



## **Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin**

### **Copyright statement**

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

### **Liability statement**

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

### **Access Agreement**

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

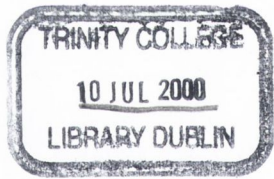




NEURAL NETWORK ENSEMBLES FOR FINANCIAL  
TIME-SERIES PREDICTION AND RISK MANAGEMENT

A THESIS SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE  
UNIVERSITY OF DUBLIN, TRINITY COLLEGE  
DEPARTMENT OF COMPUTER SCIENCE

John G. Carney  
May 2000

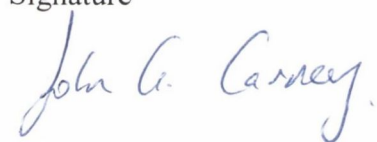


THESIS.  
5791

## **Permission to lend and/or copy**

I agree that the library in Trinity College Dublin may lend or copy the thesis “Neural Network Ensembles for Time-Series Prediction and Risk Management”.

Signature

A handwritten signature in blue ink that reads "John G. Carney". The signature is written in a cursive style with a large initial 'J' and a long, sweeping underline.

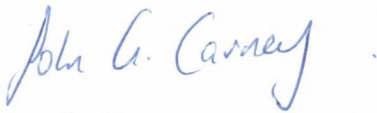
John G. Carney

June 2000

# Declaration

I declare that the work in this thesis has not been submitted for a degree at any other university, and that the work is entirely my own.

Signature

A handwritten signature in blue ink that reads "John G. Carney". The signature is written in a cursive style with a period at the end.

John G. Carney

May 2000.

# Acknowledgements

This thesis is dedicated to those who supported and encouraged me over the last 3 years. Dr. Pádraig Cunningham, my supervisor, deserves special thanks – his knowledge and vision guided the thesis and made it a success. Most importantly, when things were not going well he stood by and believed in me and the work – thanks Pádraig, I'll always appreciate that.

My dearest Alyson also deserves special thanks – sometimes I believe she should get a Ph.D. for her patience and support over the last 3 years. Thanks for putting up with so many endless days of work and stress – I would not have made it but for you.

I would also like to thank my wonderful parents who have always believed in the value of knowledge and education. I do recognise the sacrifices you have made to put me through so many years of education. Your love and support is forever appreciated.



# Abstract

Neural Network Ensembles for Financial Time-Series Prediction and Risk Management

John G. Carney

Supervisor: Dr. Pádraig Cunningham

Recently, neural networks have become popular tools for modelling financial markets. Much of this popularity can be attributed to the fact that neural networks are universal approximators i.e. they can (in theory at least) approximate any complex non-linear function to arbitrary accuracy. Given the complexity of modern financial markets, and the non-linearity that is widely accepted as driving the relationships between related financial variables, neural networks are potentially very powerful. This was recognised by financial market practitioners and researchers very early on. However, when systems were developed and tested, performance was typically poor. This is because non-parametric universal approximators such as neural networks can have serious limitations, especially when applied to model noisy, real-world systems such as financial markets.

One of the most serious is high-variance or instability i.e. small changes in training set and/or parameter selection can cause significant changes in generalisation (prediction) performance. Another problem (closely related to instability) is the tendency of neural networks to over-fit, essentially “memorize” their training sets, which also causes poor generalisation performance. The final, but probably most serious limitation of neural networks in the context of financial modelling, is the absence of

model transparency - neural networks are “black-box” estimators. Given the regulatory pressures today on financial institutions and traders to manage and limit their exposure to risk, such black-box models are just not good enough – the trader cannot determine what factors drive the model or more importantly how much confidence he can have in specific predictions.

In this thesis we attempt to (at least partly) solve these problems. To address the stability and over-fitting issues we focus on ensemble techniques and argue that bagging, an established statistical ensemble technique based on bootstrapping is particularly suitable for neural networks applied to financial time-series prediction. An important feature of our work on bagging is that we recognise the importance of diversity amongst individual members in a bagged ensemble. This motivates the development of a new early-stopping technique (the NeuralBAG algorithm) that tunes diversity by varying the fit of the individual networks in an ensemble. Significant advantages of NeuralBAG over other ensemble techniques (in the context of financial time-series prediction) include its robustness to noise and the availability of model variance (confidence) estimates from the bootstrapping process.

To address the issues relating to the absence of transparency in neural network models, we propose a new technique for generating prediction intervals i.e. estimates of bounds on the possible error of our prediction of the targets. A unique feature of this technique is the way in which it incorporates uncertainty caused by model variance (estimated using bootstrapping as outlined above) and noise variance (estimated using an established econometric technique). These prediction intervals essentially allow a trader to anticipate the quality of an ensemble prediction before the prediction horizon expires. This enables him to manage risk more effectively and essentially circumvents the problems relating to the absence of transparency – the trader is provided with a quantitative measure of how much faith he can put in a prediction.

## Associated publications

J.G. Carney and P. Cunningham. The NeuralBAG algorithm: Optimising generalisation performance in bagged neural networks. In M. Verleysen, editor, *Proceedings of the 7th European Symposium on Artificial Neural Networks*, pages 35-40, Bruges, Belgium, 1999.

J.G. Carney, P. Cunningham and U. Bhagwan. Confidence and prediction intervals for neural network ensembles. In *Proceedings of the International Joint Conference on Neural Networks 1999*, paper 2090, Washington D.C., 1999.

J.G. Carney and P. Cunningham. Tuning diversity in bagged ensembles. Technical report TCD-CS-1999-44, Department of Computer Science, University of Dublin, Trinity College, 1999. To appear in *International Journal of Neural Systems*.

P. Cunningham, J.G. Carney and S. Jacob. Stability problems with neural networks and the ensemble solution. Technical report TCD-CS-1999-52, Department of Computer Science, University of Dublin, Trinity College, 1999. To appear in *Journal of Artificial Intelligence in Medicine*.

S. Jacob, P. Cunningham, J.G. Carney and R.F. Harrison. Prediction of assisted reproduction outcome using artificial neural networks. Submitted.

P. Cunningham and J.G. Carney. Diversity versus quality in ensembles for feature selection. Technical report TCD-CS-2000-02, Department of Computer Science,

University of Dublin, Trinity College, 2000. To appear in *Proceedings of the 11th European Conference on Machine Learning*.



# Contents

<b>Declaration</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>Associated publications</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Predictability in financial markets . . . . .	3
1.3 Financial market prediction techniques . . . . .	5
1.3.1 Fundamental analysis . . . . .	5
1.3.2 Charting . . . . .	7
1.3.3 Econometrics . . . . .	7
1.3.4 “Exotic” techniques . . . . .	8
1.4 Neural networks for financial time-series prediction . . . . .	10
1.5 Neural network ensemble learning . . . . .	12
1.6 Finding predictability . . . . .	14
1.7 The data . . . . .	16
1.7.1 Stocks . . . . .	16
1.7.2 Stock-market indices . . . . .	17
1.7.3 Foreign exchange . . . . .	18
1.8 Thesis contributions . . . . .	18



1.9	Summary . . . . .	19
<b>2</b>	<b>Bagging and bootstrapping</b>	<b>21</b>
2.1	Introduction . . . . .	21
2.2	Bootstrapping . . . . .	21
2.3	Bagging . . . . .	25
2.4	Diversity and ensemble generalization performance . . . . .	29
2.5	Summary . . . . .	30
<b>3</b>	<b>Optimizing generalisation performance</b>	<b>31</b>
3.1	Introduction . . . . .	31
3.2	Optimising generalisation performance using early-stopping . . . . .	32
3.3	Applying early-stopping to bagged ensembles . . . . .	35
3.4	The NeuralBAG algorithm . . . . .	36
3.5	Evaluation . . . . .	39
3.5.1	Experiment 1: Are NeuralBAG estimates of generalisation error better than local estimates? . . . . .	39
3.5.2	Experiment 2: Does NeuralBAG over-fit or under-fit? . . . . .	41
3.5.3	Experiment 3: How does ambiguity and generalisation error evolve during training? . . . . .	41
3.5.4	Experiment 4: How many networks should the ensembles have? . . . . .	42
3.5.5	Discussion . . . . .	43
3.6	Relation to other work . . . . .	46
3.7	Summary . . . . .	50
<b>4</b>	<b>Predicting uncertainty</b>	<b>51</b>
4.1	Introduction . . . . .	51
4.2	Uncertainty in regression . . . . .	52
4.3	Confidence intervals . . . . .	53
4.3.1	Theory . . . . .	53
4.3.2	Illustration . . . . .	56
4.4	Prediction intervals . . . . .	57

4.4.1	Volatility . . . . .	58
4.4.2	Estimating the decay factor . . . . .	60
4.4.3	Combining volatility with model variance . . . . .	61
4.5	Summary . . . . .	62
<b>5</b>	<b>Evaluation</b>	<b>63</b>
5.1	Introduction . . . . .	63
5.2	Evaluation metrics . . . . .	64
5.2.1	Root mean squared error . . . . .	64
5.2.2	Correlation coefficient . . . . .	65
5.2.3	Information coefficient . . . . .	65
5.2.4	Directional change . . . . .	66
5.2.5	Interval non-coverage . . . . .	66
5.3	Experimental set-up . . . . .	66
5.3.1	Stocks . . . . .	68
5.3.2	Stock-market indices . . . . .	70
5.3.3	Foreign exchange . . . . .	71
5.4	Analysis of results . . . . .	72
5.4.1	Stocks . . . . .	73
5.4.2	Foreign exchange and stock-market indices . . . . .	79
5.4.3	Prediction interval performance . . . . .	80
5.5	Summary . . . . .	85
<b>6</b>	<b>Conclusion</b>	<b>87</b>
6.1	Introduction . . . . .	87
6.2	Thesis contributions . . . . .	87
6.2.1	Machine learning . . . . .	87
6.2.2	Time-series prediction . . . . .	88
6.2.3	Econometrics . . . . .	89
6.2.4	Finance . . . . .	89
6.3	Future work . . . . .	89
6.3.1	Feature selection . . . . .	90

6.3.2	Ensembles . . . . .	90
6.3.3	Prediction intervals . . . . .	90
6.4	Summary . . . . .	91
<b>A</b>	<b>NeuralBAG C-MPI code</b>	<b>92</b>
	<b>Bibliography</b>	<b>134</b>

# List of Tables

3.1	Experiment 1 results. Generalisation (test set) mean-squared error performance of ensembles trained using NeuralBAG (NBAG) compared to the benchmark technique adapted from (Breiman, 1996b) that uses local estimates of generalisation error (B-LOCAL) and the simple technique described in section 3.3 that also uses local estimates of generalisation error (S-LOCAL). Note that the USD/CHF, S&P500 and NYSE results have been adjusted for clarity and ease of comparison. For example, all of the S&P500 results have been multiplied by $10^{-2}$ .	41
3.2	Experiment 2 results. Average optimal number of epochs estimated for each ensemble and data-set in experiment 1. . . . .	42
4.1	Illustration of confidence interval performance expressed in terms of interval non-coverage i.e. number of times the actual target value was not covered by the interval (see section 5.2.5 for an exact description of this metric). Given that we used 1000 test points for each experiment we would expect non-coverage of 200 for an 80% interval for example.	57
4.2	Estimates for the decay factor $\lambda$ for each data-set used in this thesis. We use 11 years of observations for the stock-market experiments ( $H = 2780$ ) and 5 years of observations for the foreign exchange experiments ( $H = 1238$ ). We tested for 18 values of $\lambda$ (0.1, 0.15, ..., 0.9, 0.95) for each volatility horizon. . . . .	61
5.1	Coca-Cola 1-day, 5-days, 10-days and 20-days ahead prediction results.	75

5.2	General Electric Corporation 1-day, 5-days, 10-days and 20-days ahead prediction results. . . . .	76
5.3	International Business Machines 1-day, 5-days, 10-days and 20-days ahead prediction results. . . . .	77
5.4	Microsoft Corporation 1-day, 5-days, 10-days and 20-days ahead prediction results. . . . .	78
5.5	CHF/JPY 1-day, 5-days, 10-days and 20-days ahead prediction results.	81
5.6	USD/JPY 1-day, 5-days, 10-days and 20-days ahead prediction results.	82
5.7	S&P500 1-day, 5-days, 10-days and 20-days ahead prediction results.	83
5.8	NYSE 1-day, 5-days, 10-days and 20-days ahead prediction results. .	84
5.9	Average interval non-coverage across all of the SIMLIVE experiments for the 80%, 90%, 95% and 99% prediction intervals. . . . .	85



# List of Figures

- 2.1 50 rolls of a die. The possible roll outcomes 1, 2, 3, 4, 5, 6 occurred 7, 8, 8, 10, 9, 8 times respectively. Therefore the empirical distribution is  $(7/50, 8/50, 8/50, 10/50, 9/50, 9/50)$ . . . . . 23
- 2.2 A comparison of the real and bootstrap versions of a parameter estimation problem. In the real version of the problem we are given a finite set of  $n$  observations from some unknown probability distribution  $P$ . We use these observations to generate a single parameter estimate  $\hat{\theta}$ . We also use these observations to generate an empirical distribution  $\hat{P}$ . Note that in the diagram both the original observations and the empirical distribution are enclosed in the large arrow. This is to denote that observations from the empirical distribution are generated by just sampling at random with replacement from the original set of observations. Using this any number of bootstrap samples from the empirical distribution can be generated and therefore also any number of bootstrap parameter estimates  $\hat{\theta}^*$ . . . . . 24
- 2.3 A bagged neural network ensemble. Training sets are generated by sampling from the bootstrap empirical distribution  $\hat{P}$ . These training sets are used to train an ensemble of neural networks. The outputs of the networks in the ensemble are averaged to produce the bagged ensemble prediction. . . . . 26

3.1	Experiment 3 results, CHF/JPY data. Here we compare the ensemble generalisation performance to the ensemble ambiguity and the average individual network generalisation performance as training evolves for a single training and test set pair of the CHF/JPY data. . . . .	43
3.2	Experiment 3 results, S&P500 data. As above for figure 3.1 but with the S&P500 data. . . . .	44
3.3	Experiment 4 results, CHF/JPY data, part 1. Here we plot the average generalisation error across 10 experiments on the CHF/JPY data as a function of the number of networks in ensembles trained with NeuralBAG. . . . .	45
3.4	Experiment 4 results, S&P500 data, part 1. As above for figure 3.3 but with the S&P500 data. . . . .	46
3.5	Experiment 4 results, CHF/JPY data, part 2. Here we plot the average estimated optimal number of epochs across 10 experiments on the CHF/JPY data as a function of the number of networks in ensembles trained with NeuralBAG. . . . .	47
3.6	Experiment 4 results, S&P500 data, part 2. As above for figure 3.5 but with the S&P500 data. . . . .	48
4.1	Synthetic test data generated using Wahba’s function. . . . .	56
5.1	Training and test set organisation for the the BACKTEST and SIMLIVE experiments (1-day ahead). Each data-set used has identical training and test set organisations. However the dates will obviously differ amongst the different prediction horizons. . . . .	67
5.2	20 days of GEC SIMLIVE 20-day ahead predictions with 90% prediction intervals. . . . .	79

# Chapter 1

## Introduction

*“I can calculate the movements of heavenly bodies, but not the madness of people”*  
Sir Issac Newton, after losing £20,000 on the London stock-market.

### 1.1 Introduction

Possibly the most central (and difficult) problem in finance is predicting future market behaviour. Every time a trader places a trade, it is motivated by some prediction of future market prices. Traditionally, traders have relied on their own internal or instinctive models of market dynamics, developed over years of market participation to do this. However, over the last few decades more scientific methods have evolved that have recently begun to show significant promise. Neural networks have emerged as one of the most popular scientific prediction methodologies. However, despite their powerful universal approximation properties, neural networks have some serious limitations, especially when applied to difficult noisy problems such as financial time-series prediction. In this thesis we recognise these limitations and attempt to minimise the adverse effect they can have on the quality and usefulness of predictions.

We begin the thesis in this chapter by providing some background to financial prediction and describing in general terms the variety of modern statistical and econometric techniques that could be applied to solve the problems we aim to address.



We focus on ensemble techniques and argue that *bagging* (an abbreviation for “bootstrap aggregation”) (Breiman, 1996a) is particularly suitable for stabilising neural networks trained for financial time-series prediction. In this chapter we also describe the financial data-sets that we use, we describe the training vector set-up used for the experiments performed in later chapters, and outline the main contributions of the thesis.

In chapter 2 we begin to focus on the problems of instability and over-fitting in neural networks in more detail. In particular, we focus on the importance of diversity amongst neural networks in bagged ensembles and suggest a new approach that tunes this diversity by varying the fit of the individual networks in an ensemble. We continue this theme in chapter 3 by proposing a new early-stopping technique (the NeuralBAG algorithm) that optimises ensemble generalisation performance by tuning diversity in this way. Important features of NeuralBAG (in the context of financial time-series prediction) are its robustness to noise and the availability of model variance estimates from the bootstrapping process, which forms a core part of bagging.

