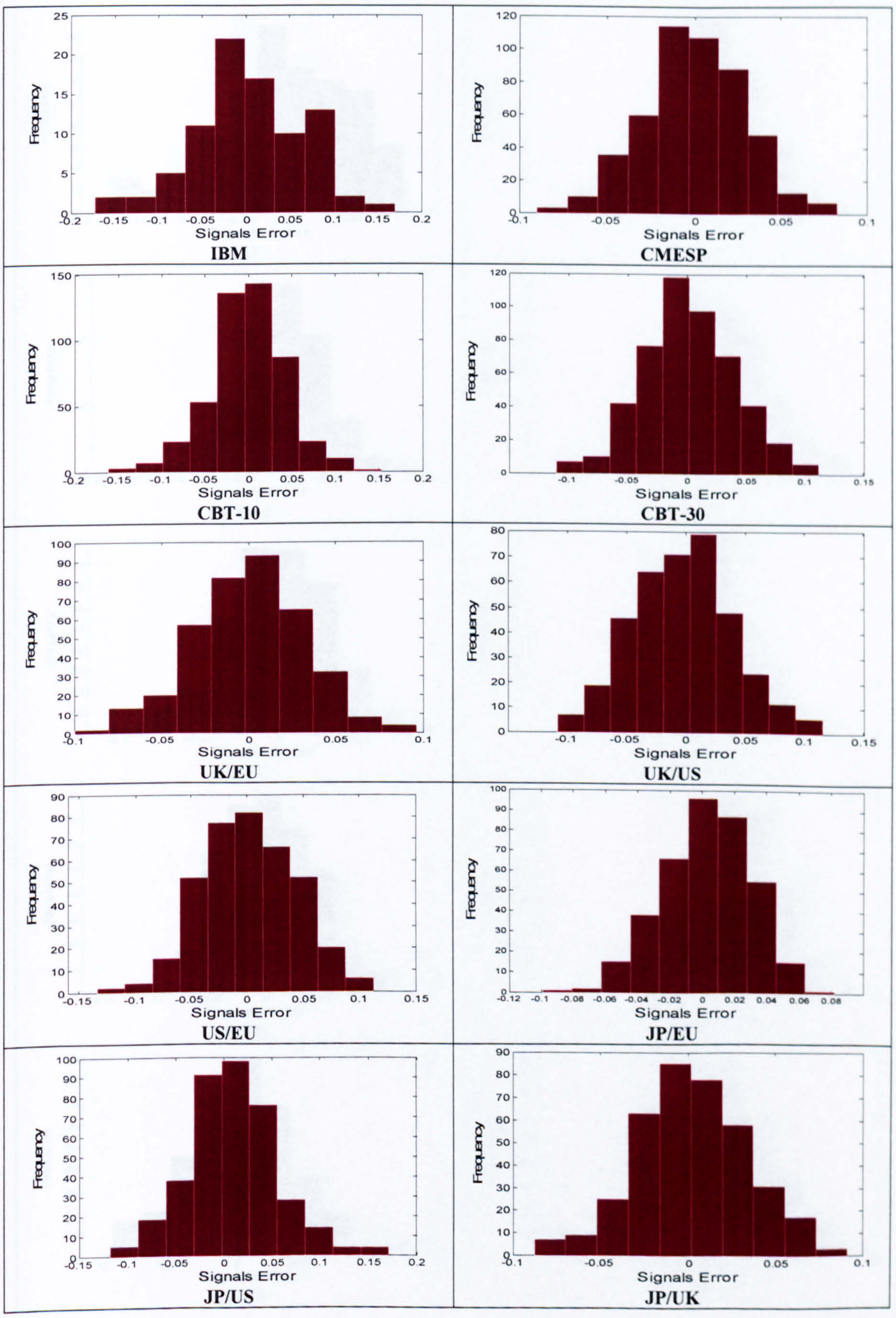


Appendix 6: Best forecasts on stationary data for the prediction of five steps ahead using RPNNs

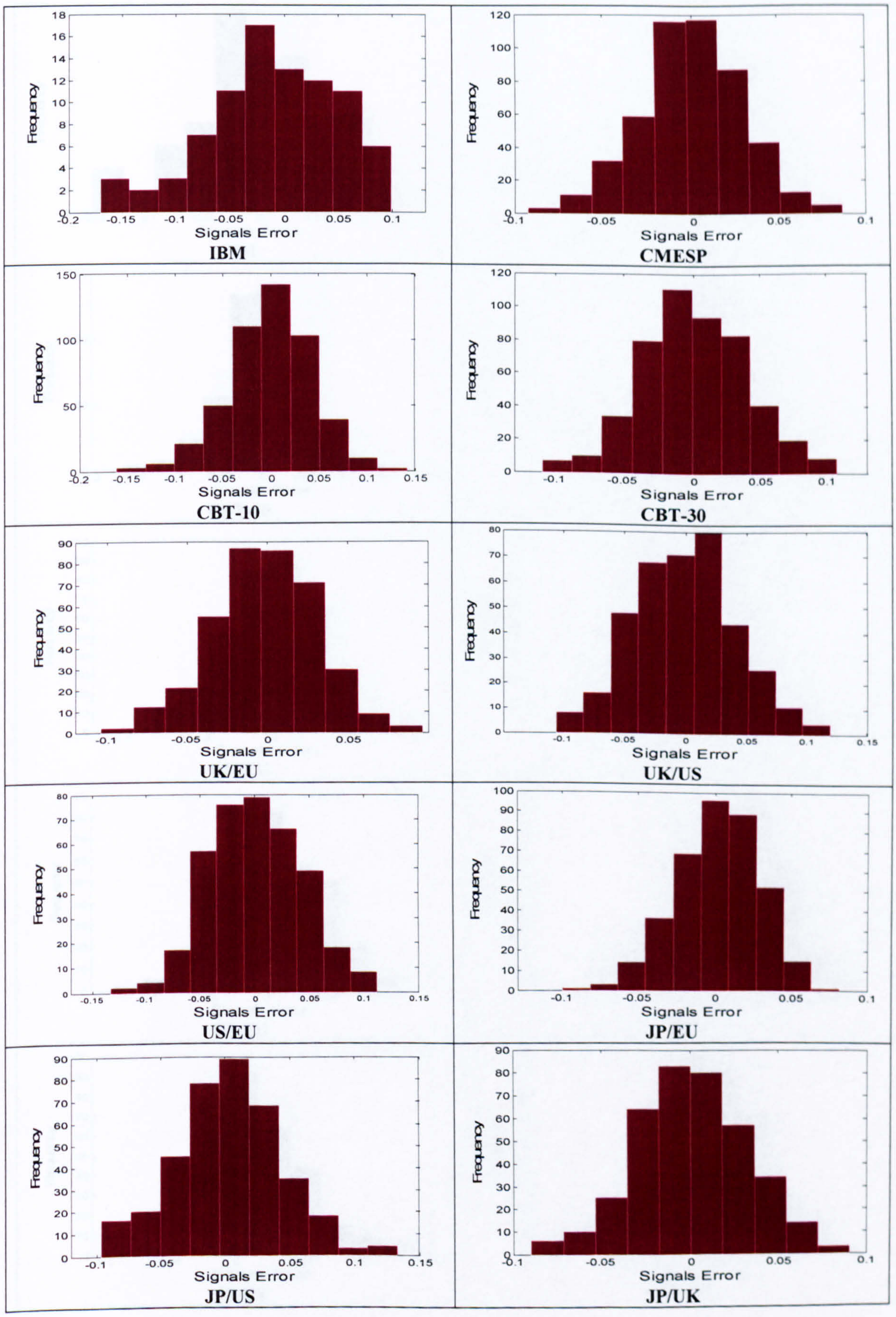
Original signal, Predicted signal



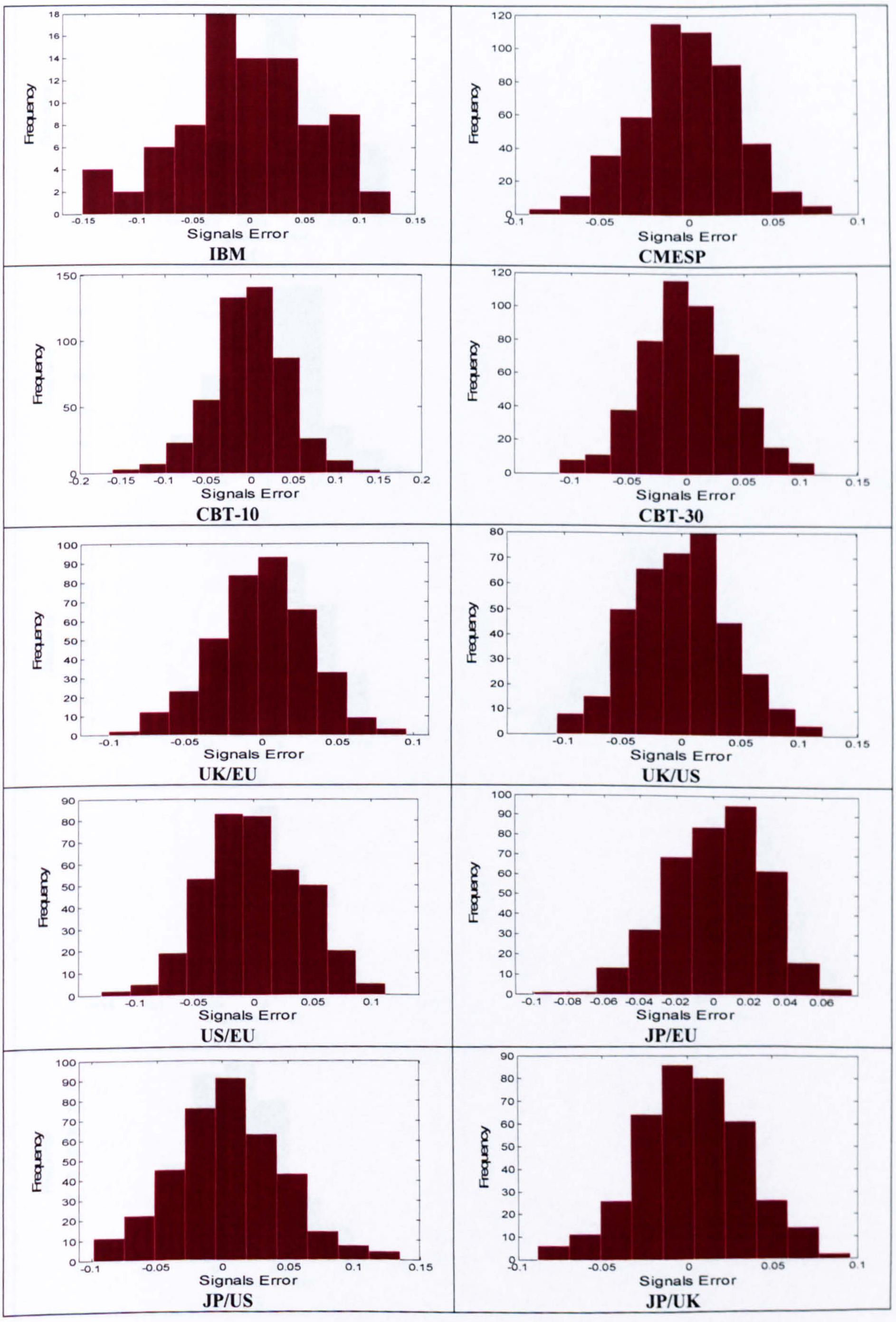
Appendix 7: Histograms of the signals error on stationary data
for the prediction of five steps ahead using MLPs