In chapter 4 we attempt to address the issues relating to the absence of transparency in neural network models. We propose a new technique that provides estimates of model variance for bagged ensembles that can be used to generate confidence intervals with very good coverage (confidence intervals provide estimates of confidence in our prediction of the underlying true regression of the data). We then extend this work to develop a new technique that provides estimates of noise variance for ensemble predictions that can be combined with the estimates of model variance to generate prediction intervals with very good coverage (prediction intervals provide estimates of confidence in our prediction of the targets themselves). These prediction intervals essentially allow a trader to anticipate the quality of an ensemble prediction before the prediction horizon expires. This enables him to manage risk more effectively.

In chapter 5, using 8 financial time-series data-sets, we evaluate the performance of all techniques proposed in this thesis by measuring the predictive accuracy of the ensembles compared to econometric benchmarks. In this chapter we also identify recurring features in the results such as the sensitivity of the ensemble predictions to market volatility and prediction horizon. Finally in chapter 6 we conclude the thesis

by summarising its main contributions and identifying what further work should to be done to improve the techniques proposed in the thesis.

## 1.2 Predictability in financial markets

One of the most fundamental questions in finance is whether or not financial markets are predictable. This topic has been a focus of considerable attention amongst both researchers and practitioners in the financial world ever since financial markets have existed. In this section we briefly review this area of research. We focus on the popular *Efficient Markets Hypothesis* and discuss its relevance in the context of modern financial markets and practice.

Possibly the earliest published work on this subject is the doctoral thesis of the French mathematician Louis Bachelier (1900). He developed the foundations of a theory which, in essence, proposes that all stock market prices follow a random walk. His work was initially ignored by the financial world, but quickly gained recognition in other fields such as physics and is rumored by many (e.g. Granger and Morgenstern (1970)) to have inspired Einstein's seminal work on Brownian motion (Einstein, 1905). Other important early work on this subject includes the empirical study of Cowles (1933) who attempted to evaluate how accurately stock market analysts could predict prices. He found that they rarely out-performed a random walk and suggested that those who did were just very lucky. Similar theoretical and empirical contributions followed for several decades that supported the work of Bachelier and Cowles. Some of the most significant include (Kendall, 1953), (Mandelbrot, 1963), (Samuelson, 1965) and (Malkiel, 1992). Over the decades, this random walk theory of financial markets was refined and extended. Today, it is known as the Efficient Markets Hypothesis.

The Efficient Markets Hypothesis essentially consists of several random walk theories of financial markets. These theories differ primarily in how strict their definition of randomness is. A comprehensive review can be found in (Campbell *et al.*, 1997 (chapter 2)). The simplest and possibly most recognised of these is the *martingale* model. There are a number of ways in which this model can be expressed. However,



possibly the most natural is as

$$p_t = p_{t-1} + \epsilon_t, \quad (1.1)$$

where  $p_t$  denotes the price at time  $t$  (of a stock for example) and  $\epsilon_t$  denotes the residual (error) at time  $t$ . The residual is assumed to have zero mean and to be uncorrelated with all previous residuals. If this model holds true, then essentially it implies that the best predictor of tomorrow's price is today's price. It can also be applied to other prediction horizons e.g. the best predictor of next week's price is today's price.

Despite the simplicity of this model, it has been consistently misinterpreted by many in the financial world. The most common misinterpretation is that it implies all financial markets are unpredictable. However, a closer look at the model reveals that it is in fact much more specific than this. The exact, correct interpretation is that price changes are unpredictable if only *linear* combinations of previous price changes from the same market are used to generate the predictions – see (Granger and Morgenstern, 1970) for details. So, if one builds a model to predict prices that uses *non-linear* combinations of previous price changes, the martingale does not claim that it would be unsuccessful. Likewise, if one builds a multivariate model that uses exogenous financial predictor variables, the martingale model does not claim this would be unsuccessful either.

Although more sophisticated random walk models have been developed that also include, for example, the restriction that price changes are unpredictable even if *non-linear* combinations of previous price changes are used (see (Campbell *et al.*, 1997 (chapter 2))), they still over-simplify the dynamics of financial markets and underestimate the models that can be built to predict their movements. It is generally accepted today that the Efficient Markets Hypothesis has been over-emphasised in the econometrics literature. Many leading practitioners and academics e.g. Campbell *et al.* (1997), Farmer (1998) and Soros (1987) believe that all random walk theories of financial markets over-simplify the dynamics of a system that is ultimately driven by complex human behaviour. Models such as Farmer's theory of *financial ecology* (Farmer, 1998) are more sensible and, significantly, concur with the consensus view

today amongst practitioners that at least some predictability in financial markets exists. For example, changing business conditions and cycles, non-linear behaviour and relationships amongst financial instruments and markets and friction in the markets all naturally contribute towards a certain degree of predictability.

It is this modern view that we take in this thesis. We do not expect markets to follow a random walk, but do expect the predictability that does exist to be very difficult to find and exploit. We will show how the particular model of neural network learning that we use and develop plays a key role in finding and exploiting this predictability. Our analysis of its predictive performance on financial market datasets will confirm that a limited amount of predictability does exist in most financial markets and can be exploited.

## 1.3 Financial market prediction techniques

The financial market prediction techniques used today can be roughly divided into four main categories; *fundamental analysis* techniques, *charting* techniques, *econometric* techniques and modern “*exotic*” techniques such as chaos theory and hidden Markov models. In this section we review some of the more popular techniques that make up these categories.

### 1.3.1 Fundamental analysis

This category of financial prediction can be used for a wide variety of prediction tasks in finance. For example, changes in individual stock prices can be predicted by analysing the fundamental financial conditions and operating performance of a company. Usually, information such as the price/earnings ratio, profit/loss history, quality of the competition and the track-record of the management are used. To predict other financial market movements, such as a country’s future foreign exchange rate, other fundamental indicators are used e.g. interest rates, inflation and growth rates relative to other economies. A discussion and general review of some popular fundamental analysis techniques and strategies can be found in (Carret, 1997).



This traditional method of financial prediction enjoys widespread use in most financial institutions. However, it does have some serious limitations. For example, predictions generated using fundamental analysis techniques are typically very subjective i.e. humans play a very important role in the process. Although some objective quantitative analysis usually exists in the form of balance sheets, economic indicators and so on, typically the only “model” that exists is in the mind of an analyst or economist. Humans can be influenced by any number of subjective and emotional factors e.g. greed, fear and ulterior motive, all of which can corrupt the process of generating a prediction.

Another problem regarding the role played by humans in fundamental analysis is that we cannot easily identify complex non-linear relationships between variables and markets. It is widely accepted today that non-linearity exists in all financial markets. Indeed, as mentioned in section 1.2 it is an important source of predictability. Excluding the possibility that non-linearity may exist in modern financial markets will inevitably lead to poor predictions.

Finally, most fundamental analysis techniques operate under the assumption that every financial asset has an intrinsic value that can be determined by analysing fundamental indicators. However, in modern sophisticated financial markets the concept of value is no longer easy to identify or clearly define. For example, technology (especially internet) companies can consistently make a loss but still maintain a very high stock price. These companies are unique in that most of their *current* value is based on their projected future performance and future potential markets for their products. Integrating this into a fundamental analysis strategy is very difficult – fundamental analysis essentially assumes that the current value of a company is mostly reflected in its current fundamental indicators. This can be carried over to economic analysis as well. For example, the effect that “globalisation” has on financial markets is not very well understood. No economy today operates in isolation, rather, they influence each other in very complex ways. Therefore, the “value” of an economy is very difficult to determine and predict if only simple linear analyses of fundamental indicators are used.

### 1.3.2 Charting

Charting (sometimes called technical analysis) is an approach to financial prediction that is based on the belief that financial time-series exhibit trends and regularities in the form of geometric patterns. Predictions are usually generated by deducing from the historical trends or geometry of a series the probable future trend. A review of some popular charting techniques can be found in (Murphy, 1986).

Traditionally, charting has been unpopular amongst academics but popular with practitioners. Academics are uncomfortable with charting because it is not based on sound, proven principles but instead largely on intuition and interpretation. However, recently the differences between charting, fundamental analysis and econometrics have become somewhat blurred. For example, fundamental indicators such as earnings and econometric indicators such as volatility<sup>1</sup> are being integrated into the charts that charting practitioners use. However, despite this, charting still remains somewhat unprincipled and trends or projections open to interpretation. Also, most charting techniques can only identify linear trends and relationships. Those techniques that claim to identify and exploit non-linear trends usually use econometric indicators.

Although some recent studies by leading academics (e.g. Blume *et al.* (1994), Brock *et al.* (1992) and LeBaron (1996)) have given charting some credibility by highlighting various similarities with econometrics, there is still significant suspicion amongst academics.

### 1.3.3 Econometrics

Econometric techniques are amongst the most popular financial prediction techniques in modern financial institutions. They include simple moving average techniques, sophisticated volatility prediction techniques such as *autoregressive conditional heteroskedastic* (ARCH) methods and pricing models such as the *Capital Asset Pricing*

---

<sup>1</sup>The volatility of a financial market is a measure of how turbulent it is. Unlike prices, volatility cannot be observed and must be estimated. A rough estimate of volatility over a specified period is given by the corresponding standard deviation of prices over that period. More sophisticated estimates of volatility also exist (see e.g. (Alexander, 1998 (chapter 4))). We will re-visit this issue in significant depth in chapter 5.



*Model* (CAPM) and its variations. Other econometric theories such as the Black-Scholes option pricing model can also be manipulated to predict future market movements (see e.g. (Alexander, 1998 (chapter 4)) for some examples). All econometric techniques are based on proven principles and theories and are very popular amongst both academics and (sophisticated) practitioners. They are also objective and some can model non-linearities. A comprehensive review of the most popular modern econometric tools and techniques can be found in (Campbell *et al.*, 1997).

Econometric techniques are rarely used to predict the absolute value of a financial asset. Most applications lie in risk management (see (Alexander, 1998 (chapter 4)) for a survey of some popular applications). For example, the predicted volatility of an asset is an important factor in determining risk exposure. Today it is very difficult to distinguish between these econometric techniques and some of the so-called “exotic” techniques. For example, amongst some academics and practitioners in finance neural networks are no longer seen as some obscure artificial intelligence technique but instead are beginning to be accepted as a valid non-parametric, non-linear econometric modelling tool. For example, in (Zapranis and Refenes, 1999) an econometric framework is developed that shows how neural networks fit into theories such as the CAPM.

In this thesis we view neural networks in this way. The techniques we develop have the statistical rigour and theoretical foundations necessary to support this. We also support this by integrating an established ARCH volatility prediction method into our technique for generating prediction intervals.

### 1.3.4 “Exotic” techniques

It is quite difficult to define what constitutes an “exotic” financial prediction technique. For example, neural networks have traditionally belonged to this group, but as discussed above, are now beginning to be accepted as an econometric modelling technique. We will therefore restrict our definition of “exotic” as meaning a theoretically promising, but empirically unproven non-linear financial modelling technique.

Currently, one of the most popular (and fashionable) of these techniques is *chaos*

*theory* which has its origins in the physics literature (Lorenz, 1963). Its main contribution is that it shows how relatively simple systems of ordinary differential and difference equations can exhibit extremely complex dynamics. However, although chaos theory is quite mature in physics and a number of possible financial applications have been proposed (see e.g. (Scheinkman and Woodford, 1994), (Kennan and O'Brien, 1993) and (Pesaran and Potter, 1992)), there is little compelling evidence to suggest that it will emerge as an important modelling tool in finance. As discussed in (Campbell *et al.*, 1997) financial markets are not specific about functional forms and econometricians have no theoretical reason for expecting to find one form of non-linearity rather than another. Chaos theory is a very active area of research however and it is possible that current theoretical limitations and assumptions may not exist in the future.

Another technique that is gaining popularity as a financial prediction tool is the *hidden Markov model* method, which is widely used in speech recognition (Huang *et al.*, 1990). In (Fraser and Dimitriadis, 1994) hidden Markov models are used to predict entire conditional probability distributions<sup>2</sup> of future foreign exchange rates. In (Weigend and Shi, 1998) they are used to predict rare events such as stock market crashes by modelling the tails of conditional target distributions. Although hidden Markov models are relatively new as a financial prediction technique, they do seem to show promise for some specific applications.

The techniques described above are examples of only a few “exotic” techniques that have been applied to predict financial market movements. There are countless other techniques that have not been mentioned e.g. *Markov-switching* methods (Hamilton, 1989; Sclove, 1983), *support vector machines* (Vapnik, 1995; Mukherjee *et al.*, 1997) and *wavelet transforms* (Starck *et al.*, 1998; Aussem *et al.*, 1998). Given the significant activity and diversity of research in financial prediction, it is very difficult to predict what will be the most powerful technique of the future. However, with the possible exception of neural networks, few so-called “exotic” techniques have made the transition from academic research to financial practice. Most prediction

---

<sup>2</sup>Typically, models are built that only predict the mean of this distribution.



techniques that are currently used in finance were developed in the econometrics literature. Therefore, if an unusual new technique is to be accepted by practitioners it should first be adopted by the econometrics community. Keeping this in mind, we follow the lead of (Zapranis and Refenes, 1999) in this thesis and attempt to develop neural networks as an econometric technique that is statistically viable and robust. The terminology and methods that we adopt will reflect this.

## 1.4 Neural networks for financial time-series prediction

In this section we review neural network approaches to financial prediction and discuss their advantages and disadvantages relative to the techniques described in section 1.3 above.

Neural networks are essentially devices for *non-parametric* statistical inference i.e. when they are trained no assumptions about model or data are made *a priori*. Significantly, they are also universal function approximators – White (1988a), Cybenko (1989) and Ito (1993) have all shown how neural networks with a single hidden layer can approximate arbitrarily well any continuous function. These non-parametric, universal approximation properties give rise to a number of important advantages and disadvantages, especially when neural networks are applied to model financial time-series.

One of the most important advantages is that they can easily identify and model subtle non-linearities between variables and markets. These non-linearities are “learned” by the neural network and so are objective and require little input by the user. This learning process is robust to noise if a suitable network regularisation algorithm is used. Given the peculiar nature of financial time-series data i.e. no specific functional form, more stochastic than deterministic, non-stationary etc. these universal approximation properties are potentially very powerful.

It is these key advantages that have motivated most of the work in the literature that applies neural networks to financial prediction. One of the most important early

works was published by an econometrician (White, 1988b). This work gave credibility to neural networks amongst financial practitioners and paved the way for a large number of other works in this area. Good overviews of this work and summaries of important specific papers can be found in (Refenes, 1994), (Zapranis and Refenes, 1999) and (Campbell *et al.*, 1997).

However, the flexibility and power afforded by the advantages of neural networks outlined above also introduce some serious limitations. Few of these limitations have been addressed by existing work in the literature. Possibly the most serious is instability or high-variance. When neural networks are trained, especially on noisy, non-linear data (such as financial time-series data), small changes in parameters and/or training data can cause large changes in prediction performance. This is a serious limitation in an application area such as finance which is very sensitive to risk. Another disadvantage of neural networks is that typically they are “black-box” i.e. the particular functional form of the data learned by the neural network is not easy to identify given the complexity of the relationships between the inputs, weights and hidden units in the network. Finally, neural networks have traditionally been seen as very computationally expensive given the significant (usually gradient descent) optimisation that must be performed to estimate a good set of weights.

So how can we justify the use of neural networks given these disadvantages? Can the above problems be solved or at least partially overcome? In this thesis we attempt to do just that. We recognise the power and potential of neural networks as a financial time-series prediction tool and attempt to overcome the problems identified above using some novel statistical, econometric and computational techniques. In chapters 2 and 3 we develop our own particular variety of neural network ensemble technique to stabilise the networks and significantly improve prediction performance. In chapter 4 we combine our ensemble technique with an established econometric volatility prediction technique to estimate prediction intervals which in essence throw light onto the modelling process and can be used to manage risk. The computational problems are a little more difficult to overcome especially given that ensemble techniques require an increased level of computational power. However, the speed of modern computers minimises the negative impact of this. Also, in this thesis all code



was written in C-MPI<sup>3</sup>. This made running the experiments for even large ensembles very manageable. The final neural network models of financial time-series that we use for our evaluation in chapter 5 do not suffer in any serious way from the problems summarised above. Moreover, the universal approximation properties and related advantages remain the same.

## 1.5 Neural network ensemble learning

In this section we briefly review some popular neural network ensemble techniques and discuss in general terms how they can significantly improve the generalisation performance of neural networks. We put particular emphasis on bagging, the ensemble technique of choice for this thesis.