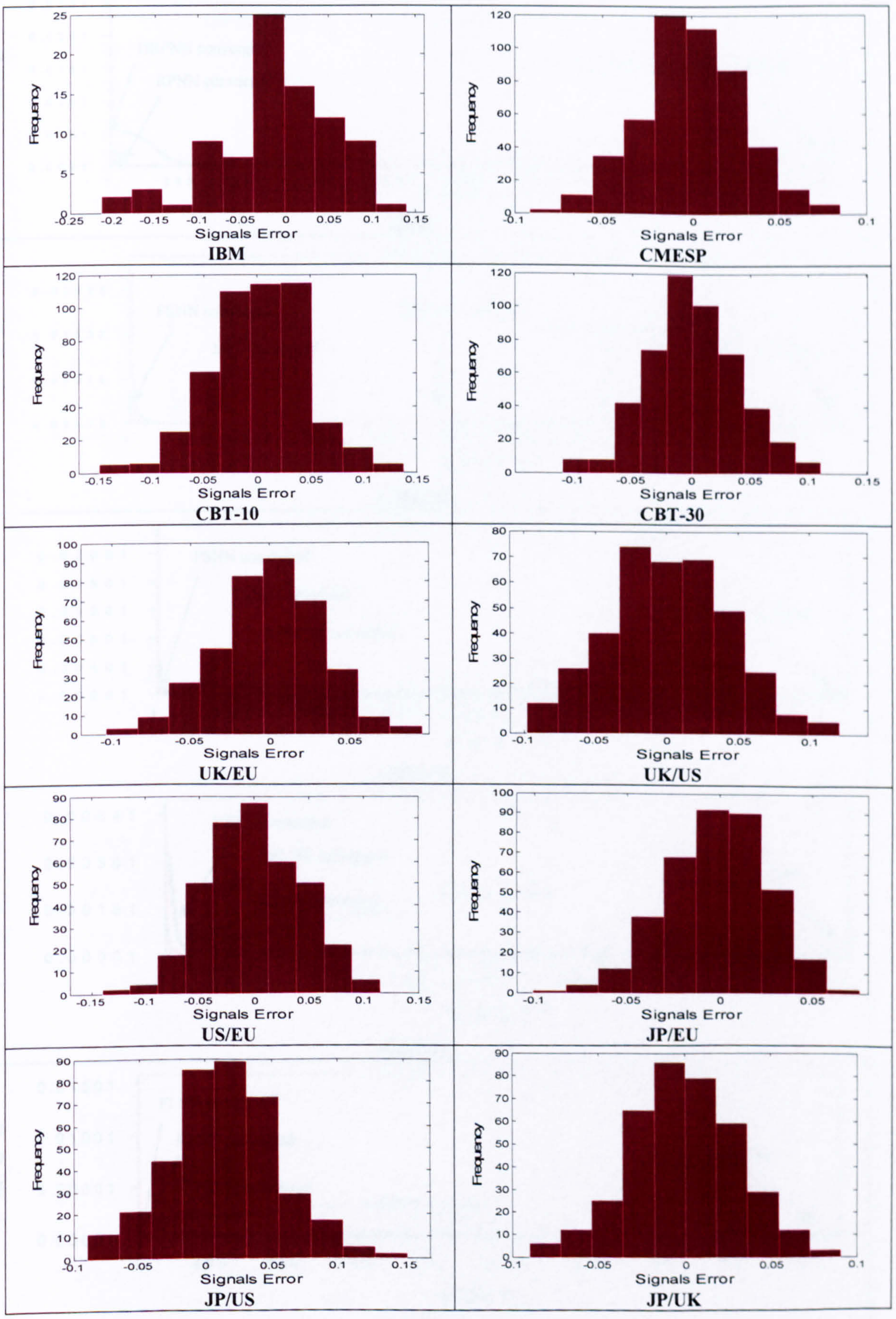
Appendix 7: Histograms of the signals error on stationary data for the prediction of five steps ahead using FLNNs



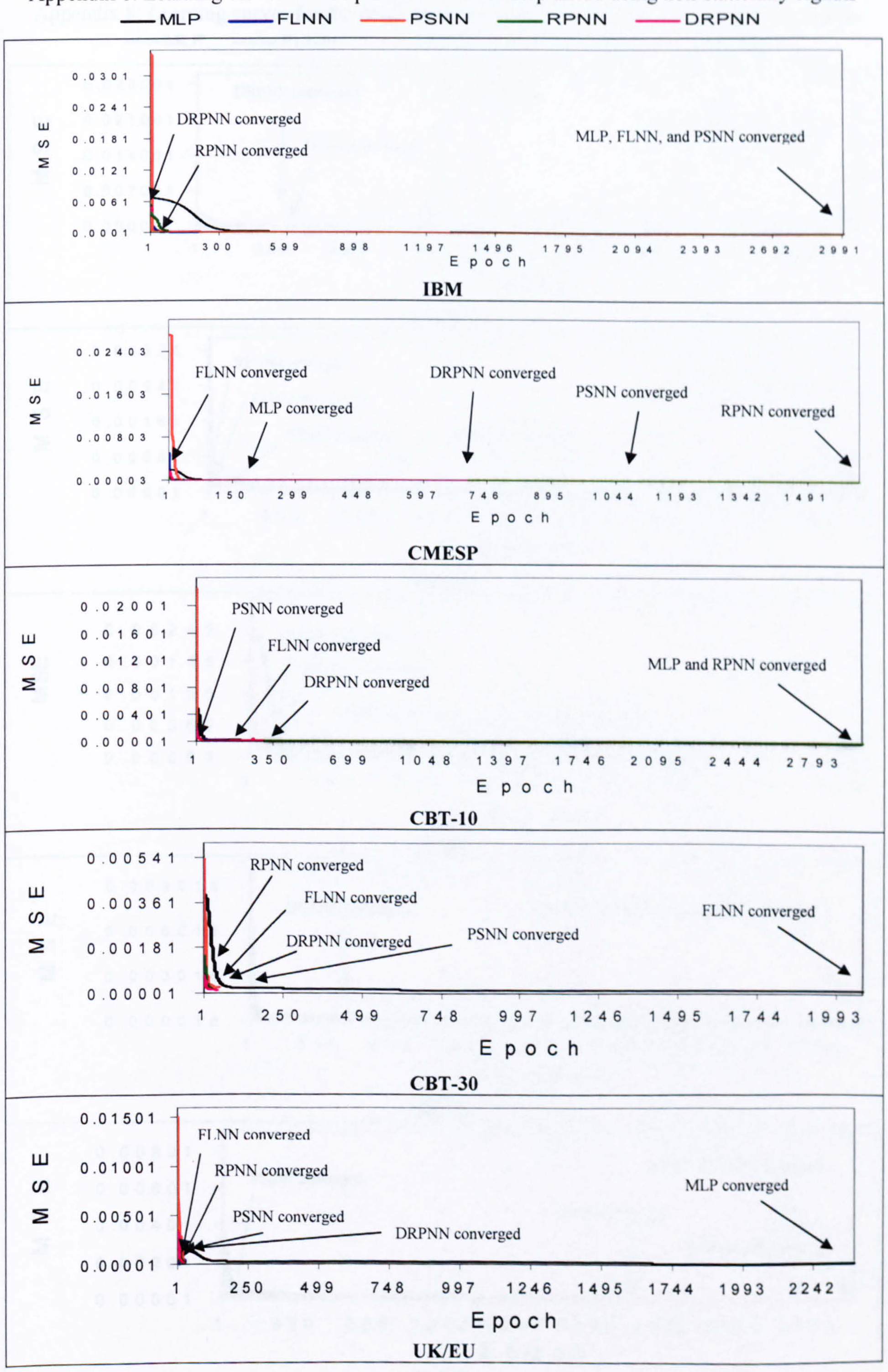
Appendix 7: Histograms of the signals error on stationary data for the prediction of five steps ahead using PSNNs



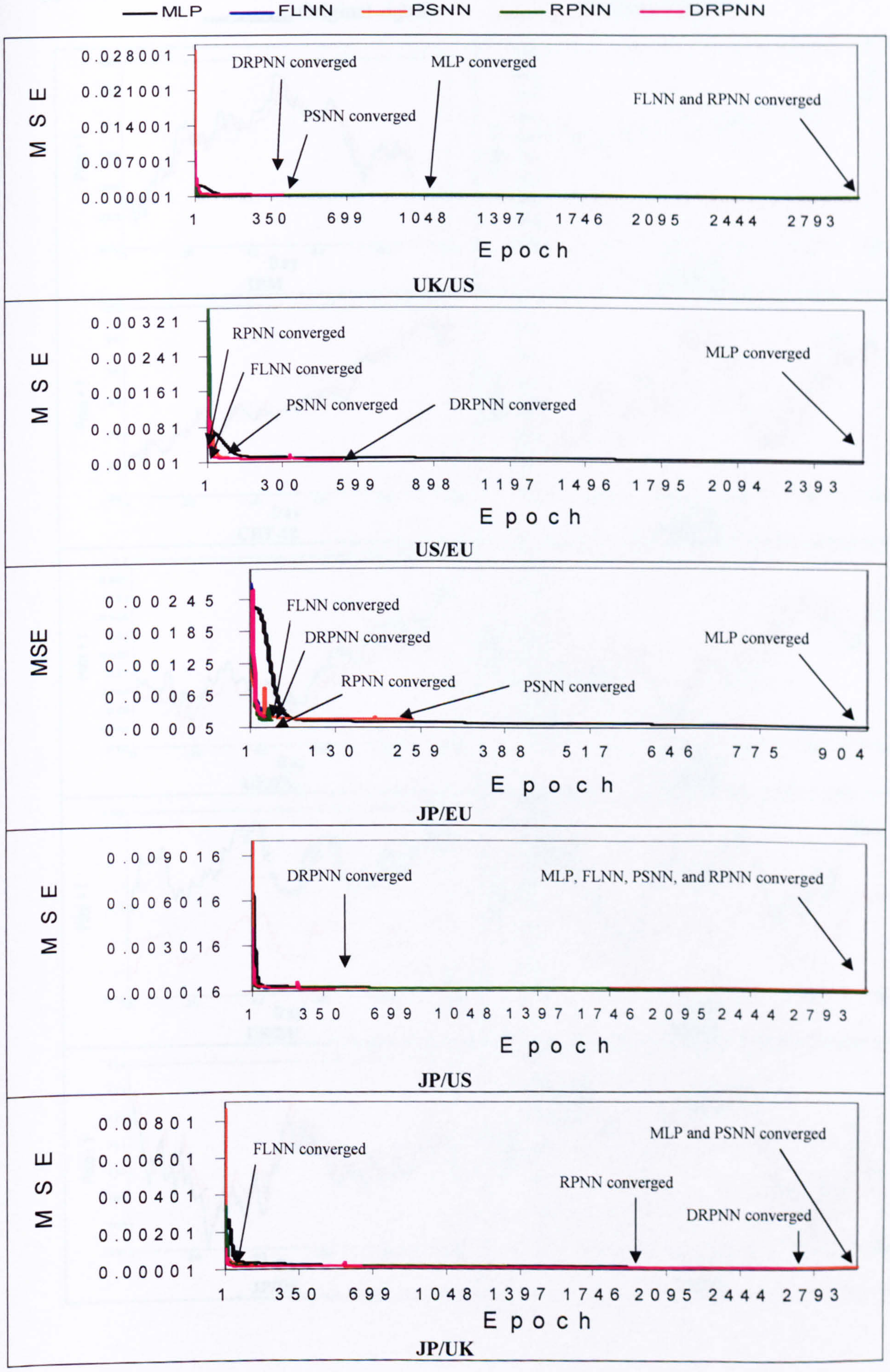
Appendix 7: Histograms of the signals error on stationary data for the prediction of five steps ahead using RPNNs