Recently, neural network ensemble techniques have gained widespread use amongst neural network practitioners (see (Sharkey, 1999) for a review of this research). There are many different varieties, but the most popular include some elaboration of bagging or *boosting* (Freund and Schapire, 1995). The basic idea of these techniques is to generate multiple versions of a predictor. When predictions from these versions are combined (averaged for example), smoother more stable predictions are generated. When applied to neural networks, these techniques can yield dramatic improvements in generalization performance (see e.g. (Carney and Cunningham, 1999a; Maclin and Opitz, 1997)). This is because neural networks are inherently unstable (Breiman, 1994; Breiman, 1996a) i.e. small changes in training set and/or parameter selection can produce large changes in performance. This idea of combining predictions from multiple versions has been around for quite a while – its origins in the neural network literature can be traced back to as early as 1965 (Nilsson, 1965). Significantly, it has also been used in other fields such as econometrics where it is called “forecast combining” (Granger, 1989). However, it has not gained widespread use until recently,

---

<sup>3</sup>This is the C programming language with MPI (Message Passing Interface) library extensions. C-MPI allows parallel code to be written that can run on any MPI compatible super-computer or workstation cluster. We use the *TCD CS Department SCI Cluster* (for more information on this machine see <http://www.cs.tcd.ie>).

largely because it requires significant computational resources, especially when applied to learners such as neural networks.

Bagging is widely accepted as one of the most popular neural network ensemble techniques. It uses the bootstrap (Efron and Tibshirani, 1993), a very popular statistical re-sampling technique, to generate multiple training sets and networks for an ensemble. Each ensemble training set is the same size as the original training set, but given that the bootstrap samples data with replacement, individual training samples may appear several times in an ensemble training set while others may be left out. Outputs from the trained networks in a bagged ensemble are combined using a simple average to produce smoother, more stable predictions. There are many works on bagging in the literature that study the technique in general terms i.e. without reference to any specific predictor e.g. (Breiman, 1996a; Breiman, 1996b; Rao and Tibshirani, 1997; Wolpert and Macready, 1996; Wolpert and Macready, 1996). Works that study bagging solely in the context of neural networks include (Carney and Cunningham, 1999a; Heskes, 1997a; Zhang, 1999).

Boosting techniques use more elaborate training set generation and network combination methods. There are a number of related boosting techniques, the most popular being *arcing* (Breiman, 1996c) and *ada-boosting* (Freund and Schapire, 1996). Unlike bagging, all boosting techniques are inherently sequential in nature – the probability of selecting a training example for a new ensemble training set is not equal across the original set of training examples; instead, this probability depends upon how often that example performed poorly across the set of previously trained networks. The idea here is to put more emphasis on training examples that are difficult to learn, which in essence (using machine learning terminology) pushes a “weak-learner” towards being a “strong-learner”. The primary difference between arcing and ada-boosting is that ada-boosting uses a weighted average to combine the networks in the ensemble whereas arcing uses a simple average.

Bagging has a number of important advantages over boosting techniques when applied to noisy real-world tasks such as financial time-series prediction. One of the most important is the ease with which confidence and prediction intervals can be



computed (Carney *et al.*, 1999; Heskes, 1997b). Another is the robustness and stability of the technique itself – it can be shown that it will always perform at least as well as an individual predictor, as long as the predictor is unstable (Breiman, 1996a). Boosting techniques on the other hand have been shown to be sometimes quite unstable. Maclin and Opitz (1997) showed that arcing sometimes produces results that are the same as or worse than a single network. However, other times it significantly out-performs individual classifiers and bagging. Maclin and Opitz (1997) also showed that on some data-sets ada-boosting produces results that are significantly worse than using a single network whereas on other data-sets it significantly out-performs any other method. Most of this erratic behaviour has been attributed to the sensitivity of boosting techniques to noise. Freund and Schapire (1996) suggest that the re-sampling procedure used by boosting techniques can over-emphasise noisy training examples by interpreting them as training examples that are just difficult to learn (but contain useful information). Boosting techniques cannot recognise the difference between training examples that are difficult to learn and those that are just noise.

Given the robustness of bagging to noise, its stability and the ease with which confidence and prediction intervals can be computed, bagging (or some elaboration of the technique) is the natural choice for financial time-series prediction.

## 1.6 Finding predictability

In this section we describe how we attempt to uncover hidden predictive structure in the financial market data-sets studied in this thesis. We use a number of new empirical results from the econometrics literature that uncover dependancies amongst related financial variables and markets.

Possibly the most valuable and interesting predictive structure that exists in financial markets is the relationship between financial market prices, trading volume<sup>4</sup> and volatility. For example, in rising markets volume increases. In volatile markets volume also increases. Although these dependencies seem quite linear, the most valuable and predictive structure is expected to be non-linear and difficult to identify in

---

<sup>4</sup>The volume of a market is the number of shares traded in that market over a specified period.

most markets – see (Gallant *et al.*, 1993; Karpov, 1987; LeBaron, 1982) for some interesting discussions and results on this topic. To exploit any such predictive structure, we include volume and volatility information as part of each training vector in each data-set.

For foreign exchange markets, interest rate changes can play an important role. For example, if interest rates increase in an economy there is likely to be an increase in the flow of foreign currency into that economy. However, this relationship is not always linear and easy to identify e.g. increases in interest rates have been known to devalue a currency under certain circumstances (see Corden, 1995 for a discussion). Nevertheless, it is generally accepted that an important relationship does exist, albeit potentially non-linear. To exploit any contribution changes in interest rates may make to the process of generating a prediction, we include the difference in the discount rate of interest between the two economies that determine a foreign exchange (cross-) rate (e.g. USD/JPY (US-Dollar/Japanese-Yen)). The exact set-up is described in section 1.7.

Other sources of predictive structure that have been identified in the econometrics literature are so-called “calendar effects”. It is generally accepted that all financial markets exhibit calendar effects. One of the most documented is the “January Effect” – the fact that smaller capitalisation stocks out-perform larger capitalisation stocks at the turn of the year. Keim (1989) provides some elaborate econometric reasons for the occurrence of this effect. Other calendar effects on smaller time-scales are also suspected to exist. For example, trading behaviour at specific times over certain time horizons can sometimes be very similar in terms of volume, volatility and so on. Some interesting discussions relating to these effects can be found in (Campbell *et al.*, 1997 (chapter 1)).

Related to these calendar effects are “lead-lag” structures. Here, some markets lead while others follow. Examples include the phenomenon of larger capitalisation stocks leading smaller capitalisation stocks. Also, general movements in indices can move stocks in directions that do not reflect their individual performance. See (Campbell *et al.*, 1997 (chapter 1)) for a discussion of these lead-lag structures and a summary of some useful econometric tools that can identify when they occur. To

exploit (at least in part) these calendar effects and lead-lag structures we include the date, day of the week and (for the stock-market data) the major indices (S&P500<sup>5</sup>, NYSE<sup>6</sup>, DJIA<sup>7</sup>) in each training vector for the data-sets in this thesis. Again, the exact set-up is described below in section 1.7.

## 1.7 The data

In this section we describe the exact set-up of the financial time-series data-sets<sup>8</sup> used for the experiments performed in this thesis. We also outline our motivation for including each data-set and describe how exogenous variables are combined with price information to create training vectors.

### 1.7.1 Stocks

In total we use 4 stock-market data-sets. We use 11 years (1/9/88-1/9/99) of daily closing price data from two mainstream industry stocks General Electric Corporation (GEC) and Coca-Cola and two mainstream technology stocks Microsoft Corporation and International Business Machines (IBM). These stocks have histories that are long enough to build useful models that generate good predictions. We also believe that together they broadly represent the behaviour of the majority of stocks in the financial markets. We use the following training vector set-up for each data-set

$$(r_{t-4}, \dots, r_t), (vl_{t-4}, \dots, vl_t), (vt_{t-4}, \dots, vt_t), (sp_t, ny_t, dj_t), (d_t, m_t, wd_t), r_{t+1}. \quad (1.2)$$

---

<sup>5</sup>Standard and Poor's 500 index.

<sup>6</sup>New York Stock-Exchange index.

<sup>7</sup>Dow-Jones Industrial Average index.

<sup>8</sup>All (unprocessed) stock-market data was provided by *Riskmetrics Ltd.* and foreign exchange data by *Beacon Foreign Exchange Ltd.*



Here  $r_t$  denotes the log-return<sup>9</sup> at time  $t$ . Note that we use 5-day lagged log-returns. We do this to model any temporal structure that may exist in the returns series. Our motivation for using a 5-day lag is simply that it seems reasonable – it is long enough to model temporal structure (markets can change significantly over a 5 day period) but sufficiently short enough so as to maintain parsimony in the models. For the same reason we also use 5-day lagged returns of volume and volatility, where  $vl_t$  is the volume and  $vt_t$  is the volatility at time  $t$ . To model any lead-lag structure between the stocks and the major indices we include the log-returns at time  $t$  of the S&P500, the NYSE and the DJIA indices ( $sp_t, ny_t, dj_t$ ). To model any calendar effects we include at time  $t$  the day of the month, the month, and the day of the week ( $d_t, m_t, wd_t$ ). Given these input features we expect the neural network to generate predictions 1-day ahead i.e. for  $r_{t+1}$ . We also perform experiments that generate predictions over other time horizons. In chapter 5 we will re-visit this issue.

## 1.7.2 Stock-market indices

We use 11 years (1/9/88–1/9/99) of 3 stock-market index data-sets of the major U.S. indices S&P500, NYSE and DJIA. The training vector set-up is very similar to the stock-market training vector set-up. To generate 1-day ahead predictions for the S&P500 index we use

$$(sp_{t-4}, \dots, sp_t), (vl_{t-4}, \dots, vl_t), (vt_{t-4}, \dots, vt_t), (ny_t, dj_t), (d_t, m_t, wd_t), sp_{t+1}. \quad (1.3)$$

Here, as for the stock-market data-sets, we use 5-day lagged returns for our target series. We also include 5-day lags of volume, volatility, related index and date information. Using a similar training vector set-up we also generated predictions for the NYSE data-set and predictions over other time horizons. We also will re-visit this in

---

<sup>9</sup>The log-return is a measure of relative change in prices over a fixed period. The log-return at time  $t$  is given by  $\log(p_t) - \log(p_{t-1})$ , where  $p_t$  is the price at time  $t$ . Almost every econometric study of financial markets uses returns rather than prices. There are a variety of reasons for this, but the most important in the context of this thesis is that returns exhibit some desirable statistical properties such as stationarity and ergodicity. See (Lucas, 1978) and (Campbell *et al.*, 1997 (chapter 1)) for a discussion.



chapter 5.

### 1.7.3 Foreign exchange

We use 5 years (20/5/92–20/5/97) of 2 foreign exchange market data-sets USD/JPY and CHF/JPY (Swiss-Franc/Japanese-Yen). The training vector set-up for each data-set is

$$(r_{t-4}, \dots, r_t), (vt_{t-4}, \dots, vt_t), (i_{t-4}, \dots, i_t), (d_t, m_t, wd_t), r_{t+1}. \quad (1.4)$$

Here  $(r_{t-4}, \dots, r_t)$  denotes a 5-day lag of foreign exchange rate returns,  $(vt_{t-4}, \dots, vt_t)$  a 5-day lag of volatility and  $(i_{t-4}, \dots, i_t)$  a 5-day lag of discount interest rate differences. Again, we also perform experiments that generate predictions over other time horizons and in chapter 5 will re-visit this issue.

## 1.8 Thesis contributions

In this section we summarise the main contributions of the thesis:

- We propose a *new neural network ensemble technique* (the NeuralBAG algorithm) based on bagging that optimises generalisation performance by tuning diversity. Advantages of this technique over other ensemble techniques (in the context of financial time-series prediction) are its robustness to noise and the availability of bootstrap model variance estimates.
- We propose a *new technique for estimating model variance* (confidence interval) estimates for neural network ensembles. These estimates adjust the model variance estimates provided by bagging so that they better reflect the (lower) variance of ensemble predictions. We show how these confidence intervals have significantly better coverage than those generated by an alternative method.
- We propose a *new technique for generating prediction intervals*. These intervals incorporate both the uncertainty caused by model variance (estimated as above) and

noise variance (estimated using an established econometric technique). We demonstrate the excellent coverage that these intervals have and also discuss how they can be applied to manage risk in trading.

- We use a number of new and established empirical research results from the econometrics literature to *propose a number of new novel training vector structures*. We also perform extensive experimentation to *estimate good model parameters for the econometric noise variance prediction technique* used for estimating the prediction intervals.
- We perform a large number of experiments to test our techniques. The results of these experiments both validate our techniques but can also be used to *draw more general inferences* such as the suitability of machine learning techniques for financial market prediction and their sensitivity to the volatility of markets and prediction horizons.
- The *success of the multi-disciplinary approach* taken in this thesis demonstrates the potential of this approach for solving other difficult real-world prediction problems.

## 1.9 Summary

In this chapter we discussed the nature of predictability in financial markets and reviewed some popular econometric, economic and exotic techniques used to generate predictions. We positioned neural networks relative to these techniques, discussing their strengths and weaknesses. We proposed how instability in neural networks can be overcome using ensemble methods and reviewed some of the more popular techniques. We argued that for financial time-series prediction bagging shows significant promise and is the natural choice especially given its robustness to noisy data and the ease with which measures of model uncertainty can be estimated. We described in significant detail the financial data-sets used for the experiments in this thesis and

the training vector set-up. Finally, we outlined the main contributions of the thesis.

# Chapter 2

## Bagging and bootstrapping

### 2.1 Introduction

In chapter 1 we briefly reviewed a variety of neural network ensemble techniques and discussed in general terms how bagging is particularly suited to financial time-series prediction. In this chapter we study bagging in more detail. We begin in section 2.2 by outlining the underlying principles of bootstrapping, the statistical resampling technique at the heart of bagging. In section 2.3 we introduce some notation, provide a concrete description of bagging and analytically show how it improves the performance of neural networks. This analytical work highlights the importance of diversity amongst networks in a bagged ensemble and in section 2.4 we discuss the implications this has for how a bagged ensemble should be trained. Note that all the work presented in this chapter is descriptive and analytical. Any claims made will be substantiated by extensive experimentation in chapter 3.

### 2.2 Bootstrapping

To understand how bagging works, one must first understand how bootstrapping works. Bootstrapping is a very general computer intensive statistical re-sampling technique. In its simplest form bootstrapping is used for estimating measures of



uncertainty and bias in parameters generated from independent and identically distributed variables. It was proposed by Efron (1979) and has become very popular as computers have become more powerful. The term “bootstrap” was well chosen by Efron – it implies “pulling oneself up by the bootstraps”. As will be illustrated in this section, in a statistical sense, this is exactly what the bootstrap does. The technique has been generalised extensively and applied to problems such as confidence interval estimation e.g. (DiCicco and Tibshirani, 1987), (Efron and Tibshirani, 1993); model selection e.g. (Efron and Tibshirani, 1997) and (Breiman, 1996b); and variance correction e.g. (Breiman, 1996a). A popular way to illustrate in general terms how the bootstrap works (see e.g. (Hjorth, 1994)) is to present two versions of a simple estimation problem; an artificial bootstrap version and a real version. Comparisons are drawn between the real and artificial versions to highlight the underlying principles of the technique.

In the *real version* of the problem we are given a set of  $n$  observations  $x_1, \dots, x_n$ . However, very little information about the underlying distribution  $P$  of these observations is available – we might know it is continuous for example, but little more. A parameter  $\theta = g(P(\cdot))$ , defined by the distribution, is estimated using the observed data to give us  $\hat{\theta} = s(x_1, \dots, x_n)$ . This parameter could be the median or mean of the distribution for example.

In the *bootstrap version* of the problem, the true distribution  $P$  is replaced by an empirical version of it. This empirical distribution, which we will denote as  $\hat{P}$ , is the discrete distribution that puts probability  $1/n$  on each of the observed values  $x_1, \dots, x_n$ . A more exact expression of this is given by

$$\hat{P}(x) = \frac{\#\{x_i \leq x\}}{n}, \quad (2.1)$$

where  $\#\{A\}$  denotes the number of times the event  $A$  occurs. An example set of observations and its corresponding empirical distribution is illustrated in figure 2.1.

A parameter  $\tilde{\theta} = g(\hat{P}(\cdot))$ , the empirical analogue of  $\theta$ , is also defined. It is estimated using “observations” drawn from the empirical distribution  $\hat{P}$ . A single set of  $n$  observations drawn from the empirical distribution  $\hat{P}$  is called a bootstrap sample

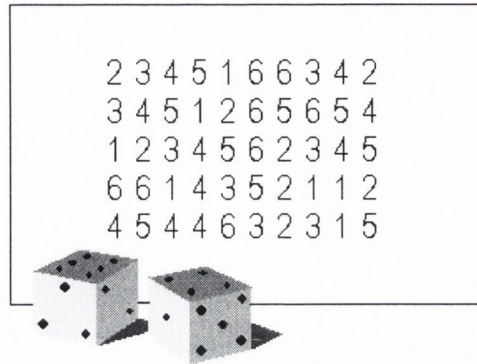


Figure 2.1: 50 rolls of a die. The possible roll outcomes 1, 2, 3, 4, 5, 6 occurred 7, 8, 8, 10, 9, 8 times respectively. Therefore the empirical distribution is  $(7/50, 8/50, 8/50, 10/50, 9/50, 9/50)$ .

$x_1^*, \dots, x_n^*$  where  $x_i^* \in \hat{P}(x)$ . A single estimate for the parameter  $\tilde{\theta}$  is therefore given by  $\hat{\theta}^* = s(x_1^*, \dots, x_n^*)$ . If we repeat this procedure many times (i.e.) generate a large number of bootstrap samples, we can generate multiple estimates for the parameter  $\tilde{\theta}$ . From these we can estimate statistics of interest for the bootstrap version of the problem. For example, we can compute the standard error of  $\hat{\theta}^*$  from

$$se_{\hat{P}}(\hat{\theta}^*) = \sqrt{\frac{1}{B-1} \sum_{b=1}^B (\hat{\theta}_b^* - \tilde{\theta})^2} \quad (2.2)$$

where

$$\tilde{\theta} = \sum_{b=1}^B \frac{\hat{\theta}_b^*}{B} \quad (2.3)$$

and  $B$  is the number of bootstrap samples. Note that the ideal number of bootstrap samples generated takes  $B = \infty$ . However, in practice, the value for  $B$  chosen is largely determined by the amount of computer processing power available<sup>1</sup>.

Now that we have established how statistics of interest can be estimated for the artificial bootstrap version of the problem, how do we relate this to the real problem? The bold claim of the bootstrap is that *the estimation properties of the bootstrap problem can be used to judge the estimation properties of the real problem*. Using

<sup>1</sup>See (Efron and Tibshirani, 1993) for more on this.



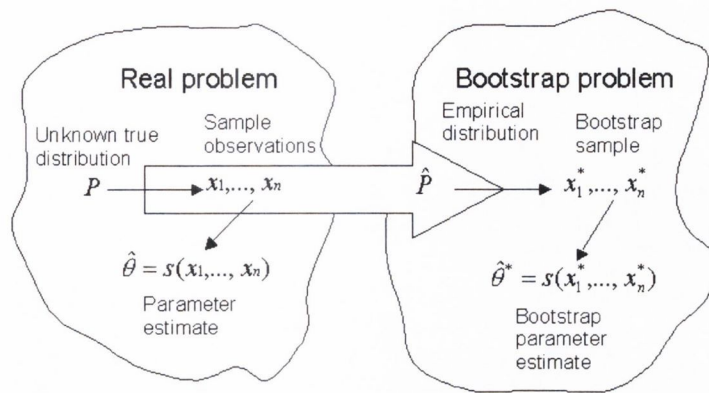


Figure 2.2: A comparison of the real and bootstrap versions of a parameter estimation problem. In the real version of the problem we are given a finite set of  $n$  observations from some unknown probability distribution  $P$ . We use these observations to generate a single parameter estimate  $\hat{\theta}$ . We also use these observations to generate an empirical distribution  $\hat{P}$ . Note that in the diagram both the original observations and the empirical distribution are enclosed in the large arrow. This is to denote that observations from the empirical distribution are generated by just sampling at random with replacement from the original set of observations. Using this any number of bootstrap samples from the empirical distribution can be generated and therefore also any number of bootstrap parameter estimates  $\hat{\theta}^*$ .

this we can use the standard error estimate for  $\hat{\theta}^*$  in the bootstrap problem as an estimate for the standard error of  $\hat{\theta}$  in the real problem. This idea can be generalised for a large number of parameters and statistics of interest. A diagram that illustrates the relationship between the real and artificial bootstrap versions of a parameter estimation problem is given in figure 2.2.

To the theoretical purist the bootstrap may seem difficult to accept. However, its success is reflected in the excellent results yielded by many studies of its application in a variety of domains – see (Efron and Tibshirani, 1993) and (Davidson and Hinkley, 1997) for just a sample. A more comprehensive introduction to the technique can be found in (Efron and Tibshirani, 1993). Some asymptotic results, theoretical analyses and comparisons to similar techniques can also be found here and in (Bickel and Freedman, 1981).

## 2.3 Bagging

To describe bagging (in the context of regression) in concrete terms, let us now introduce some notation. Assume we are given a set of  $N$  data pairs  $T = \{(t_n, \mathbf{x}_n)\}_{n=1}^N$ , described by the distribution  $P$  and generated according to

$$t = f(\mathbf{x}) + \epsilon(\mathbf{x}), \quad (2.4)$$

where  $t$  is the observed target value,  $f(\mathbf{x})$  is the true regression and  $\epsilon(\mathbf{x})$  is noise with zero mean. When we train a neural network on such data, our aim is for the network to approximate  $f(\mathbf{x})$ . Let us denote this neural network approximation as  $\phi(\mathbf{x})$ . Bagging aims to improve this individual network approximation by generating bootstrap re-samples of  $T$  and using these re-sampled training sets  $\{T_b^*\}_{b=1}^B$  to generate multiple bootstrap versions. Each bootstrap re-sample  $T_b^*$  consists of  $N$  data pairs, sampled at random with replacement from the empirical distribution  $\hat{P}$ . The training sets  $\{T_b^*\}_{b=1}^B$  give us a set of networks  $\{\phi_b^*(\mathbf{x})\}_{b=1}^B$ . Bagging aggregates these bootstrap versions by averaging to form a bagged prediction,

$$\phi_{bag}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \phi_b^*(\mathbf{x}). \quad (2.5)$$

See figure 2.3 for a schematic outline of this.

The description of bagging given above does not quantify the extent to which bagging can improve the generalisation performance of neural networks or under what circumstances. To study bagging in more depth and clearly illustrate its properties and limitations, we will now show how the general work of Krogh and Vedelsby (1995) on neural network ensembles applies to bagged ensembles.

Using the notation introduced above and the terminology introduced in (Krogh and Vedelsby, 1995), let us define the *ambiguity* of a single member of a bagged ensemble on a prediction for  $t$  as

$$a_b(\mathbf{x}) = (\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2, \quad (2.6)$$

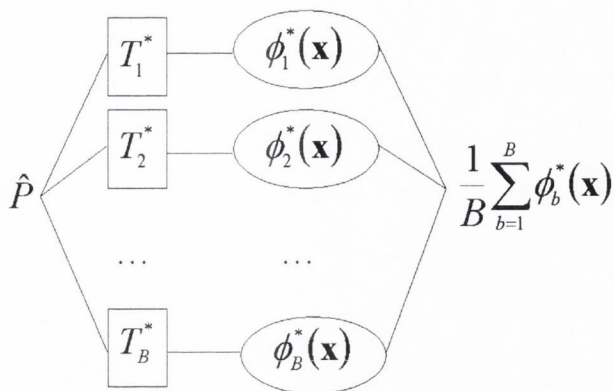


Figure 2.3: A bagged neural network ensemble. Training sets are generated by sampling from the bootstrap empirical distribution  $\hat{P}$ . These training sets are used to train an ensemble of neural networks. The outputs of the networks in the ensemble are averaged to produce the bagged ensemble prediction.

and the *ensemble ambiguity* to be

$$a_{bag}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B (\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2. \quad (2.7)$$

The ensemble ambiguity is a variance measure – it quantifies the disagreement or *diversity* amongst the networks in an ensemble on a prediction for  $t$ . Define the generalisation error of an individual network on a single prediction for  $t$  as

$$e_b(\mathbf{x}) = (\phi_b^*(\mathbf{x}) - t)^2, \quad (2.8)$$

and on an ensemble prediction as

$$e_{bag}(\mathbf{x}) = (\phi_{bag}(\mathbf{x}) - t)^2. \quad (2.9)$$

Note that here we assume the inputs are independent of the training set i.e. they are test set inputs. We also define the average of the generalisation errors of the individual networks across the ensemble as

$$\bar{e}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B (\phi_b^*(\mathbf{x}) - t)^2. \quad (2.10)$$

This important measure will be compared to the ensemble generalisation error to provide valuable insights. It can be rewritten as

$$\bar{e}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B (\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}) + \phi_{bag}(\mathbf{x}) - t)^2 \quad (2.11)$$

and after a little manipulation (see (Zenobi, 1999) for details) as

$$\bar{e}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B (\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2 + (\phi_{bag}(\mathbf{x}) - t)^2. \quad (2.12)$$

Using equations (2.4) and (2.6) above this gives us

$$\bar{e}(\mathbf{x}) = a_{bag}(\mathbf{x}) + e_{bag}(\mathbf{x}), \quad (2.13)$$

or

$$e_{bag}(\mathbf{x}) = \bar{e}(\mathbf{x}) - a_{bag}(\mathbf{x}). \quad (2.14)$$

For statistical rigour we now average this over the input distribution  $P$ . We begin by denoting the average generalisation error over  $P$  of an individual network in an ensemble to be

$$E_b = \int d\mathbf{x} P(\mathbf{x}) e_b(\mathbf{x}). \quad (2.15)$$

Using this and equation (2.7), we express the average generalisation over  $P$  across all the networks in an ensemble as  $\bar{E}$ . Similarly we denote the average ambiguity over  $P$  for an individual network in an ensemble as

$$A_b = \int d\mathbf{x} P(\mathbf{x}) a_b(\mathbf{x}), \quad (2.16)$$

and across all the networks in an ensemble as  $\bar{A}$ . Finally, using equation (2.6) we express the average bagged ensemble generalisation error over  $P$  as

$$E = \int d\mathbf{x} P(\mathbf{x}) e_{bag}(\mathbf{x}). \quad (2.17)$$



Using equation (2.11) this gives us

$$E = \bar{E} - \bar{A}. \quad (2.18)$$

This expression is extremely valuable. It relates diversity (ensemble ambiguity) in a bagged ensemble to the generalisation error of a bagged ensemble. *As long as the average of the individual network generalisation errors in an ensemble remains constant, increasing diversity will improve generalisation performance.* It also confirms intuition – an ensemble that consists of a thousand identical networks will not perform any better than an individual network. Note that, as described in (Krogh and Vedelsby, 1995), this expression is a general result and can be applied to most neural network ensembles.

What specific implications does it have for bagged ensembles? The training sets in a bagged ensemble are generated by sampling with replacement from the original training set  $T$ . The probability an individual training sample from  $T$  will not be part of a bootstrap re-sampled training set is  $(1 - 1/N)^N \approx 0.368$ , where  $N$  is the number of training samples in  $T$ . This means that only approximately 63% distinct training samples from  $T$  will be included in a bootstrap training set. This, of course, directly affects the average individual network generalisation error  $\bar{E}$  – given fewer distinct training samples it may not remain constant but instead may fall. However, the bootstrap sampling dramatically increases the diversity amongst the training sets in the ensemble and this combined with the inherent instability in neural networks will increase the ambiguity term  $\bar{A}$ . This is a key point – as long as the increase in ambiguity is larger than the decrease in average individual network generalisation error it is worthwhile bagging a predictor. This is easy to achieve for neural networks because they are unstable and so bagging can consistently improve generalisation performance. So, in summary, we can conclude that the more unstable a predictor is the more can be gained from bagging. Breiman demonstrates this empirically by bagging stable and unstable predictors and comparing relative improvements in generalisation error (Breiman, 1996a). However, he does not quantify this analytically by including an ambiguity term in his derivations.



## 2.4 Diversity and ensemble generalization performance

Now that we have established the importance of *generating* diversity in neural network ensembles, the question that remains is: how do we *tune* diversity so that ensemble generalisation performance is optimised? In this section we discuss the merits and disadvantages of two possible methods for tuning diversity. Our conclusions are used as motivation for techniques developed in later sections that optimise ensemble generalisation performance.

We will begin by discussing the effectiveness of training set resampling as a method for generating diversity. In the previous two sections we discussed how bagged ensembles use bootstrapping to generate diversity. How can we adjust this process to tune diversity? As outlined in (Krogh and Vedelsby, 1995), but in the context of simple linear ensembles, one could adjust the re-sampling process. Applying this to bagged ensembles, to generate more diversity fewer distinct training examples could be sampled, pushing down the standard bootstrap distinct sample rate of  $\approx 63\%$ . Similarly, to generate less diversity more distinct training examples could be sampled. This approach may seem promising at first, but in practice it has some major drawbacks. For example, the optimal resampling rate will vary depending on a number of factors including the level of noise in the training set, the size of the training set and the size of the ensemble. Therefore a global optimal resampling rate does not exist. Instead, it would have to be estimated for each individual training scenario. Another more serious drawback is that such “pseudo-bootstrapping” does not have a large body of theoretical work to support it as does conventional bootstrapping. It would therefore be unwise (and possibly incorrect) to use this form of bootstrapping as a method for generating other statistics of interest such as confidence and prediction intervals, which can be conveniently estimated as part of the bootstrapping process in bagged ensemble training (Carney *et al.*, 1999), (Heskes, 1997b). Losing such valuable side effects of standard bootstrapping is a high price to pay. These drawbacks motivate the search for an alternative method for tuning ambiguity.

One possible approach highlights a fascinating feature of neural network ensembles. If we over-fit the networks in an ensemble we generate more diversity. If we under-fit the networks we generate less diversity. Therefore, if we use standard bootstrapping to generate a basic level of diversity we can fine-tune the diversity by controlling the fit of each network in the ensemble. In chapter 3 we develop and evaluate a new technique that optimises the generalisation performance of bagged ensembles using this idea. The results of our analysis highlight an interesting feature of bagged (back-propagation) ensembles – a controlled level of over-fitting can consistently improve overall ensemble generalisation performance. This result is consistent with work performed on other ensembles, e.g. in (Sollich and Krogh, 1996) it is demonstrated for simple linear ensembles and in (Husmeier, 1999) it is demonstrated for Random Vector Functional Link (*RVFL*) ensembles. A key advantage of this approach is that ensemble diversity tuning and individual network parameter tuning can be unified and performed simultaneously using the same algorithm.

## 2.5 Summary

In this chapter we examined in significant detail the underlying principles of bagging and bootstrapping. In particular, we analytically justified the use of bagging as a technique for stabilising high variance predictors such as neural networks and discussed the importance of diversity. We proposed a new method for tuning this diversity so that ensemble generalisation performance is optimised and outlined its key advantages compared to other approaches. The analytical results and claims made in this chapter will be supported by extensive experimentation in chapter 3.

# Chapter 3

## Optimizing generalisation performance

### 3.1 Introduction

In chapter 2 we demonstrated the importance of diversity in bagged ensembles and suggested that it should be tuned to optimise ensemble generalisation performance. We argued that a good way to do this is to vary the fit of the individual networks in an ensemble. In this chapter we support these claims by developing and evaluating a new technique which we call the *NeuralBAG* algorithm that optimises ensemble generalisation performance by tuning diversity in this way.

A more specific outline of this chapter goes as follows. In section 3.2 we discuss the merits and disadvantages of different approaches to optimising generalisation performance and argue that early-stopping is particularly suitable for bagged ensembles. In section 3.3 we support this argument by describing a simple but effective technique that applies early-stopping to bagged ensembles. In section 3.4 we detail the *NeuralBAG* algorithm and in section 3.5 evaluate its performance on some of the financial time-series data-sets summarised in chapter 1. Finally, in section 3.6 we discuss how our work relates to similar work on neural network ensembles.



## 3.2 Optimising generalisation performance using early-stopping

To optimise a neural network's generalisation performance is to optimise its performance on test or out-of-sample data i.e. data not used during training. Much of modern neural network research is focused on developing techniques that optimise the performance of neural networks in this way. A large number of these techniques exist. They can however be loosely divided into two main categories – pruning techniques and regularisation techniques.

Pruning techniques work by explicitly choosing an optimal number of hidden units and weights for a network. Examples include optimal brain damage pruning (Le Cun *et al.*, 1990) and optimal brain surgeon pruning (Hassibi and Stork, 1993). Regularisation techniques work by constraining or penalising the training of a network so that smoother, more general models are built. Examples of this approach include weight decay (Hertz *et al.*, 1991) and early-stopping, which cannot be fairly attributed to any single author.

It is generally accepted that, in practice, early-stopping based techniques are amongst the most popular for optimising generalisation performance. This is especially true for ensemble learning – see e.g. (Heskes, 1997a). We attribute this to a number of factors. Firstly, they are much more general than pruning techniques and can be easily adapted to a large variety of architectures and algorithms. Secondly, they do not require any *a priori* assumptions about the model (network) or training data – all pruning algorithms require prior assumptions, some of which are difficult to justify – see e.g. (Reed, 1993). Thirdly, they are easy to understand and implement. Finally, and most importantly for this thesis, they work very well with ensembles. For example, as we will show in this section 3.4 they allow diversity to be easily tuned, which as discussed in chapter 2, is a very important consideration for bagged ensemble training. Given these factors, we pursue the development and evaluation of an early-stopping based technique in this thesis.

The basic idea of early-stopping is to terminate neural network training as soon



as some estimate of generalisation error begins to increase. This estimate of generalisation error is usually generated using cross-validation (Stone, 1974) or some related statistical re-sampling technique. The simplest and possibly most popular early-stopping technique for neural networks uses “hold-out validation” (often just called cross-validation) to estimate generalisation error.