Appendix 8: Learning curves for the prediction of one step ahead using non-stationary signals

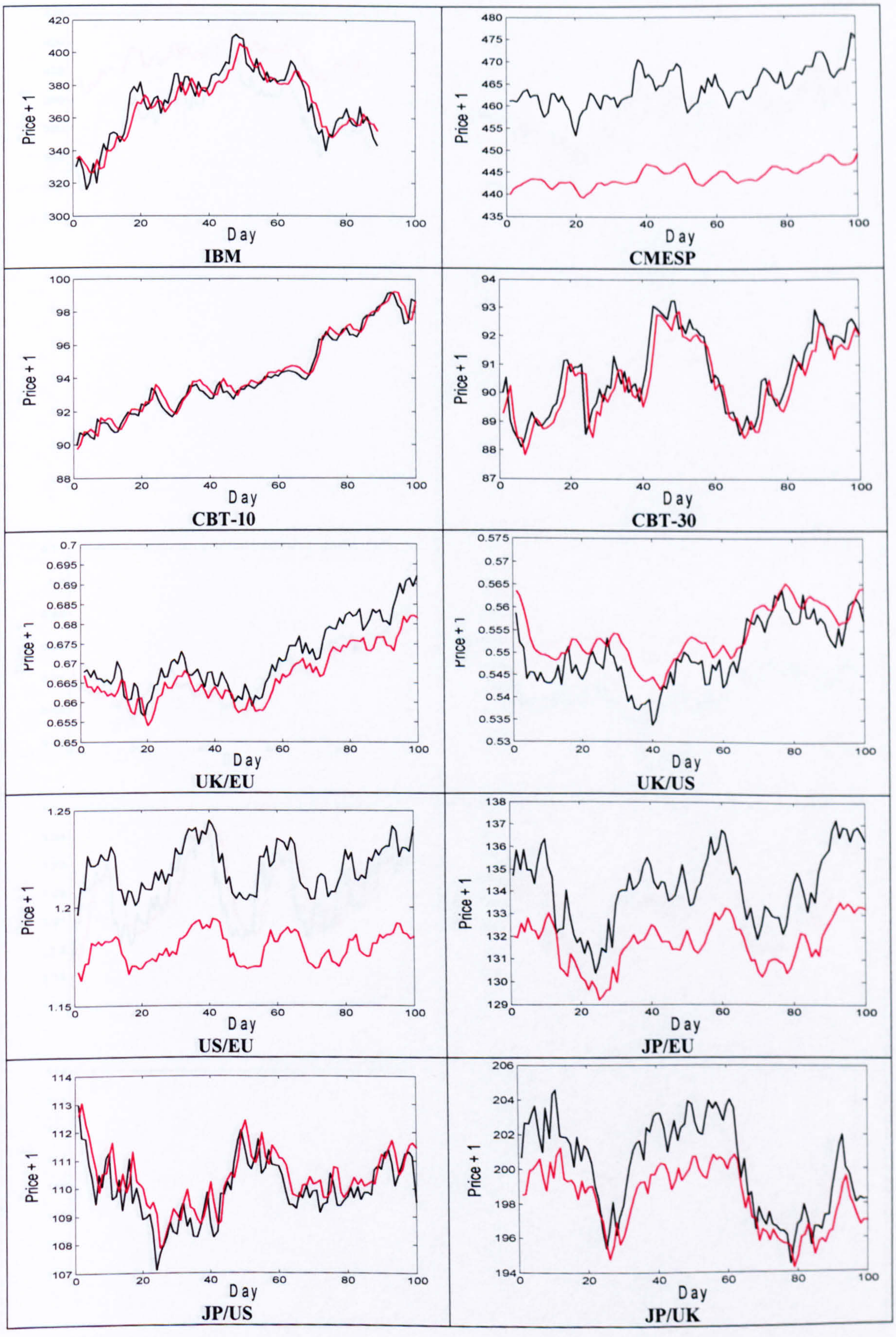


Appendix 8: Learning curves for the prediction of one step ahead using non-stationary signals



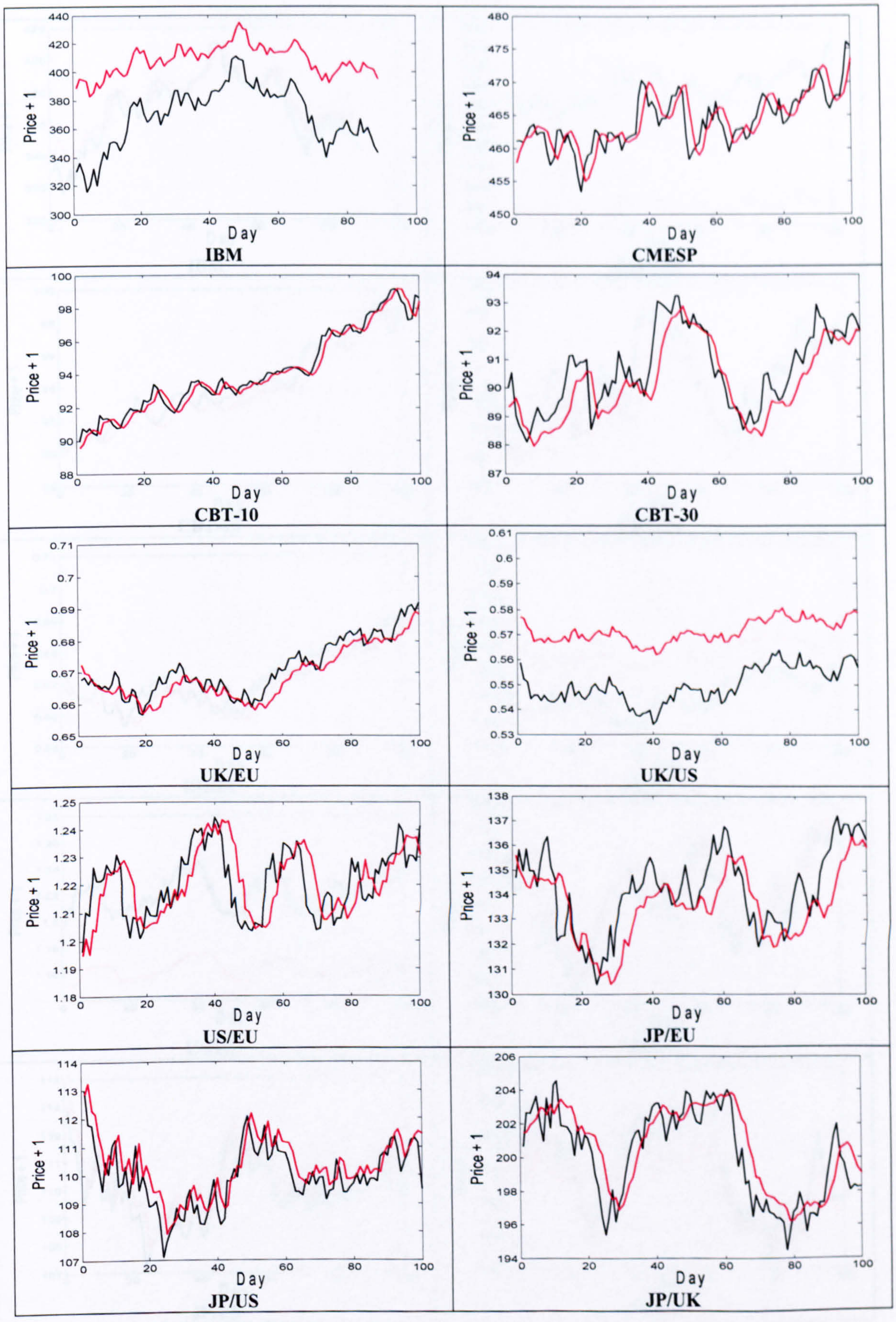
Appendix 9: Best forecasts on non-stationary data for the prediction of one step ahead using MLPs

Original signal, Predicted signal



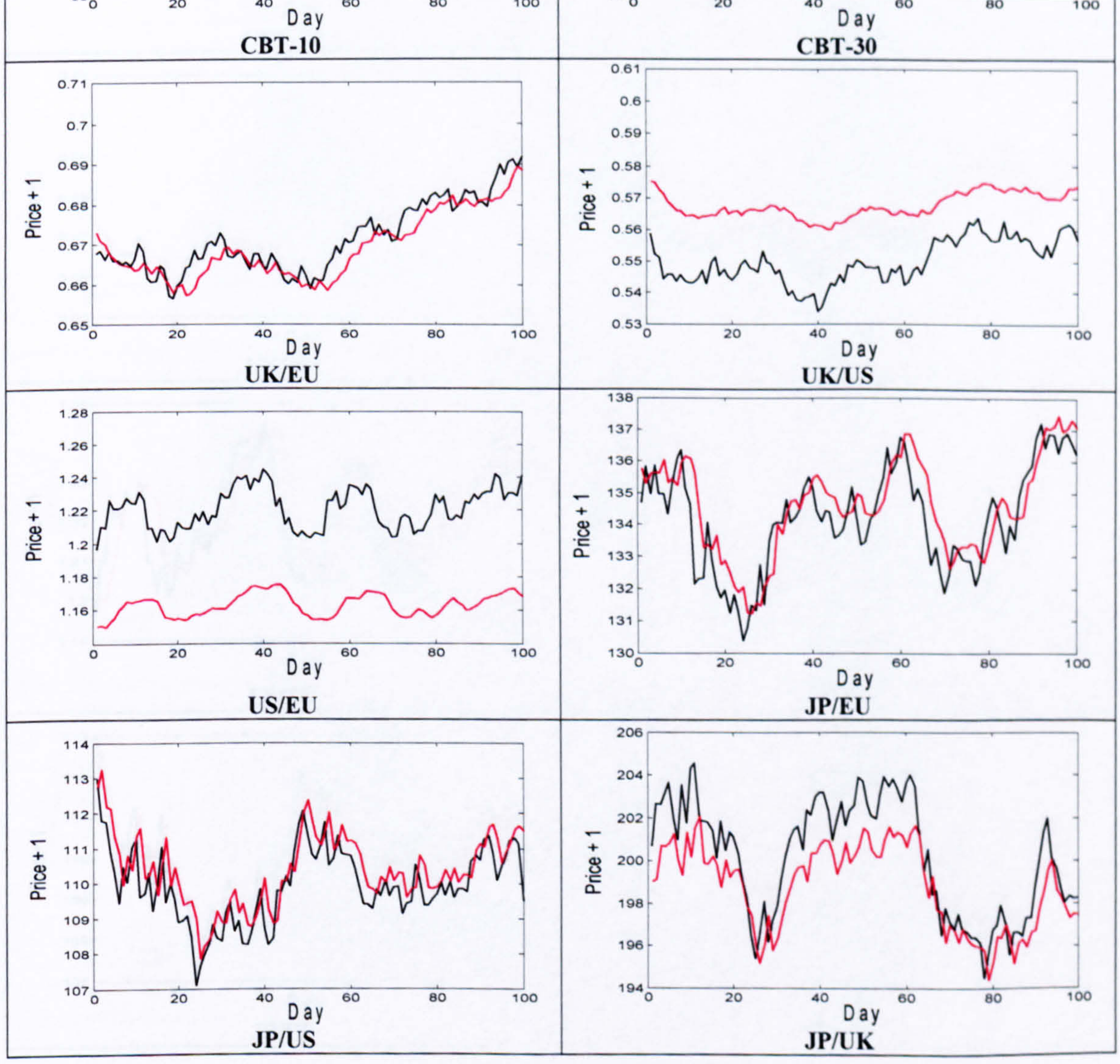
Appendix 9: Best forecasts on non-stationary data for the prediction of one step ahead using FLNNs