Hold-out validation consists of using  $N - N_v$  of the available  $N$  training examples for training the network i.e. estimating the network weights  $\hat{\mathbf{w}}_{N-N_v}$ , and the remaining  $N_v$  examples for computing the generalisation error estimate. This generalisation error estimate can be expressed in terms of a neural network cost function as

$$G_{hov} = \frac{1}{N_v} \sum_{n=N-N_v+1}^N (t_n - \phi(\mathbf{x}_n; \hat{\mathbf{w}}_{N-N_v}))^2, \quad (3.1)$$

The estimate  $G_{hov}$  is measured at regular intervals during training (usually every epoch or training iteration). When it begins to increase, training is stopped.

This is early-stopping in its simplest form. However, it is widely accepted that the estimate  $G_{hov}$  can fluctuate during training if there is noise and/or non-linearity in the training data. Therefore, to simply terminate training as soon as it begins to increase can result in choosing a sub-optimal set of weights. One way to overcome this is to train a network to convergence and then choose the set of weights that correspond to the minimum value of  $G_{hov}$  as the optimal set. Other more elaborate techniques also exist. For example, in (Prechelt, 1994a) a patience threshold, set by the user, is used to determine for how long training should continue after  $G_{hov}$  begins to increase during training. Although such techniques can improve overall training speed by circumventing the requirement that a network should be trained to convergence, the patience threshold is not easy to estimate given that it can vary significantly depending on the training data and network architecture used. The simpler option of training a network to convergence and then choosing the optimal set of weights will always be more robust and reliable. Given this, combined with the general view today that training speed is not as vital an issue as it used to be given modern computer processing power, we will pursue the simpler approach in this thesis.

To describe in more concrete terms the variety of early-stopping adopted in this thesis, let us extend equation (3.1) above so that the generalisation error estimates and the network weights are “indexed” by the number of epochs they correspond to,

$$G_{hov}(e) = \frac{1}{N_v} \sum_{n=N-N_v+1}^N (t_n - \phi(\mathbf{x}_n; \hat{\mathbf{w}}_{N-N_v}(e)))^2. \quad (3.2)$$

Here,  $G_{hov}(e)$  is the hold-out validation estimate of generalisation error for a network trained to epoch  $e$ . Values for  $G_{hov}(e)$  are found for  $e = 1, \dots, E$ , where  $E$  is the “maximum” number of epochs and chosen to ensure convergence. A simple patience threshold idea, similar to that used above could be used here e.g. if training error does not change significantly for 100 epochs, terminate training. Once the network has been trained through the range of epochs specified, the user chooses the value for  $e$  that provides the best estimated generalisation performance. The set of weights saved on disk corresponding to this best  $e$  is chosen as the optimal set.

An obvious drawback of early-stopping techniques that use hold-out validation is that only  $N - N_v$  of the available training data can be used for estimating network weights. This damages the generalisation performance of a network. If a small value for  $N_v$  is chosen to minimise this effect, then the estimate of generalisation error will be compromised. One way to overcome this dilemma is to use a different variety of cross-validation. For example, “leave-one-out” cross-validation (Stone, 1974) does not require any training data to be sacrificed by the user. There are a number of different ways this technique (and elaborations of it such as  $k$ -fold cross-validation) can be applied to estimate generalisation error for neural networks – see (Geman *et al.*, 1992) for a general discussion.

However, in the context of this paper, a key point regarding the leave-one-out cross-validation family of techniques is that when applied to bagged ensembles they fail. This is because the bootstrapped training sets in a bagged ensemble are generated by sampling *with* replacement. This means that a single training example may occur several times in a training set. If a training example is removed from a training set for leave-one-out cross-validation, a replicate of it may still remain. This will produce an estimate of generalisation error that is biased downwards, as the training



example removed for cross-validation is not truly out-of-sample. However, sampling with replacement also has some important advantages in the context of bagged ensembles. As we will describe in the next section, it can be used as the basis for a simple but effective method for applying early-stopping to bagged ensembles.

Most criticism of early-stopping techniques comes from the statistics community. This is largely because, until recently, no theoretical analyses have been performed that verify in any concrete way the potential of early-stopping as a technique for improving generalisation performance. However, the work of (Wang *et al.*, 1994), which is rarely cited in the statistics literature, provides some very valuable insights. Here it is shown that early-stopping is (ironically) closely related to ridge regression, one of the most popular and successful statistical regularisation methods.

### 3.3 Applying early-stopping to bagged ensembles

As described in section 3.2, training sets in a bagged ensemble are generated by sampling with replacement from the original training set  $T$ . As previously described in section 2.3, the probability a training example from  $T$  will not be part of a bootstrap re-sampled training set is  $(1 - 1/N)^N \approx 0.368$ , where  $N$  is the number of training examples in  $T$ . This means that approximately 37% of the original training examples in  $T$  will not be used for training i.e. they will be out-of-sample. These can be used to estimate generalisation error.

Let us describe this in more concrete terms by outlining a technique that uses these out-of-sample training examples for estimating generalisation error in an early-stopping context. We present the technique in algorithmic form for clarity.

**Step 1:** *Set-up bootstrap training sets*

Generate  $B$  bootstrap re-samples of  $T$ :  $T_1^*, T_2^*, \dots, T_B^*$  where  $T = \{(t_n, \mathbf{x}_n)\}_{n=1}^N$ .

**Step 2:** *Compute generalisation error estimates for each network in the ensemble*

for  $b = 1$  to  $B$

for  $e = 1$  to  $E$

Compute:

$$G_b(e) = \sum_{n=1}^N \gamma_n^b (t_n - \phi(\mathbf{x}_n; \hat{\mathbf{w}}_{T_b^*}(e)))^2 \quad (3.3)$$

where  $\gamma_n^b = 1$  is an indicator variable that indicates whether training sample  $n$  is out-of-sample for training set  $T_b^*$ ;  $\gamma_n^B = 1$  if it is and  $\gamma_n^B = 0$  if it is not. A value for  $E$  that guarantees convergence is chosen as discussed in section 3.2 above. In section 3.5, we discuss what value for  $B$  should be chosen.

**Step 3:** Find the best value for  $e$  for each network in the ensemble

for  $b = 1$  to  $B$

Compute:

$$OPT_b(e) = \underset{e}{\operatorname{argmin}} (G_b(e)) \quad (3.4)$$

Here, for each network in the ensemble, the user finds the value for  $e$  that minimises generalisation error. The corresponding networks are chosen as the optimal set for the ensemble.

Although there is no record of this technique in the neural network literature, it is generally accepted that, in practice, it enjoys widespread use. It is fast, easy to implement and easy to understand. More importantly, it does not require data to be sacrificed for estimating generalisation error. However, as we will discuss in much detail in the next section, it only provides “local” estimates of generalisation error i.e. it does not consider how diversity amongst networks in an ensemble can significantly influence overall ensemble generalisation error. This point was highlighted in chapter 2 – a controlled level of over-fitting can in fact improve overall ensemble generalisation performance by generating more diversity. We support these assertions by experimentation in section 3.5.

### 3.4 The NeuralBAG algorithm

In this section we propose a new early-stopping algorithm we call NeuralBAG that optimizes generalisation performance in bagged neural network ensembles. It operates in a similar fashion to the technique presented in section 3.3 above and is also



presented in algorithmic form for clarity. However, it tunes diversity i.e. it will over-fit/under-fit networks in an ensemble if it estimates that this will improve overall ensemble generalisation performance. An earlier version of NeuralBAG was first introduced in (Carney and Cunningham, 1999a). However this presentation was only an outline of the technique – no concrete justification or evaluation was included.

So how does NeuralBAG tune diversity? The key to solving this problem is to compute estimates of *ensemble generalisation error* rather than *individual network generalisation error*. This issue has not been directly addressed for neural networks – most techniques proposed have concentrated on optimising the networks in an ensemble locally e.g. in (Zhang, 1999) weight-decay is used to regularise the individual networks. Although some techniques do use estimates of ensemble generalisation error e.g. (Heskes, 1997a) they are used for a different purpose and not with the explicit aim of tuning diversity. The details of the NeuralBAG algorithm go as follows.

**Step 1:** *Set-up bootstrap training sets*

Generate  $B$  bootstrap re-samples of  $T : T_1^*, T_2^*, \dots, T_B^*$  where  $T = \{(t_n, \mathbf{x}_n)\}_{n=1}^N$ .

**Step 2:** *Compute ensemble generalisation error estimates for each training example*

for  $n = 1$  to  $N$

for  $e = 1$  to  $E$

Compute:

$$G_n(e) = \left( t_n - \frac{\sum_{b=1}^B \gamma_n^b (\phi(\mathbf{x}_n; \hat{\mathbf{w}}_{T_b^*}(e)))}{\sum_{b=1}^B \gamma_n^b} \right)^2 \quad (3.5)$$

Here we find the networks in the ensemble that were trained with bootstrap resampled training sets that do not contain training example  $n$ . We propagate training example  $n$  through these networks, average (bag) their outputs and calculate the squared error. This error is an estimate of the *ensemble generalisation performance* for a single test example. We repeat this procedure for each training example in  $T$  throughout the specified range of epochs. At the end of this step the user will have ensemble generalisation error estimates for each of the  $N$  training examples in  $T$  for  $e = 1, \dots, E$ .

**Step 3:** *Aggregate the ensemble generalisation error estimates specific to each network in the ensemble*

for  $b = 1$  to  $B$

for  $e = 1$  to  $E$

Compute:

$$G_b(e) = \frac{1}{N_b} \sum_{n=1}^N \gamma_n^b G_n(e) \quad (3.6)$$

where  $N_b = \sum_{n=1}^N \gamma_n^b$  denotes the number of training examples for training set  $T_b^*$ . At the end of this step, for each network in the ensemble, the user will have aggregated ensemble generalisation error estimates throughout the range  $e = (1, \dots, E)$ .

**Step 4:** *Find the best value for  $e$  for each network in the ensemble*

for  $b = 1$  to  $B$

Compute:

$$OPT_b(e) = \underset{e}{\operatorname{argmin}} (G_b(e)) \quad (3.7)$$

Here, for each network in the ensemble, the user finds the value for  $e$  that minimises generalisation error. The corresponding networks are chosen as the optimal set for the ensemble.

As already mentioned, the very important difference of this algorithm is that diversity is tuned. This is achieved by computing estimates of ensemble generalisation error as opposed to estimates of individual network generalisation error. From these we can approximate the optimal number of epochs individual networks should be trained for so that overall ensemble generalisation performance is optimised.

Up to now we have not mentioned the number of hidden units each network in a bagged ensemble should have. In (Carney and Cunningham, 1999a), a different version of NeuralBAG is presented that includes another loop to estimate experimentally a (possibly) different number of hidden units for each network in the ensemble. Over a small range of hidden units this provides modest improvements in performance. However, computationally, a very high price is paid. We suggest that for small problems i.e. problems represented by small training sets and relatively few dimensions/inputs, such an approach should be adopted. However, for larger problems we suggest that the version of NeuralBAG presented in this section should be used. The number of

hidden units can be estimated prior to training using some simpler method (e.g. the technique described in (Baum and Haussler, 1988)) and remain fixed for each network in the ensemble.

## 3.5 Evaluation

In this section we analyse the results of 4 sets of experiments that evaluate the performance and highlight the properties of ensembles trained using NeuralBAG. We use 4 of the financial data-sets summarised in section 1.7; CHF/JPY, CHF/JPY, S&P500 and NYSE. Each foreign exchange rate data-set is arranged into 1230 training vectors that are set-up as described by equation 1.4. The stock market index data-sets are also arranged into 1230 training vectors and are set up as described equation 1.3.

### 3.5.1 Experiment 1: Are NeuralBAG estimates of generalisation error better than local estimates?

In this experiment we compare the generalisation performance of ensembles trained using NeuralBAG to the generalisation performance of ensembles trained using 2 techniques that both use only local estimates of generalisation error to optimise ensemble performance. The first local technique that we compare NeuralBAG to is a benchmark technique that we adapt from (Breiman, 1996b). The second is the simple early-stopping technique described in section 3.3.

Let us first describe how the benchmark technique works. Breiman showed in (Breiman, 1996b) that a good measure of the quality of any generalisation error estimate based on the training set can be found by comparing its accuracy to the estimate one would get using an independent validation set which is the same size as the training set. Adapting this idea to our problem, we train ensembles that have each individual network optimised *locally* using estimates of generalisation error generated from independent validation sets that are the same size as the training sets.



Our reasoning behind this idea goes as follows. If we optimise the individual networks of an ensemble as described above, then we can expect the performance of such an ensemble to be close to the best achievable for an ensemble where only local estimates of generalisation error based on the training set are used, and the importance of tuning diversity is not considered. Therefore, if we compare the generalisation performance of such ensembles to the generalisation performance of ensembles trained using NeuralBAG, any improvement in performance yielded by NeuralBAG can be attributed (mostly) to tuning diversity.

The experiments were set up as follows. We generated 10 randomly sampled training, validation and test sets for each data-set and averaged results across 10 experiments. Randomising the data-sets in this fashion is not normally done for time-series prediction experiments – one usually trains on the past and predicts the future. However, this convention for evaluating time-series models combined with the use of validation sets in this experiment severely limits the statistical significance of results. This is because only one test set for each data-set would be available for evaluating generalisation performance. Treating the process as a standard function approximation task as we do here allows us to generate more test sets and suffices as a method to increase the statistical significance of results.

For all of the data-sets the training, validation and test sets were of size 500, 500 and 230 respectively. Each combination of training, validation and test set for each data-set are independent of each other. Note that the same training and test set combinations are used for ensembles trained using NeuralBAG, the benchmark technique and the simple early-stopping technique so as to get a direct comparison of performance. All ensembles trained had 30 networks (in experiment 4 below we justify this), a learning rate of 0.2 and momentum rate of 0.9. Ensembles trained with the foreign exchange rate data-sets had 7 hidden units and with the stock-market index data-sets 6 hidden units. We used the technique described in (Baum and Hausler, 1988) to estimate these figures. The results are presented in table 3.1 and discussed in section 3.5.5.



Table 3.1: Experiment 1 results. Generalisation (test set) mean-squared error performance of ensembles trained using NeuralBAG (NBAG) compared to the benchmark technique adapted from (Breiman, 1996b) that uses local estimates of generalisation error (B-LOCAL) and the simple technique described in section 3.3 that also uses local estimates of generalisation error (S-LOCAL). Note that the USD/CHF, S&P500 and NYSE results have been adjusted for clarity and ease of comparison. For example, all of the S&P500 results have been multiplied by  $10^{-2}$ .

Dataset	NBAG	stdev	B-LOCAL	stdev	S-LOCAL	stdev
CHF/JPY	1.23	0.07	1.26	0.06	1.26	0.06
USD/JPY	1.98	0.16	1.99	0.14	2.00	0.15
S&P500 ( $\times 10^2$ )	3.71	0.42	3.75	0.41	3.76	0.39
NYSE ( $\times 10$ )	7.62	0.88	7.97	0.80	8.17	0.78

### 3.5.2 Experiment 2: Does NeuralBAG over-fit or under-fit?

In this experiment we compare the average number of epochs or training iterations that each of the 3 techniques (NBAG, B-LOCAL and S-LOCAL) estimate as optimal. The aim is to determine whether NeuralBAG over-fits or under-fits the training sets (relative to the other two techniques) when it tunes diversity. Using estimates recorded from experiment 1, we average the optimal number of epochs estimated for each network in each ensemble. This gives us a single average number of epochs for each ensemble. We then average these figures across the 10 experiments performed for each data-set. Note that we do not include standard deviations with the results. This is because in the context of this experiment they are not meaningful or useful. The results are presented in table 3.2 and discussed in section 3.5.5.

### 3.5.3 Experiment 3: How does ambiguity and generalisation error evolve during training?

In this experiment we attempt to illustrate the importance and value of tuning diversity by plotting the ensemble test set (generalisation) performance (an estimate for  $E$ ) against the corresponding ensemble ambiguity (an estimate for  $\bar{A}$ ) and the average individual network test set performance (an estimate for  $\bar{E}$ ) as training evolves. We

Table 3.2: Experiment 2 results. Average optimal number of epochs estimated for each ensemble and data-set in experiment 1.

Dataset	NBAG	B-LOCAL	S-LOCAL
CHF/JPY	2577	964	1412
USD/JPY	2359	1736	1439
S&P500	2713	2002	1675
NYSE	2820	1935	1665

do this for a single training and test set pair of the CHF/JPY data and the S&P500 data.

Note that we could not directly use NeuralBAG to generate such a plot. This is because it estimates a (possibly) different optimal number of epochs for each network in the ensemble. This makes a single plot that compares these measures at each epoch impossible. To overcome this and to generate a meaningful plot, we recorded the responses generated by each network in the ensemble at each epoch for the test set. Following training we computed the ensemble test set performance, the ensemble ambiguity and the average individual network test set performance at each epoch. The resulting plots are illustrated in figures 3.1 and 3.2 and discussed in section 3.5.5.