Original signal, Predicted signal



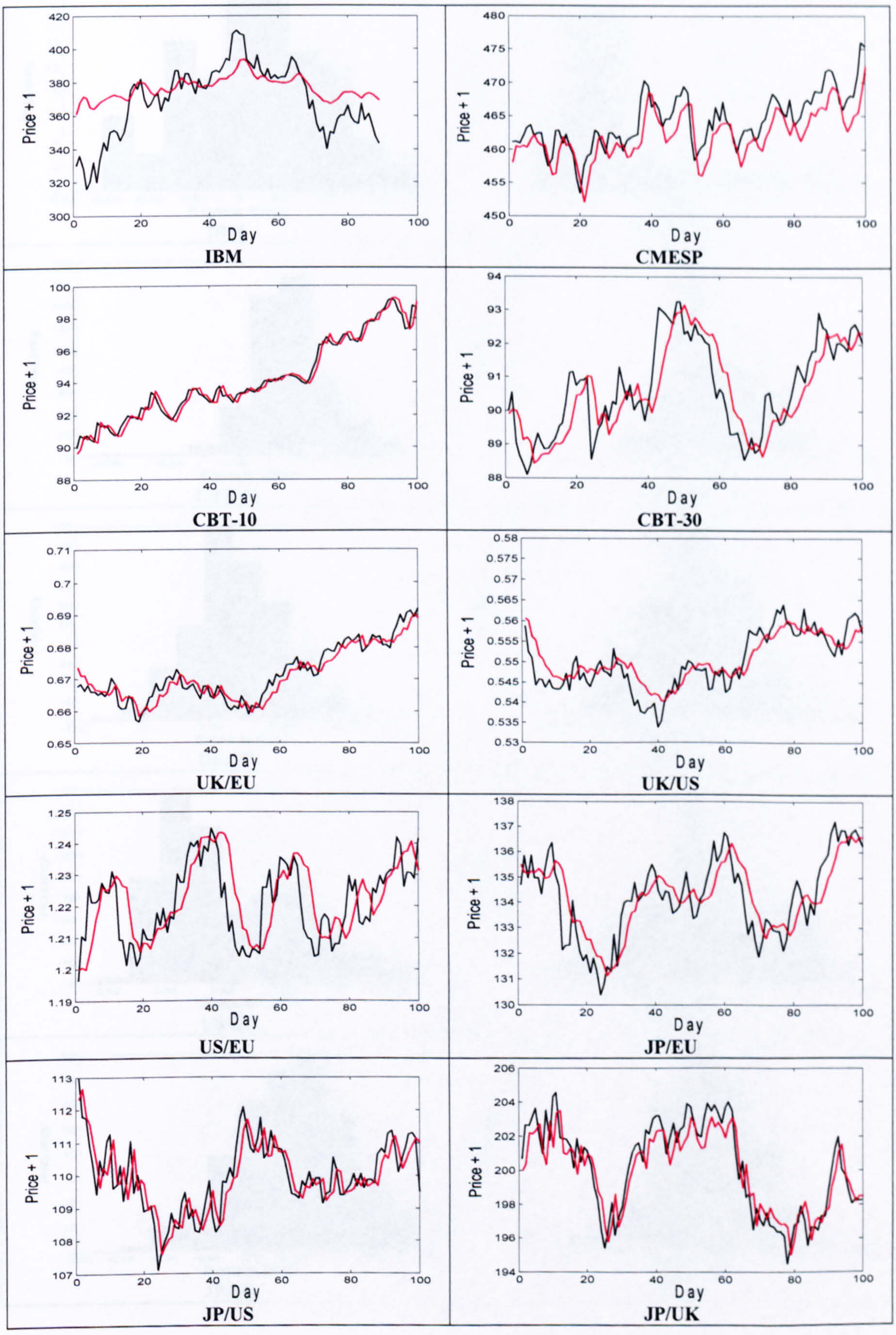
Appendix 9: Best forecasts on non-stationary data for the prediction of one step ahead using PSNNs

Original signal, Predicted signal

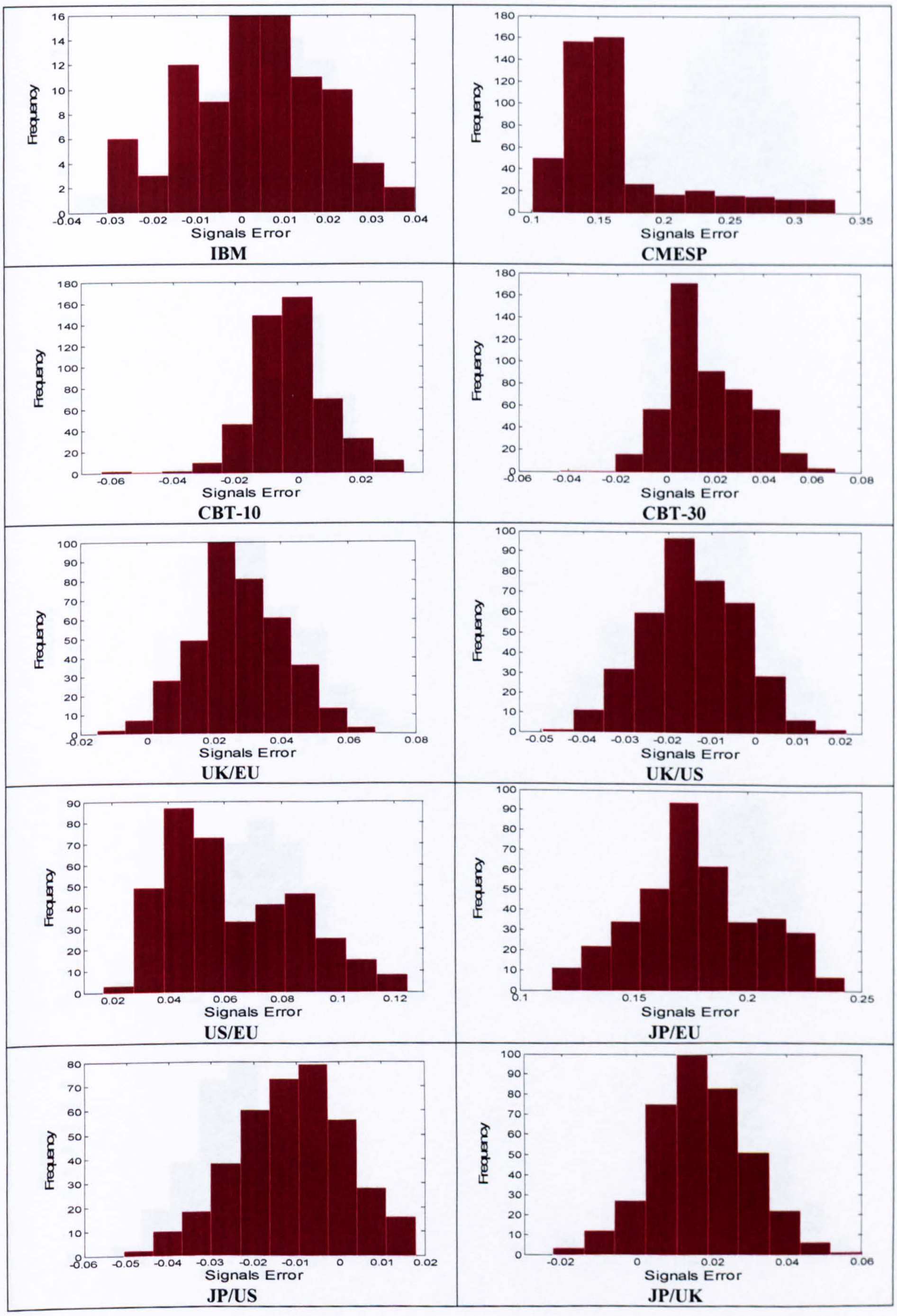


Appendix 9: Best forecasts on non-stationary data for the prediction of one step ahead using RPNNs

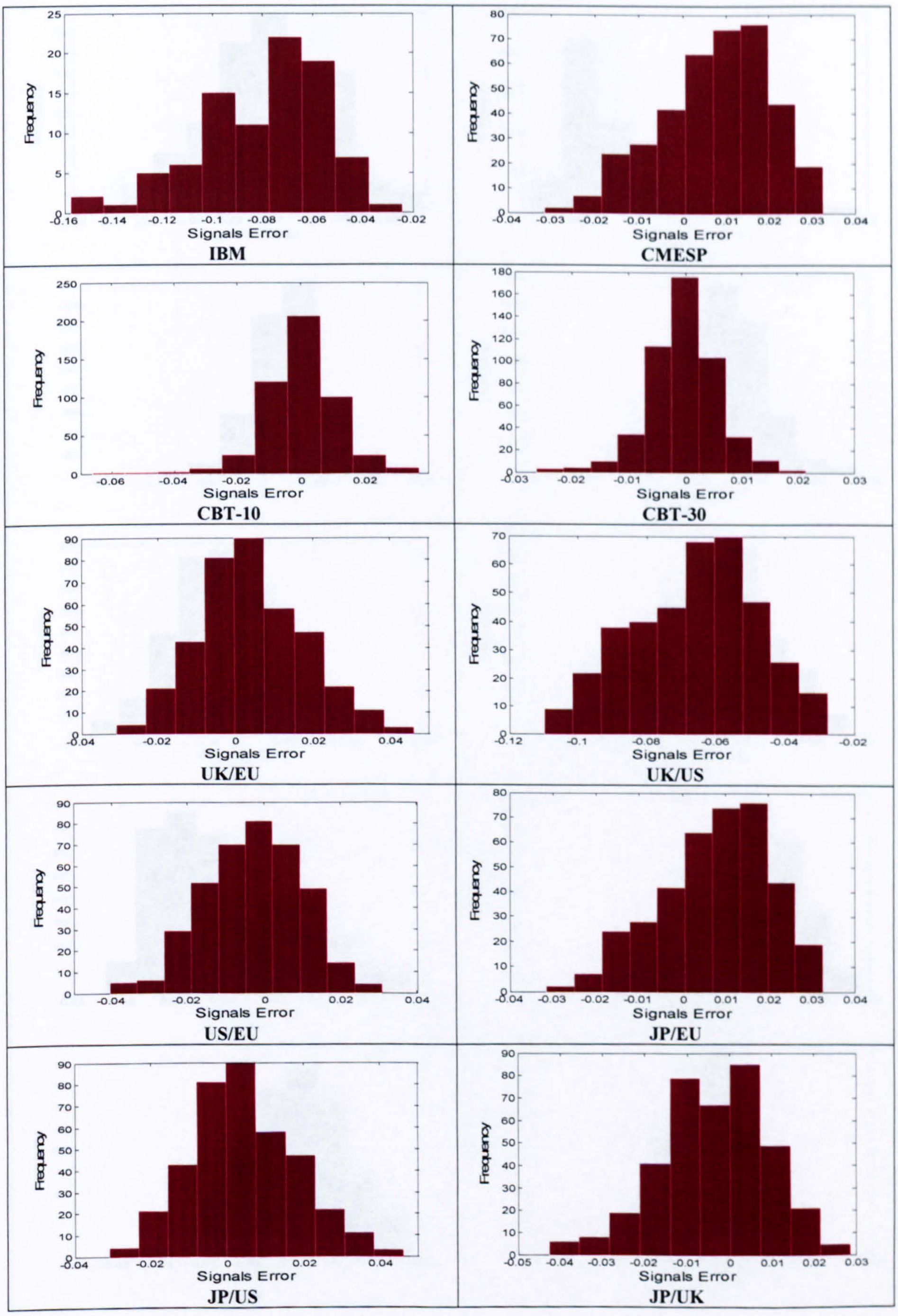
Original signal, Predicted signal



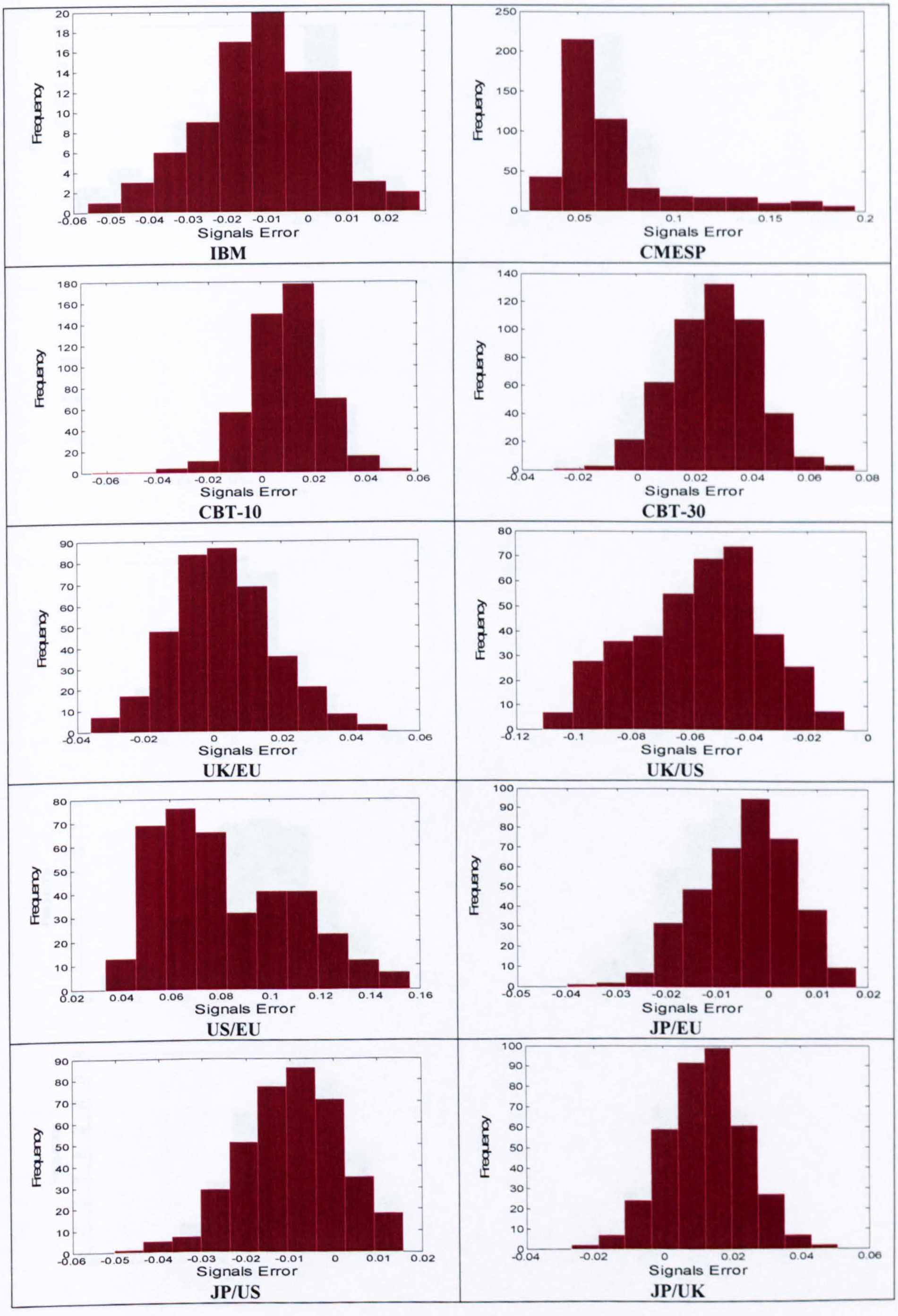
Appendix 10: Histograms of the signals error on non-stationary data for the prediction of one step ahead using MLPs



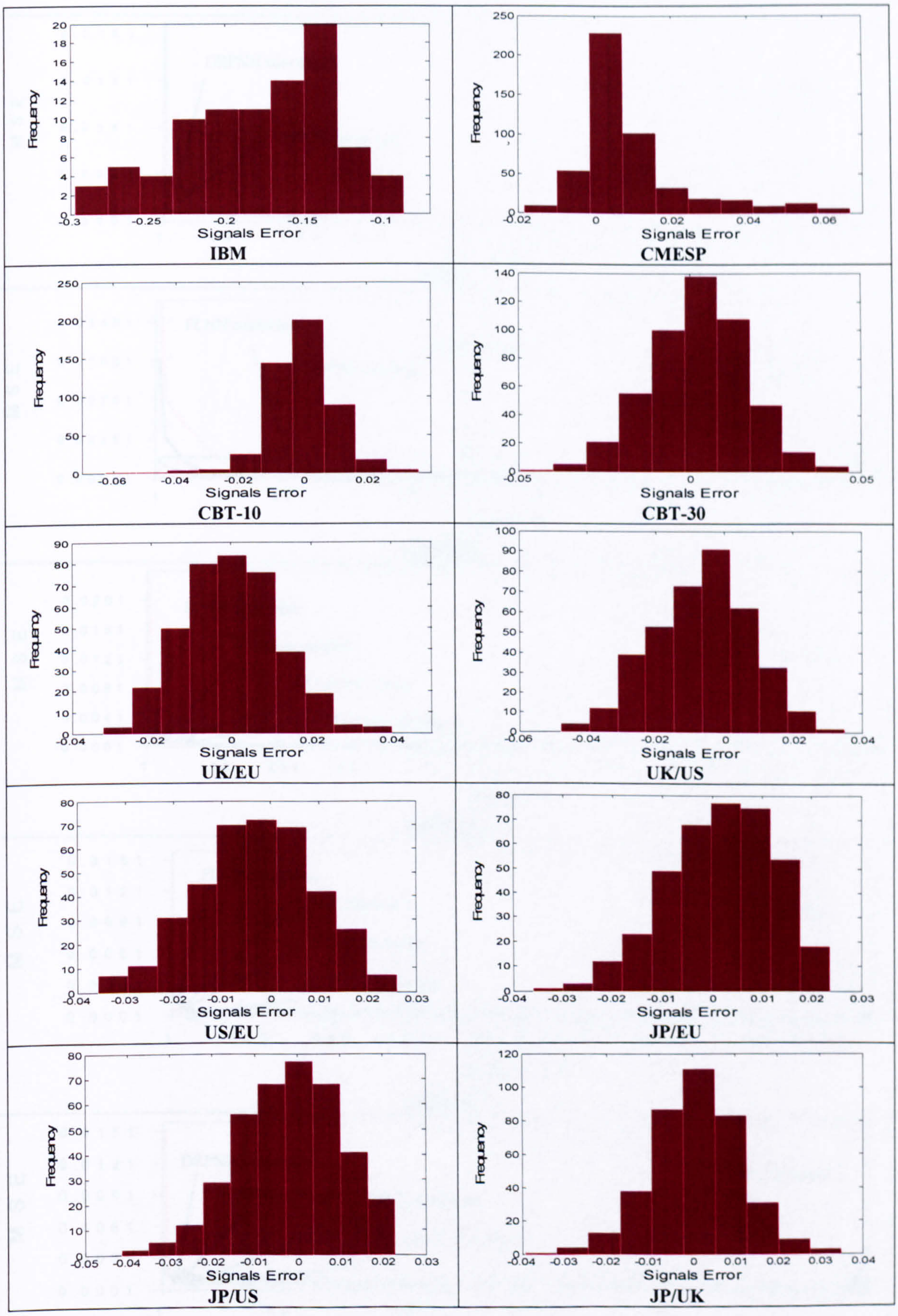
Appendix 10: Histograms of the signals error on non-stationary data for the prediction of one step ahead using FLNNs