#### 3.5.4 Experiment 4: How many networks should the ensembles have?

In this experiment we attempt to investigate the effect the size of the ensemble has on NeuralBAG and how it tunes diversity. Using the 10 CHF/JPY and S&P500 training and test set pairs, we plot the average test set performance across 10 experiments against the number of networks in the ensemble. We also plot the average number of epochs across 10 experiments against the number of networks in the ensemble. The plots are illustrated in figures 3.2 and 3.3 and discussed in section 3.5.3.

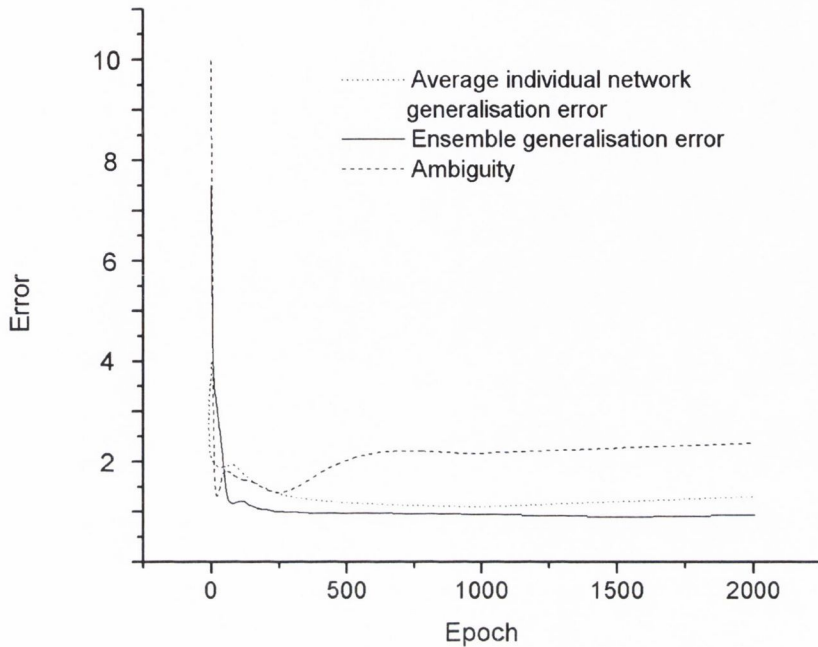


Figure 3.1: Experiment 3 results, CHF/JPY data. Here we compare the ensemble generalisation performance to the ensemble ambiguity and the average individual network generalisation performance as training evolves for a single training and test set pair of the CHF/JPY data.

### 3.5.5 Discussion

In this section we discuss the results of the experiments described above. The results of experiment 1 demonstrate the potential of NeuralBAG as a technique for tuning diversity. On every data-set, it out-performs the benchmark local technique (B-LOCAL). It does this despite having fewer validation samples ( $\approx 63\%$  the number of samples used in benchmark technique per network in each ensemble). As expected it also out-performs the simple local technique (S-LOCAL). The simple local technique has approximately the same number of validation samples per network in each ensemble but does not tune diversity.

One of the key aims of these experiments is to verify that NeuralBAG improves



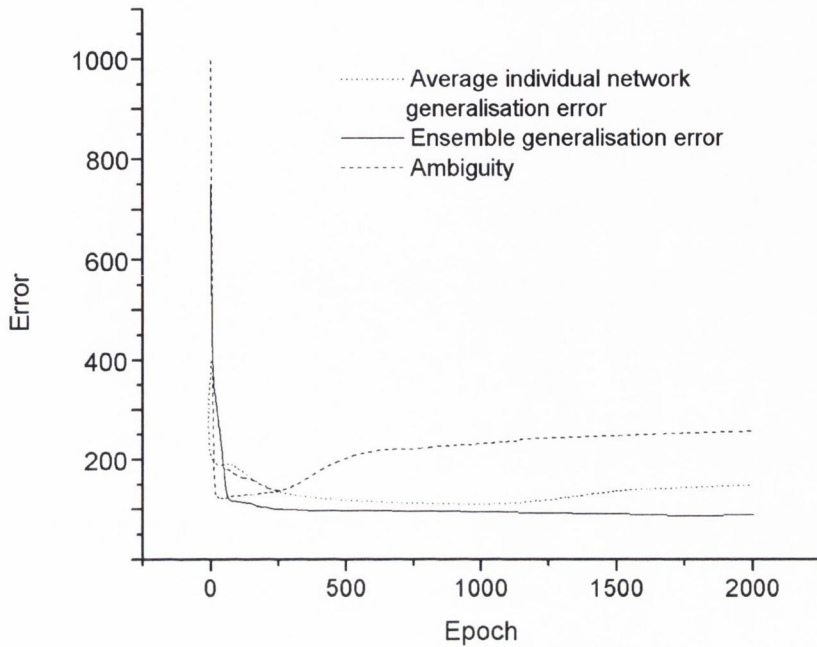


Figure 3.2: Experiment 3 results, S&P500 data. As above for figure 3.1 but with the S&P500 data.

performance by tuning diversity and not by any other means. The results of experiments 2 and 3 help to confirm this. Taking experiment 2 first, the average number of epochs estimated as optimal for NeuralBAG is consistently larger than that for the benchmark technique and the simple technique. This suggests that networks are over-fitting their training sets (relative to the other techniques) in order to tune (which translates to generating more) diversity. It is important to note however that the ensemble generalisation error begins to increase after the optimal number of epochs – it does not converge, instead the ensemble itself begins to over-fit the training set. This is important as it implies that it is not good enough to just train every network in a bagged ensemble to convergence – a *controlled* level of over-fitting is required. The results of experiment 3 illustrate this effect very clearly. For the CHF/JPY data, the average individual network generalisation error is minimised much earlier (epoch

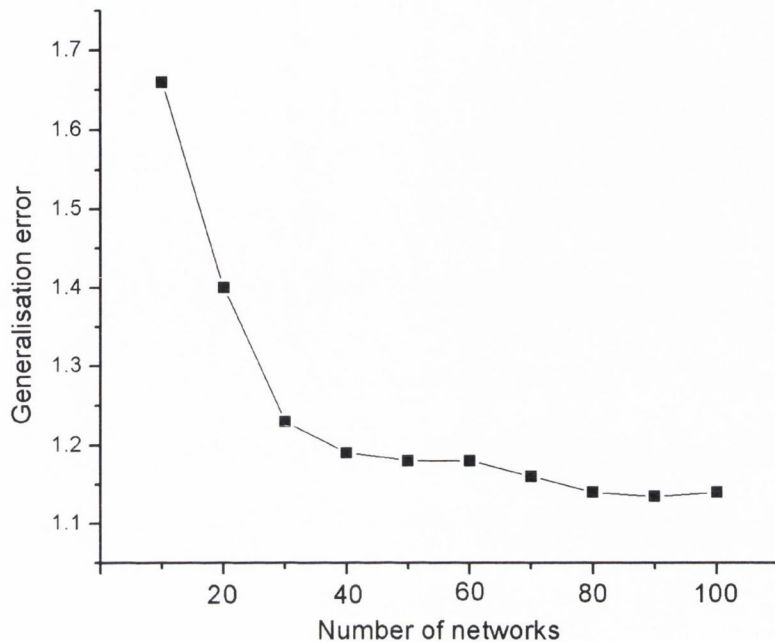


Figure 3.3: Experiment 4 results, CHF/JPY data, part 1. Here we plot the average generalisation error across 10 experiments on the CHF/JPY data as a function of the number of networks in ensembles trained with NeuralBAG.

970) than the ensemble generalisation error (epoch 1590). For the S&P500 data, the average individual network generalisation error is minimised at epoch 1005 and the ensemble generalisation error at epoch 1910. However, for both data-sets the ensemble generalisation error begins to increase again following the ensemble generalisation error minima.

In experiment 4 we attempt to investigate what effect the size of an ensemble has on NeuralBAG. The results for the first part of the experiment are not a surprise – as the size of an ensemble increases, generalisation error decreases. However, little is to be gained by using more than 30 networks and improvements seem to level out at around 100 networks. This is consistent with previous work on bagged ensembles e.g. (Breiman, 1996a) and (Heskes, 1997a). The results on the second part of the experiment are a little more interesting. Here, the average optimal number of epochs

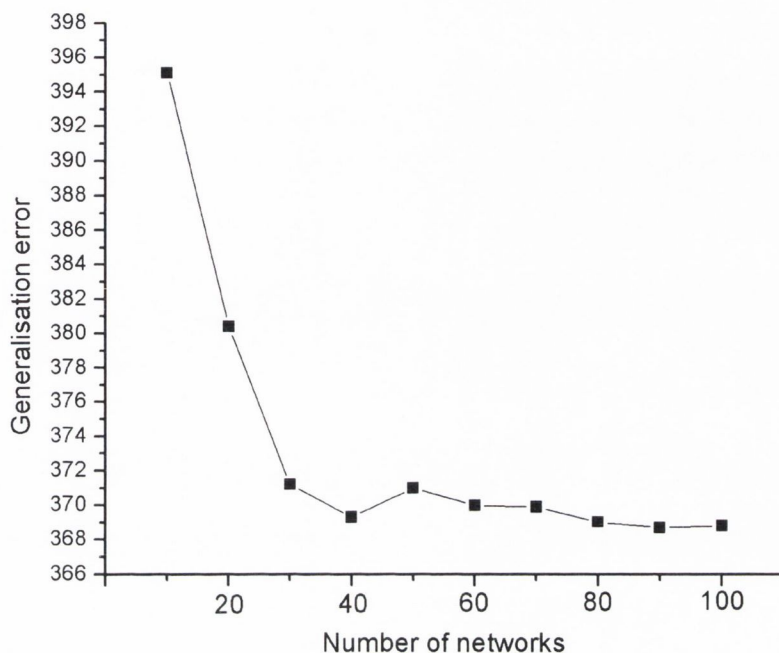


Figure 3.4: Experiment 4 results, S&P500 data, part 1. As above for figure 3.3 but with the S&P500 data.

increases because the variance (and therefore also the diversity) of a larger ensemble will naturally be lower. To compensate for this effect NeuralBAG will over-fit the networks a little more to generate more diversity.

Note that this work is also presented in (Carney and Cunningham, 1999b). However, popular benchmark regression data-sets were used for the experiments instead of the financial data-sets used here.

### 3.6 Relation to other work

In this section we discuss how the work presented in this chapter relates to similar, previous studies on neural network ensembles. Much of the work presented in this chapter was inspired by the the work of Sollich and Krogh (1996). They studied the



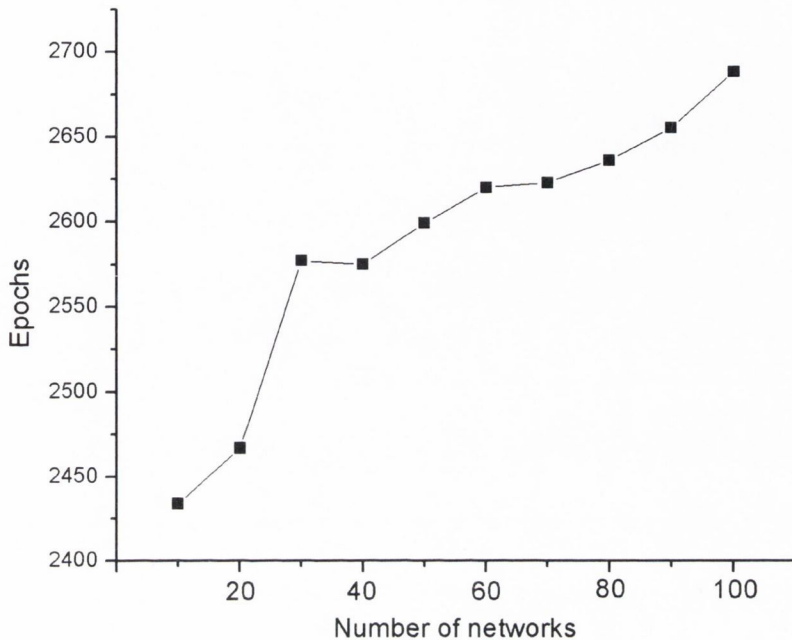


Figure 3.5: Experiment 4 results, CHF/JPY data, part 2. Here we plot the average estimated optimal number of epochs across 10 experiments on the CHF/JPY data as a function of the number of networks in ensembles trained with NeuralBAG.

characteristics and properties of simple linear ensembles and together with Krogh and Vedelsby's work (Krogh and Vedelsby, 1995) were the first to study in depth the importance and value of diversity in neural network ensembles. More specifically, in (Sollich and Krogh, 1996) they proposed that for large *linear* ensembles, under-regularized networks should be used and that a globally optimal ensemble generalisation error can be reached by also varying the training set sizes of the individual networks. If only small ensembles can be trained due to computational restrictions, then they suggested that varying the re-sampling rate of the training sets is unnecessary (especially if the original training set is very noisy) and that a better approach is to use an optimised weighted average to combine the networks in the ensemble.

So, how does their work relate to ours? Although there are a number of obvious similarities in some of the conclusions, there are a number of important differences in

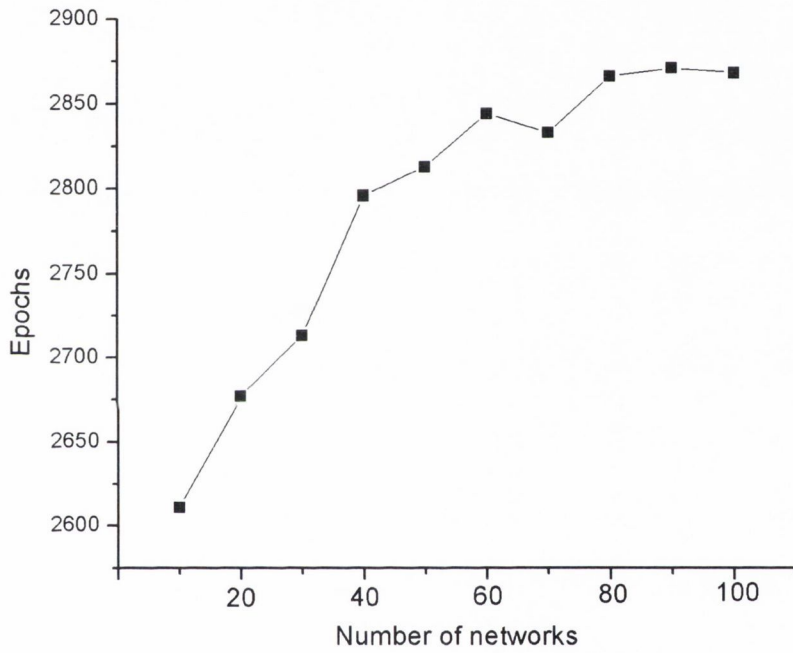


Figure 3.6: Experiment 4 results, S&P500 data, part 2. As above for figure 3.5 but with the S&P500 data.

the analysis and aims of their research. Firstly, our analysis is based on experiments performed using ensembles of *non-linear* back-propagation neural networks trained using noisy, real-world data. Secondly, we use an early-stopping technique to regularise the networks, they use weight-decay. Thirdly, we are only interested in bagged ensembles – we do not attempt or desire to vary the re-sampling rate of the bootstrapping process. As discussed in section 2.4, we believe that this can potentially be cumbersome to optimise and may cause difficulties when confidence and prediction intervals are estimated. Finally, we don't attempt to use a weighted average – we find the robustness of a simple average very attractive. We prefer to simply increase the size of the ensemble to compensate for any problems caused by poorly performing individual networks. Given the wide availability of significant computer processing power today, we believe this approach is justified. Also, as described in (Carney *et al.*, 1999) and chapter 4, an added advantage of this approach is that large ensembles can produce more accurate confidence and prediction intervals.

Other related work includes Husmeier's analysis of *RVFL* ensembles (Husmeier, 1999). He was the first to provide empirical evidence that over-fitting an ensemble of non-linear networks can be useful. However, his focus is solely on *RVFL* ensembles whose properties are significantly different to conventional back-propagation ensembles. For example, in (Husmeier, 1999) he showed how bagging does not usually improve the performance *RVFL* networks. He suggests that enough inherent instability exists in *RVFL* networks and introducing more by re-sampling the training sets will not improve performance. Also, his focus is a little different from ours in that he doesn't explicitly attempt to tune diversity.

Finally, the work of Heskes also has some similarities with ours. In (Heskes, 1997a), he develops and evaluates a technique for optimising the performance of weighted back-propagation ensembles. Although he does compute estimates of ensemble generalisation error he uses them for a different purpose and does not explicitly attempt to tune diversity.



## 3.7 Summary

In this chapter we demonstrated the importance and value of tuning diversity in bagged neural network ensembles. Our approach of tuning diversity by varying the fit of the networks is a simple idea, but nevertheless works very well for bagged ensembles in practice. Also, it has one key advantage over other approaches – diversity tuning and network parameter (weights) tuning are unified – both can be performed simultaneously using the same algorithm. This would not be easy to achieve if a different method (e.g. varying the re-sampling rate of the training set) were used to tune diversity.