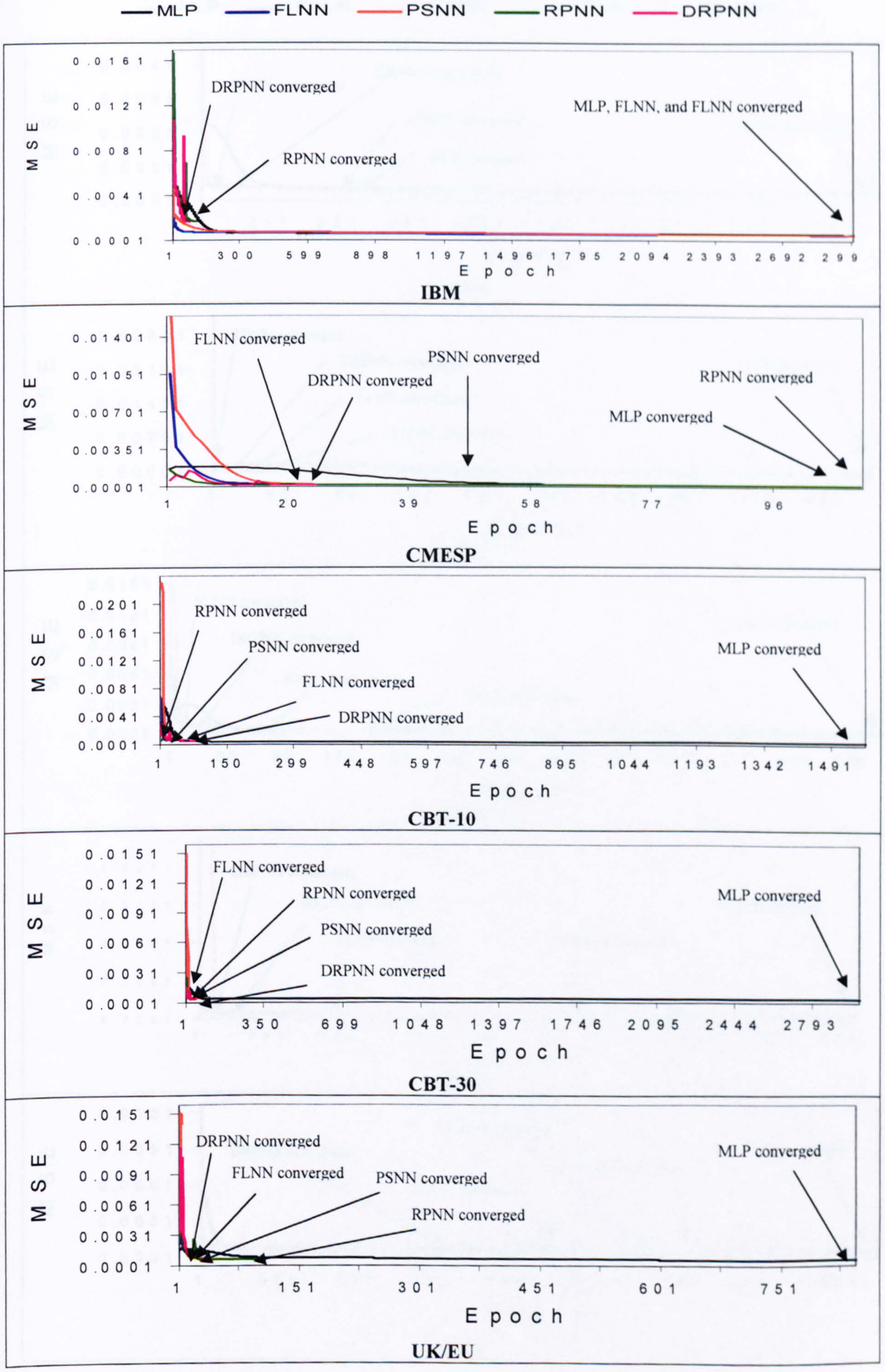
Appendix 10: Histograms of the signals error on non-stationary data for the prediction of one step ahead using PSNNs



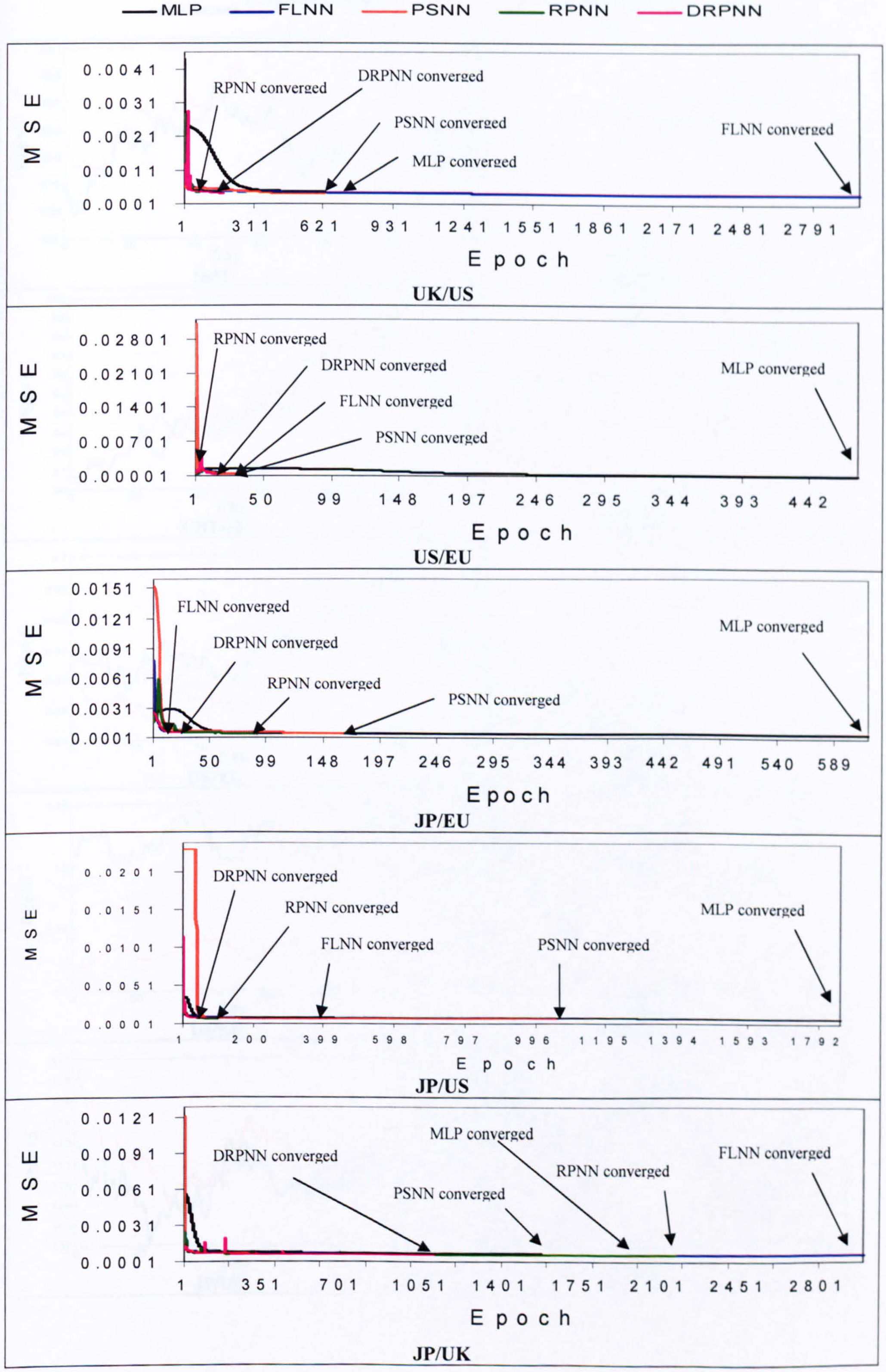
Appendix 10: Histograms of the signals error on non-stationary data for the prediction of one step ahead using RPNNs



Appendix 11: Learning curves for the prediction of five steps ahead using non-stationary signals

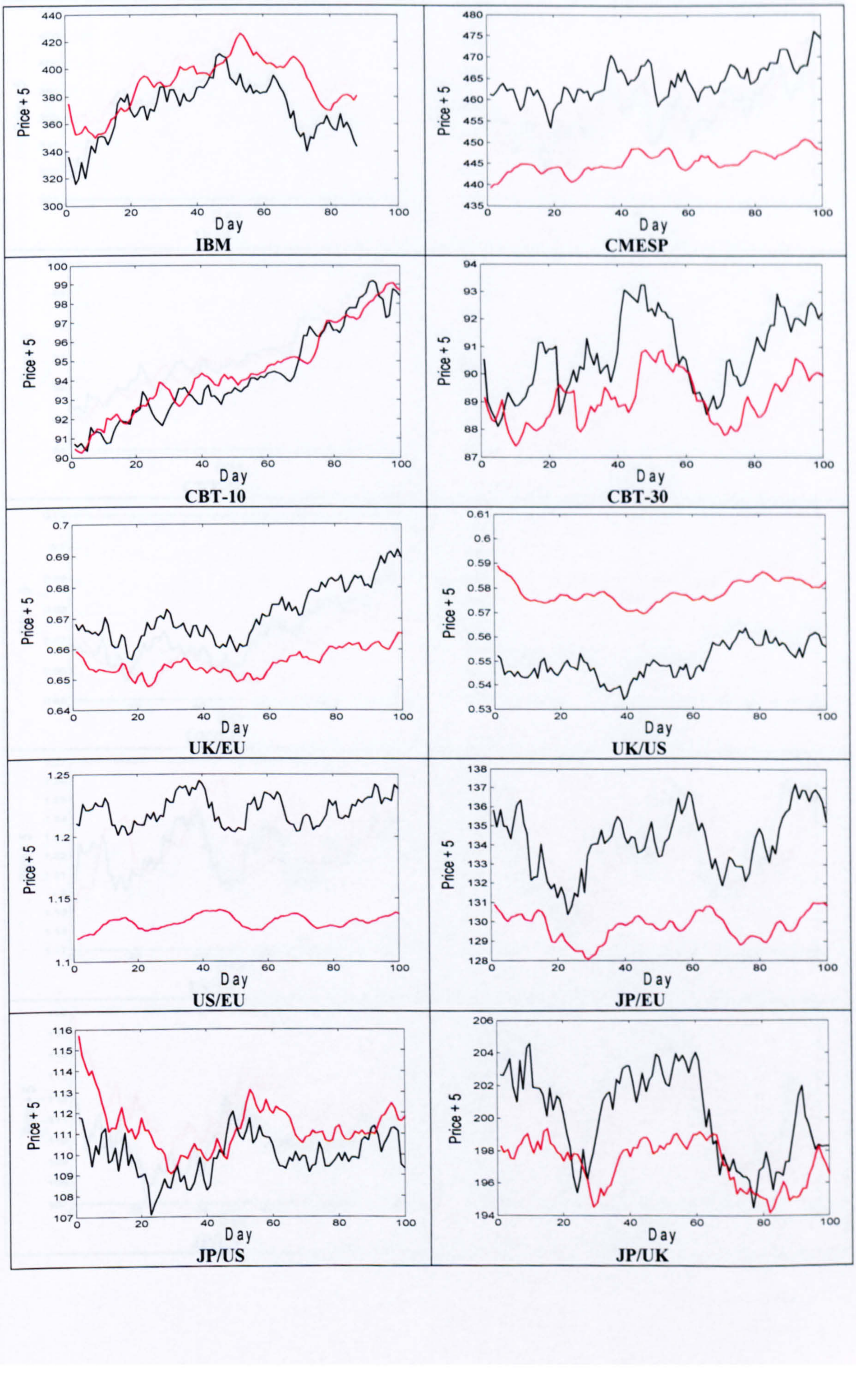


Appendix 11: Learning curves for the prediction of five steps ahead using non-stationary signals



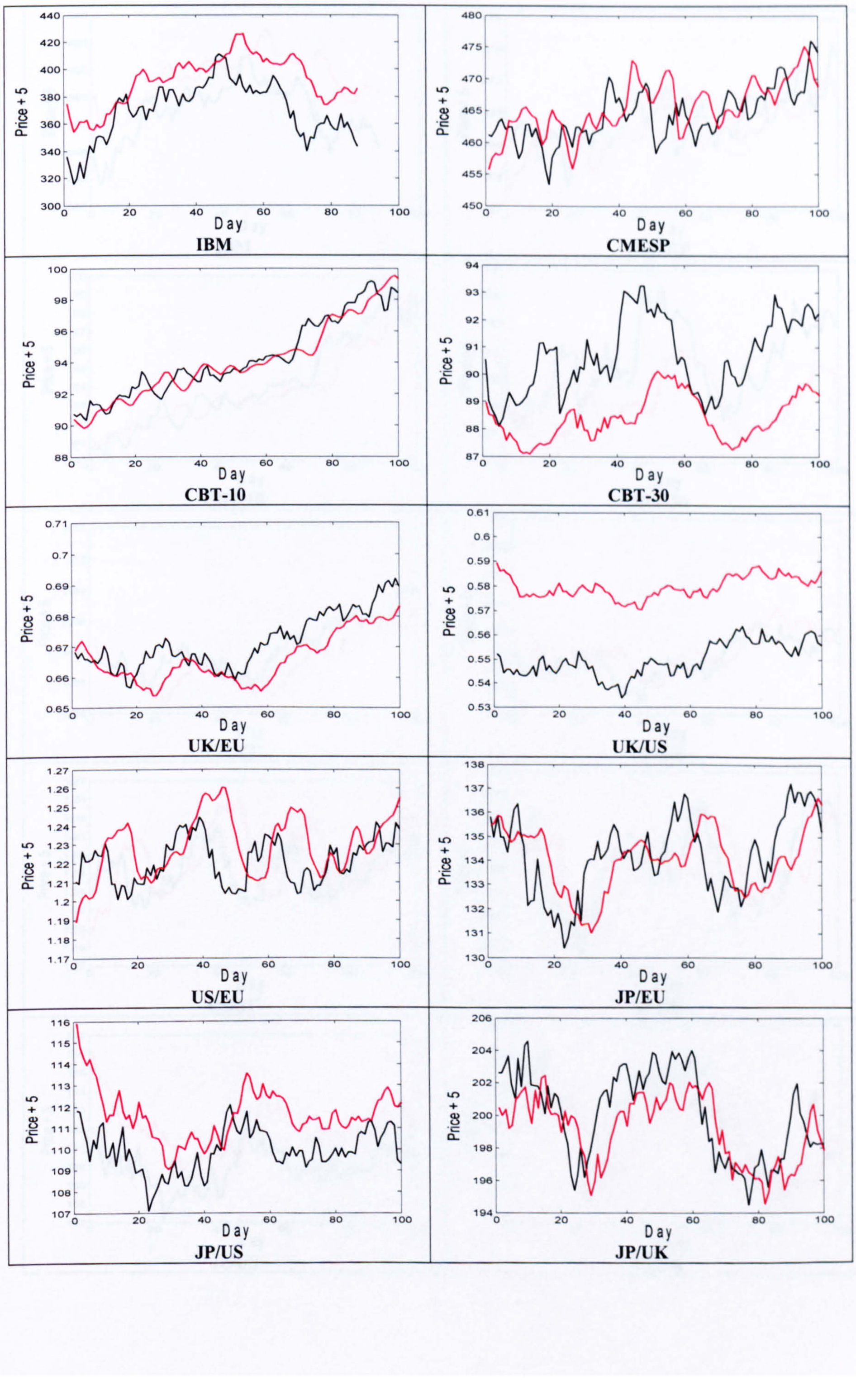
Appendix 12: Best forecasts on non-stationary data for the prediction of five steps ahead using MLPs

Original signal, Predicted signal



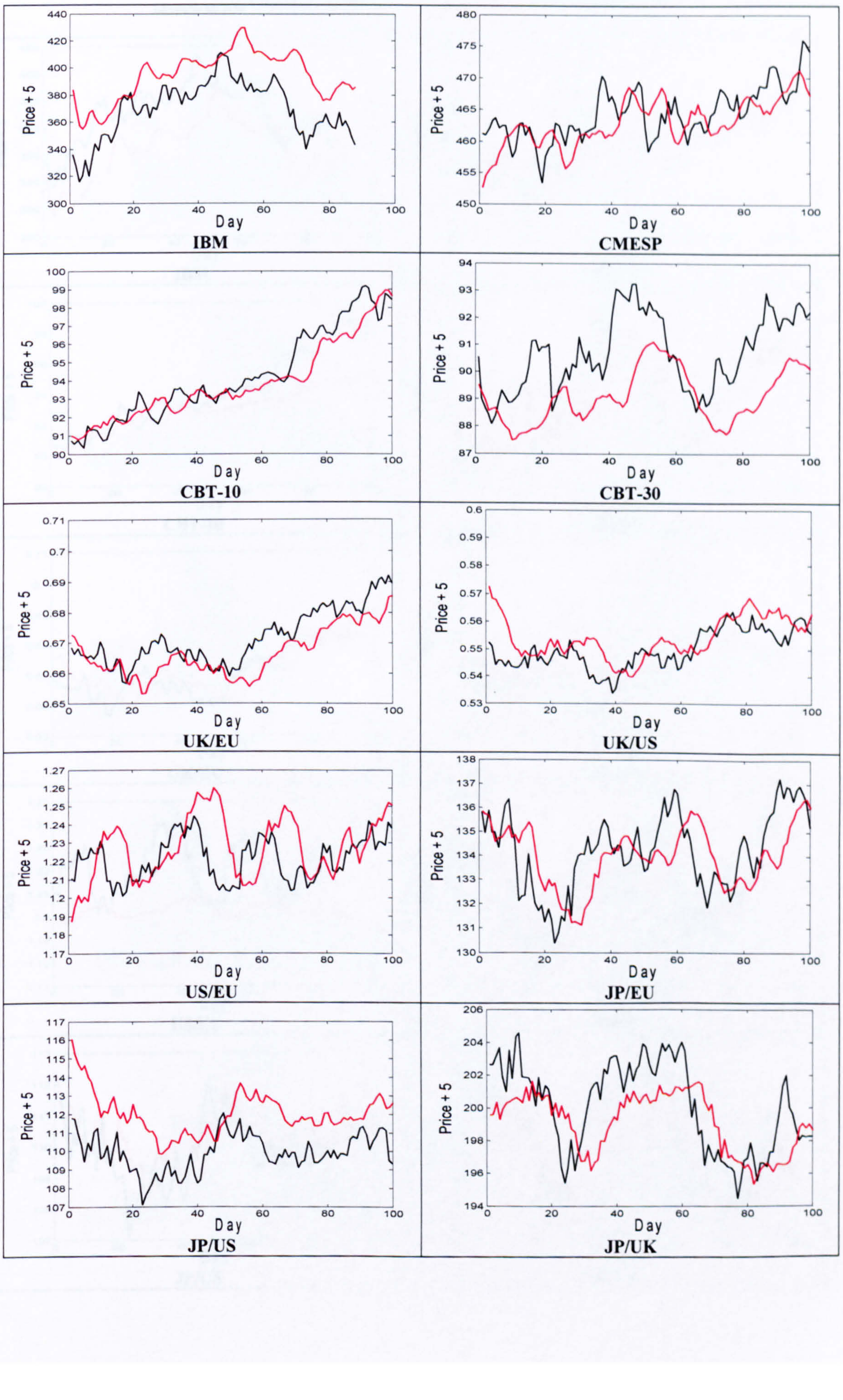
Appendix 12: Best forecasts on non-stationary data for the prediction of five steps ahead using FLNNs

Original signal, Predicted signal



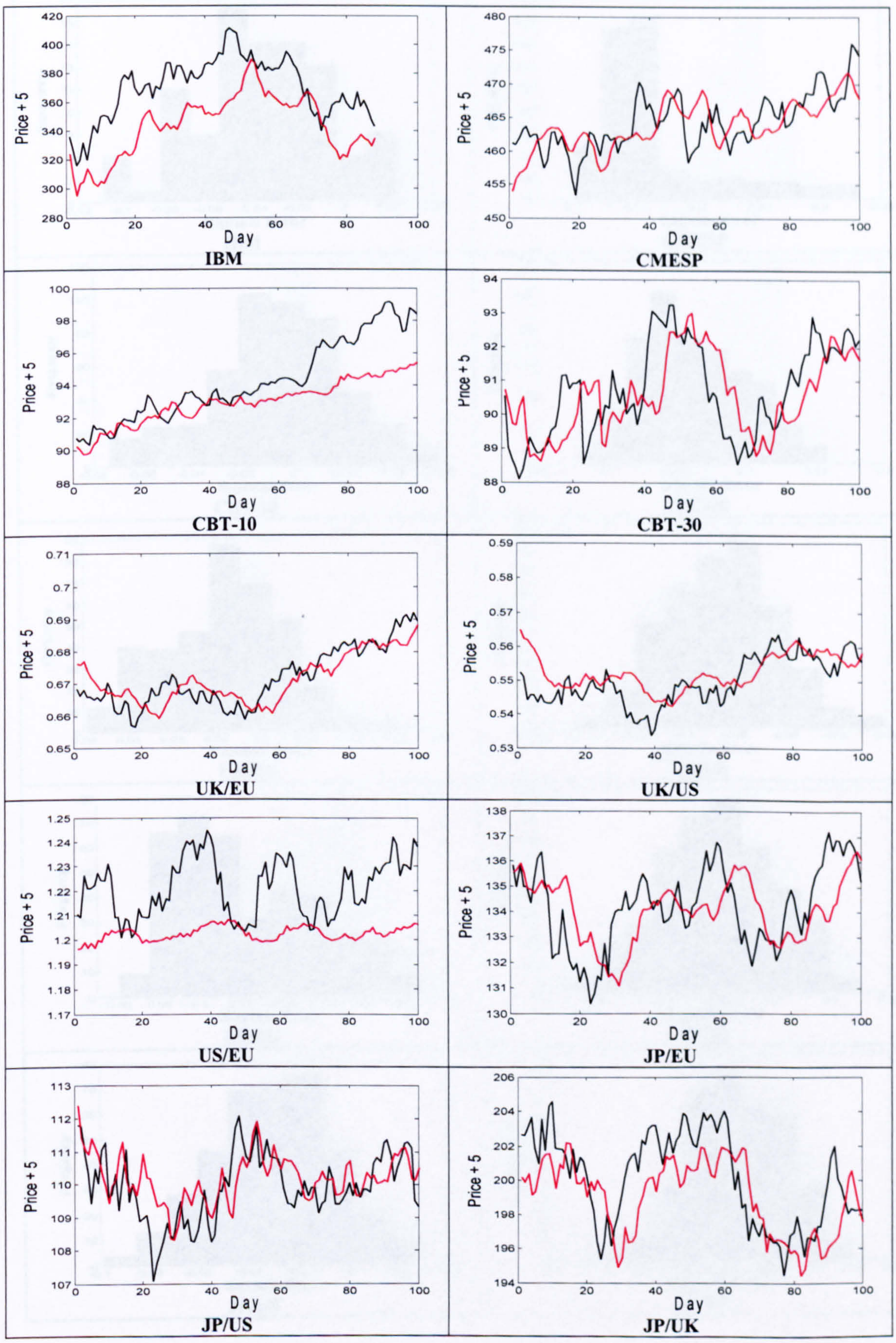
Appendix 12: Best forecasts on non-stationary data for the prediction of five steps ahead using PSNNs