Also, an important finding of our work that must be stressed is that although some over-fitting amongst networks in a bagged ensemble is usually required, it must be controlled. If it is not, even large ensembles will over-fit and ensemble generalisation performance will be compromised. Nevertheless however, it is fascinating to observe over-fitting improving the generalisation performance of neural networks for a change.

# Chapter 4

## Predicting uncertainty

### 4.1 Introduction

In chapter 3 we proposed a new technique for optimising the generalisation performance of bagged neural network ensembles. However, the predictions generated by these ensembles are point predictions i.e. they do not include any measure of prediction uncertainty. In reality, the quality of these predictions can vary significantly. For example, there may be a large amount of unpredictable noise in the test data caused by high volatility or other extreme market events. Given the importance placed on managing risk in financial institutions today point predictions are therefore of little practical value. In this chapter we develop a new technique that generates interval predictions that provide a valuable insight into the ensemble models and market behaviour.

In section 4.2 we introduce the underlying theory of interval prediction by describing how uncertainty in regression is represented and outlining the difference between confidence and prediction intervals. In section 4.3 we propose a new method for computing confidence intervals for neural network ensembles and illustrate the performance gains it produces over previous techniques using a popular synthetic data-set. In section 4.4 we show how the confidence intervals can be combined with econometric estimates of future volatility movements to generate prediction intervals. Our approach is novel and in chapter 5 is shown to provide excellent interval predictions

of financial market movements over a number of prediction horizons.

## 4.2 Uncertainty in regression

In this section we introduce an analytical framework through which uncertainty in regression can be described. Let us assume we are given a set of  $N$  training pairs  $\{(t_n, \mathbf{x}_n)\}_{n=1}^N$ , generated according to

$$t = f(\mathbf{x}) + \epsilon(\mathbf{x}) \quad (4.1)$$

where  $t$  is the observed target value,  $f(\mathbf{x})$  is the true regression and  $\epsilon(\mathbf{x})$  is noise with zero mean. When we train a neural network on such data, our aim is for the network to approximate the true regression  $f(\mathbf{x})$ . Using the notation introduced in section 2.3 let us denote this neural network approximation as  $\phi(\mathbf{x})$ , which can be interpreted as an estimate of the mean of the distribution of target values given an input vector  $\mathbf{x}$ . For many real-world regression applications, it is highly desirable to have some measure of confidence in this point prediction.

There are two “components” of confidence however. The first is concerned with the accuracy of our estimate of the true regression i.e. the distribution of the quantity  $f(\mathbf{x}) - \phi(\mathbf{x})$ . This distribution is a conditional distribution and in statistics is normally expressed as  $P(f(\mathbf{x})|\phi(\mathbf{x}))$ . In a regression context, measures of confidence based on this distribution are usually termed *confidence intervals* (Heskes, 1997b). The second component of confidence is concerned with our prediction of the targets themselves i.e. the distribution of the quantity  $t - \phi(\mathbf{x})$  or  $P(t|\phi(\mathbf{x}))$ . These estimates are usually termed *prediction intervals* (Heskes, 1997b). As illustrated below in equation (4.2), a confidence interval is enclosed in a prediction interval

$$t - \phi(\mathbf{x}) = [f(\mathbf{x}) - \phi(\mathbf{x})] + \epsilon(\mathbf{x}). \quad (4.2)$$

Prediction intervals are of more practical use than confidence intervals for real-world (especially financial) applications. This is because prediction intervals are concerned with the accuracy with which we can predict the targets or observed values



themselves, not just the accuracy of our prediction of the true regression.

### 4.3 Confidence intervals

As described in section 4.2 above, confidence intervals are enclosed in prediction intervals and are concerned with the accuracy of our estimate of the true regression i.e. the distribution of the quantity  $f(\mathbf{x}) - \phi(\mathbf{x})$  or  $P(f(\mathbf{x})|\phi(\mathbf{x}))$ . When we use a bagged neural network ensemble, we are interested in the distribution of the quantity  $f(\mathbf{x}) - \phi_{bag}(\mathbf{x})$  or  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ , where  $\phi_{bag}(\mathbf{x})$  denotes the bagged ensemble prediction as described in section 2.3. In this section we show how confidence intervals can be computed for the estimate  $\phi_{bag}(\mathbf{x})$ .

#### 4.3.1 Theory

To generate the confidence (and prediction) intervals we must assume our neural networks provide unbiased estimates of the true regression  $f(\mathbf{x})$ . In other words (for confidence intervals) we must assume the distribution  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$  is centered on the estimate  $\phi_{bag}(\mathbf{x})$ . This assumption, of course, does not hold in practice – as with any other estimator, neural networks can and usually are biased i.e. residual errors are not caused by variance alone. However, it is generally accepted that the variance component of residual error in neural network learning dominates the bias component – see (Geman et al., 1992) for a comprehensive study of this issue.

To form our confidence intervals, we need to estimate the variance of the distribution  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ . However, we have no direct access to this distribution and (for real-world tasks) do not know what the true regression  $f(\mathbf{x})$  is. Using the outputs of the bootstrap re-sampled networks in the ensemble, we can however approximate it. The bootstrap outputs provide us with an empirical estimate of the distribution  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$  which is the “inverse” of the distribution  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ . This empirical estimate of  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$  is  $P(\phi(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ . Here,  $\phi_{bag}(\mathbf{x})$  replaces the true regression to which we have no access. The variance of this distribution can be found by calculating the variance across the bootstrap outputs. Note that by assuming

$P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$  is Gaussian we also assume its inverse  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$  is Gaussian and so any estimates of variance for  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$  can be used as estimates of variance for  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ . This gives us

$$\sigma^2(\mathbf{x}) = \frac{1}{B-1} \sum_{b=1}^B (\phi_b(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2 \quad (4.3)$$

In (Heskes, 1997b), this variance measure is used to construct standard Gaussian confidence intervals for weighted neural network bootstrap ensemble predictions.

This variance estimate however will be biased upwards for most predictions. This is because it more accurately reflects the variance of the distribution  $P(f(\mathbf{x})|\phi(\mathbf{x}))$ , not  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ . In other words, it really only provides a variance measure suitable for computing a confidence interval for a single network prediction  $\phi(\mathbf{x})$ . As discussed in chapter 2 bagging has the effect of significantly reducing the variance of neural networks. This reduced variance should be reflected in the confidence interval. We will now propose a way to do this.

Using a large number of bootstrap networks for the ensemble (we use 200, but fewer would suffice) we divide the ensemble into  $M$  smaller ensembles (we use 8 groups of 25 networks each). This gives us a set of  $M$   $\phi_{bag}(\mathbf{x})$  values

$$\zeta = \{\phi_{bag}^i(\mathbf{x})\}_{i=1}^M \quad (4.4)$$

from which we approximate a more accurate variance measure for the distribution  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$ . However, we don't simply compute a variance measure across (in our case) the 8 ensemble outputs - this variance measure itself would be highly variable and unreliable. Instead we undergo one more iteration of bootstrapping and use the technique for which it was originally designed as described in section 2.2. We form  $P$  (we use  $P = 1000$ ) bootstrap re-sampled sets of  $\zeta$

$$\Upsilon = \{\zeta_j^*\}_{j=1}^P \quad (4.5)$$

where

$$\zeta_j^* = \{\phi_{bag}^{j_1^*}(\mathbf{x}), \dots, \phi_{bag}^{j_M^*}(\mathbf{x})\} \quad (4.6)$$

each containing  $M$   $\phi_{bag}(\mathbf{x})$  values sampled with replacement from  $\zeta$ . We calculate a variance measure for each of these sets and then calculate an average of these to provide a smoother, lower variance estimate of the variance of the distribution  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$ :

$$\sigma_S^2(\mathbf{x}) = \frac{1}{P} \sum_{j=1}^P \sigma_j^{2*}(\mathbf{x}) \quad (4.7)$$

where

$$\sigma_j^{2*}(\mathbf{x}) = \frac{1}{M} \sum_{k=1}^M (\phi_{bag}^{j_k^*}(\mathbf{x}) - \phi_{avg}^j(\mathbf{x}))^2 \quad (4.8)$$

and

$$\phi_{avg}^j(\mathbf{x}) = \frac{1}{M} \sum_{k=1}^M \phi_{bag}^{j_k^*}(\mathbf{x}). \quad (4.9)$$

Note that here, to estimate the true regression, we use the combination of values across all the  $B$  networks in the ensemble and denote this as  $\phi_{BAG}(\mathbf{x})$ . In other words, we approximate the distribution  $P(\phi_{bag}(\mathbf{x})|f(\mathbf{x}))$  from  $P(\phi_{bag}(\mathbf{x})|\phi_{BAG}(\mathbf{x}))$ . This second iteration of bootstrapping is not computationally intensive (there are no networks to train) and is easy to implement.

Now that we have a good variance estimate for the distribution  $P(f(\mathbf{x})|\phi_{bag}(\mathbf{x}))$ , which we assume to be Gaussian, we can calculate a confidence interval in the usual fashion

$$\phi_{BAG}(\mathbf{x}) - z^{(1-\alpha)}\sigma_S(\mathbf{x}) \leq f(\mathbf{x}) \leq \phi_{BAG}(\mathbf{x}) + z^{(1-\alpha)}\sigma_S(\mathbf{x}) \quad (4.10)$$

The factor  $z^{(1-\alpha)}$  depends on the desired level of confidence ((e.g.) 90%, 95% etc.) and can be taken from a standard Gaussian distribution table.

It should be noted that this technique can be applied to any neural network ensemble technique that uses the bootstrap to generate the training sets for the ensemble e.g. balancing (Heskes, 1997a) which uses a weighted average to combine the ensemble outputs. See (Carney *et al.*, 1999) for a more general description of the technique.



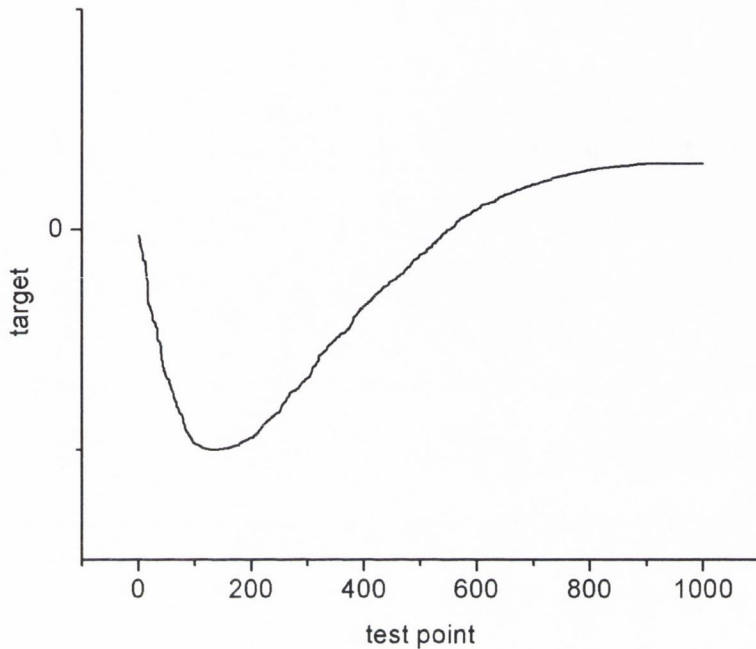


Figure 4.1: Synthetic test data generated using Wahba's function.

### 4.3.2 Illustration

In this section we attempt to illustrate the properties and performance of our confidence interval technique by testing it on synthetic data. Our motivation behind using synthetic data here is to maintain some statistical transparency and flexibility e.g. we can exclude any noise and estimate how biased our ensembles are by comparing their prediction to the true regression. Note that this section is merely an illustration of the technique. A complete evaluation using financial data will be provided in chapter 5.

The synthetic data-set that we used is generated as follows. Inputs are uniformly drawn at random from the interval  $[0,2]$ . Here, the input vectors contain only a single value. The target values are generated according to

$$t = 4.26(e^{-x} - 4e^{-2x} + 3e^{-3x}) \quad (4.11)$$

Table 4.1: Illustration of confidence interval performance expressed in terms of interval non-coverage i.e. number of times the actual target value was not covered by the interval (see section 5.2.5 for an exact description of this metric). Given that we used 1000 test points for each experiment we would expect non-coverage of 200 for an 80% interval for example.

INTERVAL	NEW	SIMPLE	IDEAL
80%	220	35	200
90%	112	0	100
95%	58	0	50
99%	13	0	10

This function was first introduced in (Wahba and Wold, 1975) and is known as the Wahba's function. We generated 1000 data pairs for training and 1000 for testing. The test set is illustrated in figure 4.1.

For our experiments, 200 networks were trained for the bagged ensemble. We ran the test set through our ensemble following the steps outlined in section 4.3.1 above to produce 80%, 90%, 95% and 99% confidence intervals for each test point. We repeated the experiment using the simpler technique described in (Heskes, 1997b). As illustrated by the results which are summarised in table 4.1, the simpler technique produces intervals that are consistently too wide for ensemble predictions and provide no non-coverage for the 90%, 95% and 99% intervals.

## 4.4 Prediction intervals

As described above in section 4.2 prediction intervals are concerned with the distribution of the quantity  $t - \phi_{bag}(\mathbf{x})$  or  $P(t|\phi_{bag}(\mathbf{x}))$  i.e. they estimate the accuracy with which we can predict the targets or observed values themselves. To estimate prediction intervals, we must incorporate the noise variance factor of a regression. This is the variance of the noise component  $\epsilon(\mathbf{x})$  in equations 4.1 and 4.2. Incorporating this factor, the variance of the complete regression can be given as

$$s^2(\mathbf{x}) = \langle (t - \phi_{BAG}(\mathbf{x}))^2 \rangle = \langle (f(\mathbf{x}) - \phi_{BAG}(\mathbf{x}))^2 \rangle + \langle \epsilon^2(\mathbf{x}) \rangle \quad (4.12)$$

where  $\langle \cdot \rangle$  denotes expectation. Note that we already have an estimate of the model variance from our bootstrap technique described in section 4.3,

$$\langle (f(\mathbf{x}) - \phi_{BAG}(\mathbf{x}))^2 \rangle = \sigma_s^2(\mathbf{x}). \quad (4.13)$$

In this section we will develop a new technique to model the noise variance so that  $s^2(\mathbf{x})$  can be estimated and prediction intervals generated.

The key to solving this problem is recognising that noise variance in regression is equivalent to market volatility in econometrics. This is an important connection to make – volatility has been shown to be very predictable given the clustered nature of its behaviour i.e. clusters of high volatility are followed by clusters of low volatility (see (Alexander, 1998 (chapter 4) for a discussion and some illustrations of this effect). Using this, we propose to use an established econometric volatility prediction technique to generate predictions of volatility and combine these with predictions of the model variance (estimated using the bootstrap as in section 4.3) to generate prediction intervals.

Before we illustrate this equivalence of noise variance and volatility and show how it can be used to generate accurate prediction intervals, we will first describe how volatility is estimated and predicted in econometrics.

#### 4.4.1 Volatility

The volatility of a financial market is a measure of how turbulent it is i.e. a measure of how much prices “jump about”. Given this, volatility cannot be observed and so must be estimated. Traditionally, the *n-period historic volatility* (HV) estimate has been used to do this. The HV estimate at time  $T$  is usually expressed in terms of an annualised percentage standard deviation<sup>1</sup> which we will denote as

$$(100\hat{\nu}_T\sqrt{A})\% \quad (4.14)$$

---

<sup>1</sup>This is done to standardise the volatility estimates so that volatilities of different maturities may be compared on the same scale – see (Alexander, 1998 (chapter 4)).



where  $A$  denotes the number of observations per year and

$$\hat{v}_T^2 = \sum_{i=T-n}^{i=T-1} r_i^2/n \quad (4.15)$$

Note that here  $r_i$  denotes the log-return at time  $i$  and  $n$  the period. This technique for estimating volatility has been replaced by more sophisticated *general autoregressive conditional heteroskedastic* (GARCH) and *exponentially weighted moving average* (EWMA)<sup>2</sup> methods in most financial institutions today. The EWMA technique has been popularised by its use for generating the RiskMetrics volatility datasets and is generally accepted as the industry standard for volatility estimation (and prediction) today (Alexander, 1998 (chapter 4)).

The EWMA approach differs from the HV approach in that it places a higher weight on more recent observations in the calculation for  $\hat{v}_T$ . This approach has two important advantages over the HV approach. The first is that the volatility estimate reacts faster to an abrupt change in the market i.e. a very large return. The second is that following a shock the estimate of volatility declines exponentially as the market reverts to normal behaviour and the weight of the shock observation falls. HV methods on the other hand introduce what are known as “ghost features” – the effect of a shock can be reflected in the volatility estimate long after the market returns to normal behaviour. For an EWMA volatility estimate at time  $T$ ,

$$\hat{v}_T = \sqrt{(1 - \lambda) \sum_{i=1}^{\infty} \lambda^{i-1} r_{T-i}^2} \quad (4.16)$$

which can be re-written in recursive form as

$$\hat{v}_T = \sqrt{(1 - \lambda)r_{T-1}^2 + \lambda\hat{v}_{T-1}^2}. \quad (4.17)$$

Here it is assumed that we have access to an infinite set of returns<sup>3</sup>. This recursive

<sup>2</sup>An EWMA is equivalent to an integrated GARCH (I-GARCH) without a constant term (Alexander, 1998 (chapter 4); Zangari, 1996).