Original signal, Predicted signal

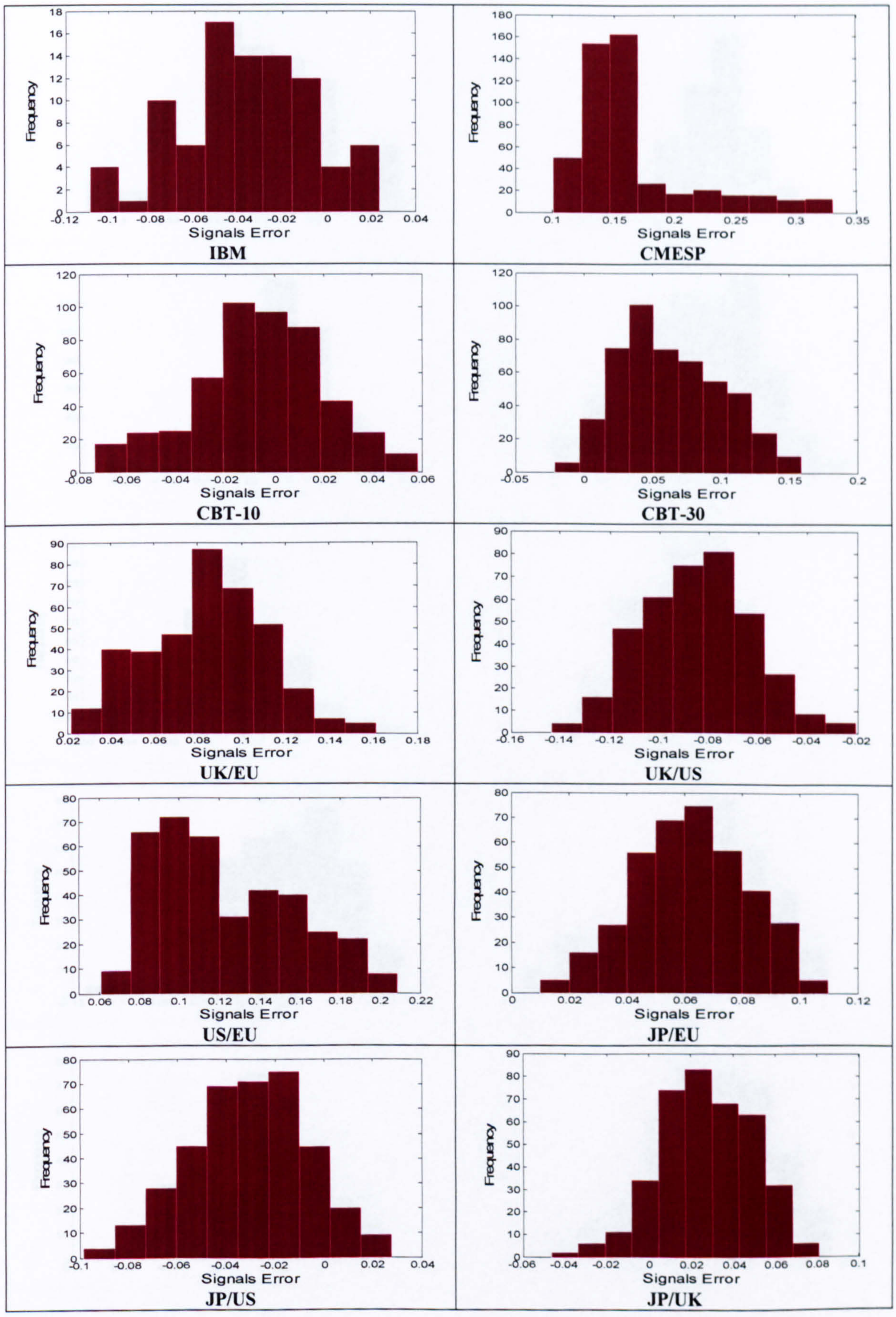


Appendix 12: Best forecasts on non-stationary data for the prediction of five steps ahead using RPNNs

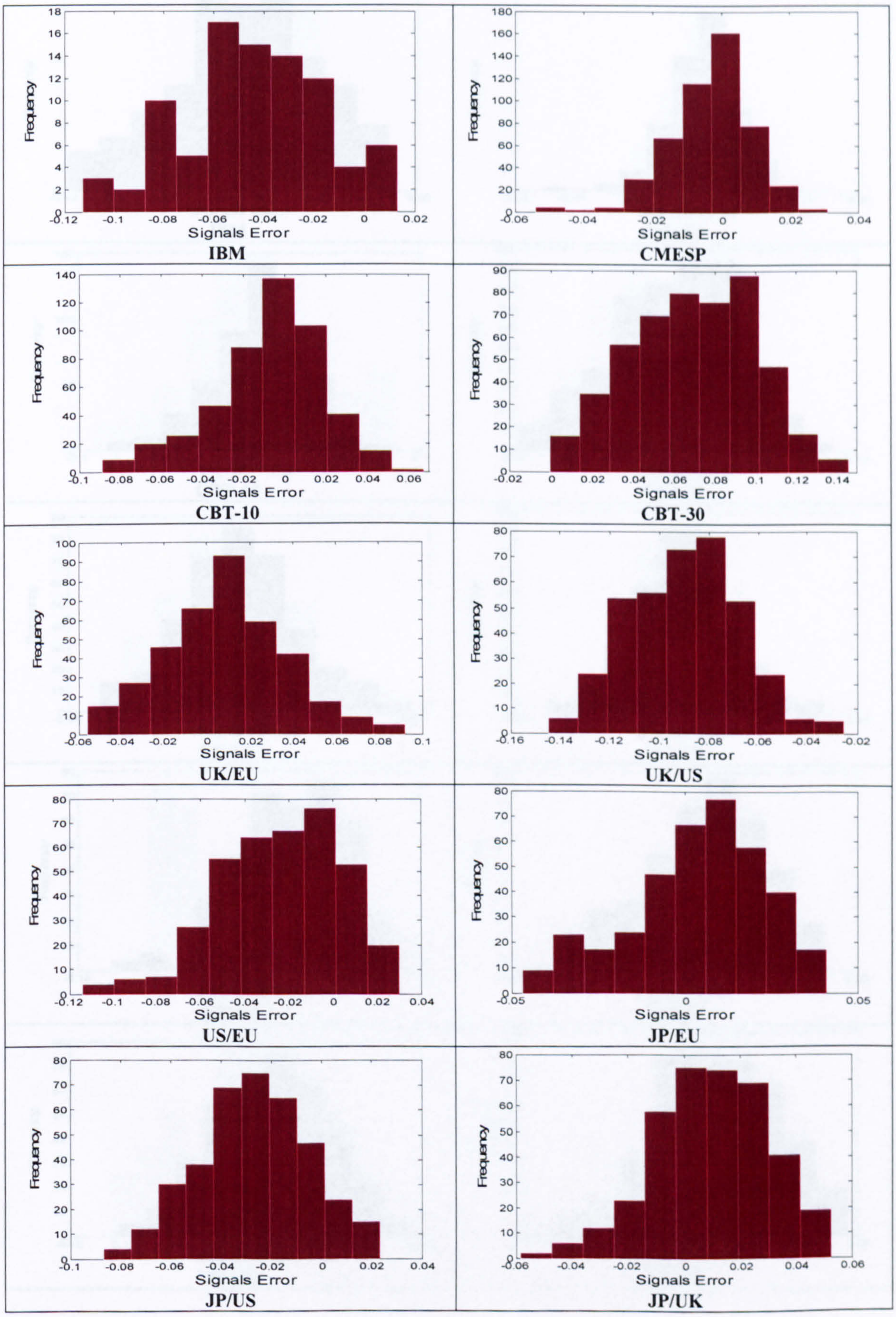
Original signal, Predicted signal



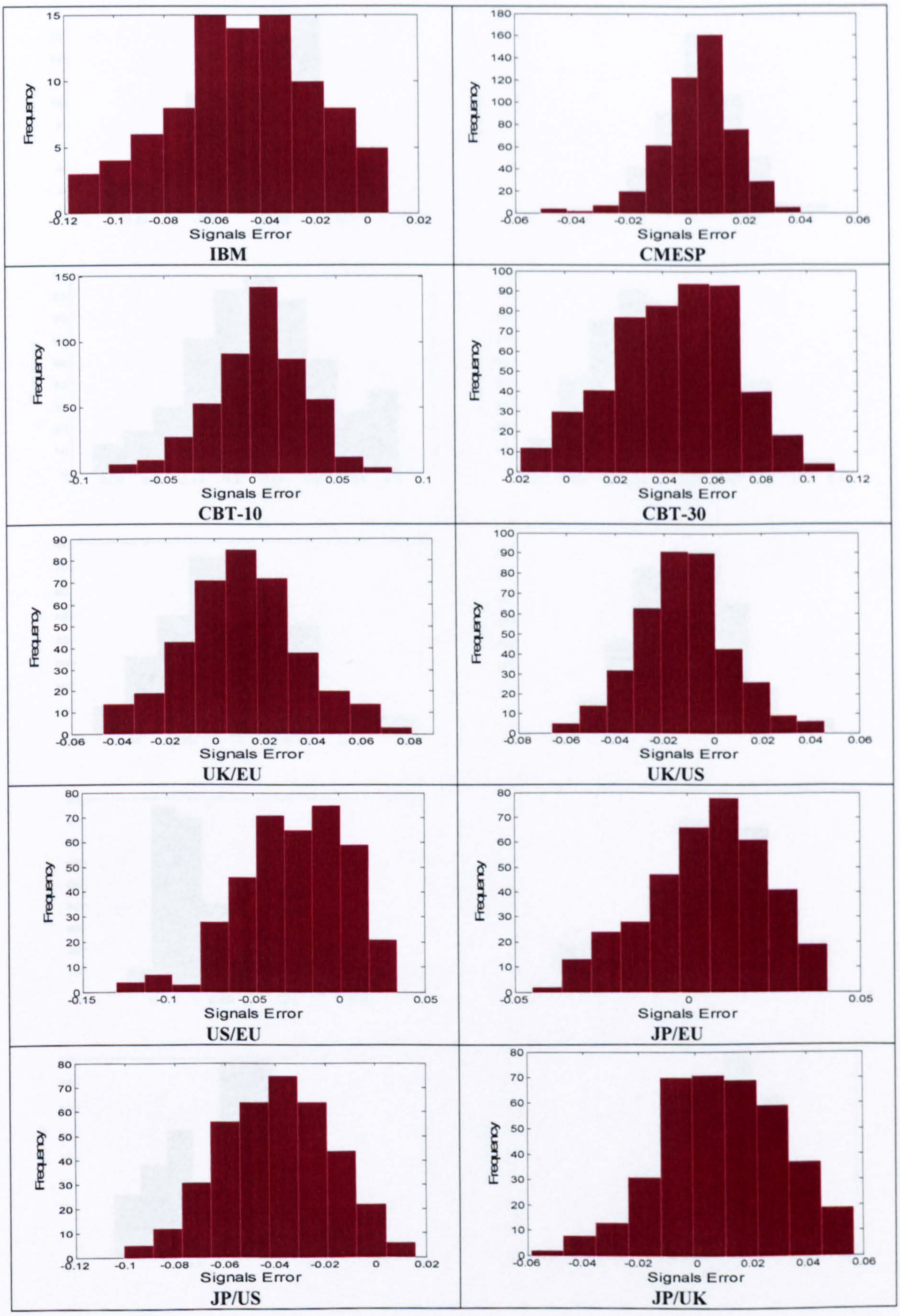
Appendix 13: Histograms of the signals error on non-stationary data for the prediction of five steps ahead using MLPs



Appendix 13: Histograms of the signals error on non-stationary data
for the prediction of five steps ahead using FLNNs



Appendix 13: Histograms of the signals error on non-stationary data for the prediction of five steps ahead using PSNNs



Appendix 13: Histograms of the signals error on non-stationary data for the prediction of five steps ahead using RPNNs