<sup>3</sup>In reality we never have access to an infinite set of returns and so the EWMA is usually “seeded” using a simple squared log-return for example.

form makes it convenient to use the EWMA technique for *predicting* volatility 1-step (e.g. day) ahead,

$$\hat{v}_{T+1} = \sqrt{(1 - \lambda)r_T^2 + \lambda\hat{v}_T^2}. \quad (4.18)$$

Zangari (1996) shows how this can be easily adapted to conveniently generate multiple-step ahead predictions of volatility by using a simple multiple of a 1-step ahead prediction,

$$\hat{v}_{T+h}^2 = h\hat{v}_{T+1}^2 \quad (4.19)$$

or

$$\hat{v}_{T+h} = \sqrt{h}\hat{v}_{T+1}. \quad (4.20)$$

Here  $h$  denotes the prediction horizon e.g.  $h = 5$  will generate a prediction 5-steps ahead.

#### 4.4.2 Estimating the decay factor

The obvious drawback of using EWMA methods over HV methods is that the decay factor  $\lambda$  must be estimated for each data-set and volatility horizon (e.g. 1-day volatility, weekly volatility etc.). We use the simple cross-validation technique outlined in (Zangari, 1996) (which he calls the root mean squared error (RMSE) criterion) to do this. This technique uses a *variance prediction error* which is defined as

$$e_{T+1} = r_{T+1}^2 - \hat{v}_{T+1}^2. \quad (4.21)$$

See (Zangari, 1996) for an analytical justification for the use of this error.

Using this, the root mean squared prediction error is defined as

$$RMSE = \sqrt{\frac{1}{H} \sum_{T=1}^H (r_{T+1}^2 - \hat{v}_{T+1}^2(\lambda))^2}. \quad (4.22)$$

Here the variance prediction  $\hat{v}_{T+1}^2$  is written as a function of  $\lambda$ . This RMSE is computed for a range of  $\lambda$  values and the value that generates the minimum RMSE is chosen as optimal for the data-set in question. Note that  $H$  denotes the number of

Table 4.2: Estimates for the decay factor  $\lambda$  for each data-set used in this thesis. We use 11 years of observations for the stock-market experiments ( $H = 2780$ ) and 5 years of observations for the foreign exchange experiments ( $H = 1238$ ). We tested for 18 values of  $\lambda$  (0.1, 0.15, ..., 0.9, 0.95) for each volatility horizon.

DATASET	1-day	5-day	10-day	20-day
Coca-Cola	0.93	0.20	0.10	0.10
GEC	0.93	0.25	0.15	0.10
IBM	0.98	0.10	0.10	0.10
Microsoft	0.95	0.13	0.10	0.10
CHF/JPY	0.98	0.30	0.20	0.15
USD/JPY	0.98	0.30	0.25	0.15
S&P500	0.92	0.20	0.10	0.10
NYSE	0.93	0.15	0.10	0.10

observations that are used to compute the RMSE.

Unlike the RiskMetrics methodology outlined in (Zangari, 1996) which estimates global estimates for  $\lambda$  for each volatility horizon (e.g.  $\lambda = 0.94$  for all daily market data,  $\lambda = 0.97$  for all monthly market data) we estimate a different  $\lambda$  for each data-set and volatility horizon used. The results are summarised in table 4.2.

### 4.4.3 Combining volatility with model variance

In this section we will illustrate the equivalence of noise variance and volatility. We also outline how we combine the model variance with the volatility estimates to generate prediction intervals.

Consider the following standard econometric model of financial returns (taken from (Alexander, 1998 (chapter 4))),

$$r_T = C + \epsilon_T. \quad (4.23)$$

Here  $r_T$  is the return at time  $T$ ,  $C$  is a constant and  $\epsilon_T$  is the residual error or noise at time  $T$ . Note that  $\epsilon_T$  is assumed to be normally distributed with variance  $\nu_T^2$ . In econometrics this noise variance is called the *volatility*.

If we build a more sophisticated model of financial returns e.g. using a neural



network ensemble, this noise variance (volatility) term does not change. However, it is usually expressed in a different form e.g. as a function of the input vector  $\mathbf{x}$  as in equation (4.12). Therefore, the only hurdle to overcome if we want to use an econometric volatility prediction technique to estimate noise variance is notational – we need to somehow connect the econometric notation with the functional (regression) notation.

To do this, we simply include the subscript  $T$  to denote the target that the volatility estimate  $\hat{\nu}_T$  corresponds to. More specifically

$$s^2(\mathbf{x}) = \langle (t_T - \phi_{BAG}(\mathbf{x}))^2 \rangle = \langle (f(\mathbf{x}) - \phi_{BAG}(\mathbf{x}))^2 \rangle + \langle \epsilon^2(\mathbf{x}) \rangle = \sigma_S^2(\mathbf{x}) + \hat{\nu}_T^2. \quad (4.24)$$

Here the EWMA technique outlined in sections 4.4.1 and 4.4.2 is used to estimate  $\hat{\nu}_T^2$ . To generate a prediction interval we use

$$\phi_{BAG}(\mathbf{x}) - z^{(1-\alpha)}s(\mathbf{x}) \leq t_T \leq \phi_{BAG}(\mathbf{x}) + z^{(1-\alpha)}s(\mathbf{x}) \quad (4.25)$$

Again the factor  $z^{(1-\alpha)}$  depends on the desired level of confidence and can be taken from a standard Gaussian distribution table.

## 4.5 Summary

In this chapter we proposed new techniques for generating confidence and prediction intervals for bagged neural network ensembles used for financial time-series prediction. A unique feature of the prediction interval technique is that it relies on techniques from a number of disciplines; statistics (the bootstrap), econometrics (the EWMA technique) and machine learning (bagged ensembles). This multi-disciplinary approach is key to the technique's success. In the next chapter we will empirically evaluate all of the techniques proposed in this chapter.

# Chapter 5

## Evaluation

### 5.1 Introduction

In this chapter we evaluate the ability of our ensembles to predict future prices, directional change and prediction uncertainty across the 8 financial time-series datasets (described in section 1.7).

We begin in section 5.2 by describing the statistical metrics used to evaluate prediction performance. In this section we also attempt to justify why a revenue (profit) based evaluation method was not used. In section 5.3 we outline the experimental set-up of the experiments and in section 5.4 analyse the results of the experiments in detail. One of our main aims in section 5.4 is to highlight important recurring features in the results. As we shall illustrate and discuss in detail, the most consistent recurring feature in the results is the dependency of the predictions (of price and directional change) on the magnitude of volatility – any poor results correspond to periods of excessive volatility, any excellent results to periods of low volatility. Another important feature related to this is the quality and stability of the prediction intervals even over such periods of excessive volatility – as we shall show, this has significant implications for risk management in real-world trading scenarios.

Overall, the results of this chapter clearly illustrate the potential of the techniques proposed in earlier chapters for predicting future financial market behaviour.

## 5.2 Evaluation metrics

In this section the metrics used to measure the relative and absolute prediction performance of ensembles trained using our techniques are described. Most of them are well known; they include the *root mean squared error (RMSE)*, the *correlation coefficient (CC)*, the *information coefficient (IC)* (also known as the *t*-test or Theil's coefficient of inequality) and the *d-statistic (DC)* (used to measure directional change prediction accuracy). We also describe how we evaluate the accuracy of the prediction intervals (and therefore also implicitly the confidence intervals) by measuring the *non-coverage (NC)* of the intervals over the test sets.

Note that in this thesis we do not attempt to evaluate the performance of the ensembles by tracking profits earned during simulated trading sessions or some other revenue based method. This is because our techniques generate predictions, not trading (e.g. buy or sell) signals. Using predictions to generate such signals requires expertise that is beyond the scope of this thesis and, in any case, is sensitive to a large number of situation specific variables e.g. transaction costs, gearing (percentage of money borrowed to make the trade), market liquidity, portfolio value-at-risk and so on. It is generally accepted today that to effectively evaluate a prediction system using revenue based techniques, actual trades with real money should be executed. However, this is not usually an option for academic research – using objective statistical measures of prediction performance such as those described in this section is perhaps the best alternative.

### 5.2.1 Root mean squared error

The *RMSE* is probably the most widely used measure of estimator performance in times-series prediction and econometrics. We have already used it in section 4.4.2 for parameter tuning. Here we use it to measure prediction accuracy in absolute terms across a test set of size  $N$ ,

$$RMSE = \sqrt{\sum_{n=1}^N (t_n - y_n)^2} \quad (5.1)$$



where  $y_n$  is the predicted value and  $t_n$  the target value.

### 5.2.2 Correlation coefficient

The  $CC$  is another popular measure of absolute prediction accuracy. It measures the linear correlation between predicted values ( $y_n$ ) and actual values ( $t_n$ ), averaged over all observations,

$$CC = \frac{\sum_{n=1}^N (t_n - \bar{t})(y_n - \bar{y})}{\sqrt{\sum_{n=1}^N (t_n - \bar{t})^2} \sqrt{\sum_{n=1}^N (y_n - \bar{y})^2}} \quad (5.2)$$

where  $\bar{y} = 1/n \sum_{n=1}^N y_n$  and  $\bar{t} = 1/n \sum_{n=1}^N t_n$ .  $CC^2 = 1$  denotes perfect correlation between actual and predicted values and  $CC^2 = 0$  signifies no correlation.

### 5.2.3 Information coefficient

It is very important to compare the performance of the ensembles against the performance of trivial predictors. The  $IC$  gives a good measure of the ensemble prediction performance relative to the *martingale* random walk model (see section 1.2),

$$IC = \frac{\sqrt{\sum_{n=1}^N (t_n - y_n)^2}}{\sqrt{\sum_{n=1}^N (t_n - t_{n-1})^2}}. \quad (5.3)$$

For  $IC > 1$ , the neural network is worse than the martingale; for  $IC < 1$  it is better than the martingale. As  $IC$  approaches zero the ensemble is doing infinitely better than the martingale.

This equation can be easily adjusted to estimate the information coefficient for predictions more than one day ahead. For example if we wanted to compare the accuracy of the ensemble against the performance of the martingale for 5-day ahead predictions we would use the following,

$$IC = \frac{\sqrt{\sum_{n=1}^N (t_n - y_n)^2}}{\sqrt{\sum_{n=1}^N (t_n - t_{n-5})^2}}. \quad (5.4)$$

### 5.2.4 Directional change

Although predicting the levels of price changes is desirable, in many cases the direction of the change is equally important. To estimate the directional prediction accuracy we use the following test statistic,

$$DC = \frac{1}{N} \sum_{n=1}^N d_n \quad (5.5)$$

where  $d_n = 1$  if  $(t_n - t_{n-1})(y_n - t_{n-1}) > 0$  and  $d_n = 0$  otherwise (for 1-day ahead predictions). Any estimator with  $DC > 0.5$  is doing better than tossing a coin,  $DC = 1$  implies the estimator is predicting 100% of the directional changes and  $d = 0$ , 0% of the directional changes.

### 5.2.5 Interval non-coverage

The  $NC$  is a measure of the number of times the actual target value is not covered by or falls outside an interval. We use the following statistic to measure this,

$$NC = \frac{1}{N} \sum_{n=1}^N c_n \quad (5.6)$$

where  $c_n = 1$  if  $(t_n - l_n)(u_n - t_n) > 0$  and  $c_n = 0$  otherwise. Here  $l_n$  denotes the prediction for the lower bound of an interval and  $u_n$  denotes the prediction for the upper bound of an interval. If  $c_n = 0.1$  for example, then 10% of the targets are not covered by the intervals – this would be a perfect result for a 90% interval.

## 5.3 Experimental set-up

In this section we describe the experimental set-up of the experiments used to evaluate the predictive ability of the ensembles. For each data-set we perform experiments over 4 different prediction horizons; 1-day, 5-days, 10-days and 20-days ahead<sup>1</sup>. Our

---

<sup>1</sup>A 5-day prediction horizon for example spans a week of trading – if today is Monday then a 5-day ahead prediction aims to predict the closing price of the following Monday.

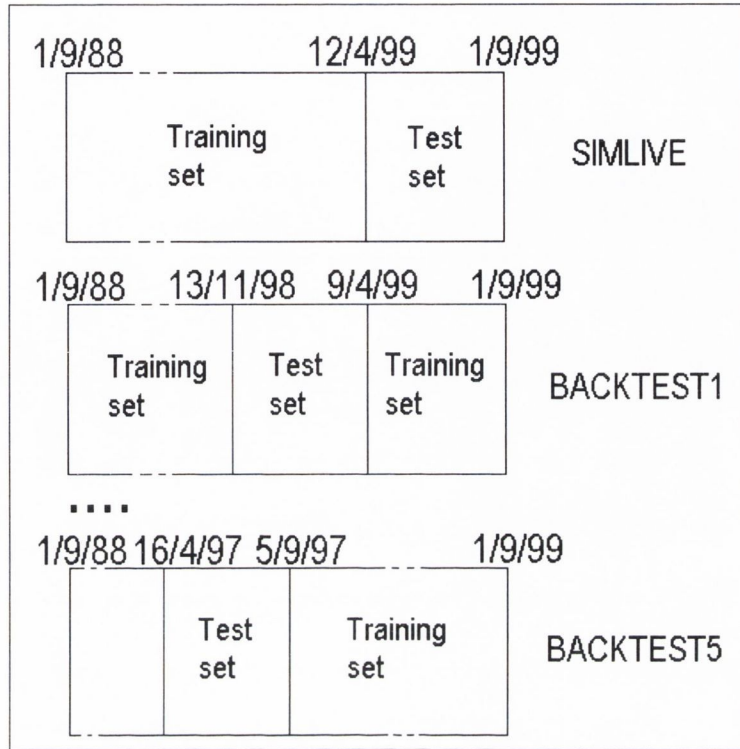


Figure 5.1: Training and test set organisation for the the BACKTEST and SIMLIVE experiments (1-day ahead). Each data-set used has identical training and test set organisations. However the dates will obviously differ amongst the different prediction horizons.

motivation for using these prediction horizons is that they are typical in real-world trading scenarios, particularly for options and forwards trades. We perform two types of experiment for each data-set also. The first, which we call “simulated live” (SIMLIVE), uses 100 test vectors (approximately 20 weeks of trading) which were not used in any part of the training process i.e. for building the ensemble or estimating the decay factor for the EWMA volatility model. Therefore the test set is completely new to all of the models used to generate the predictions and so simulates a live trading scenario. The SIMLIVE experiments are classical time-series experiments – the test sets are made up of the most recent observations in the time-series and so we train on the past and test on the future.



The second, called “back-testing” (BACKTEST) also uses 100 test vectors. However we perform 5 BACKTEST experiments for each data-set and time horizon. The BACKTEST experiments are used to increase the statistical significance of the results – the test sets here consist of contiguous sets of vectors that may occur before some training vectors in the time series. Although these test vectors will not have been used for training the ensembles, they will have been used to estimate the decay factor for the EWMA volatility model. Figure 5.1 illustrates this set-up.

### 5.3.1 Stocks

As described in section 1.7.1 we test the prediction performance of the ensembles on 11 years (1/9/88–1/9/99) of 4 stock-market data-sets; General Electric Corporation (GEC), Coca-Cola, Microsoft Corporation and International Business Machines (IBM). The training vector set-up for each data-set is the same.

Using the notation introduced in section 1.7.1 but a more detailed expression for the training vector, the 1-day ahead vector set-up is

$$r_{1day}, vl_{1day}, vt_{1day}, id_{1day}, dt_{1day}, r_{t+1} \quad (5.7)$$

where

$$r_{1day} = (r_{t-4}, r_{t-3}, r_{t-2}, r_{t-1}, r_t), \quad (5.8)$$

$$vl_{1day} = (vl_{t-4}, vl_{t-3}, vl_{t-2}, vl_{t-1}, vl_t), \quad (5.9)$$

$$vt_{1day} = (vt_{t-4}, vt_{t-3}, vt_{t-2}, vt_{t-1}, vt_t), \quad (5.10)$$

$$id_{1day} = (sp_t, ny_t, dj_t), \quad (5.11)$$

and

$$dt_{1day} = (d_t, m_t, wd_t). \quad (5.12)$$

Here we use a more elaborate expression for the 1-day ahead training vector set-up than that used in section 1.7.1 to allow it to be clearly distinguishable from training vector set-ups used for other prediction horizons.

The 5-day ahead training vector set-up is

$$r_{5day}, vl_{5day}, vt_{5day}, id_{5day}, dt_{5day}, r_{t+5} \quad (5.13)$$

where

$$r_{5day} = (r_{t-20}, r_{t-15}, r_{t-10}, r_{t-5}, r_t), \quad (5.14)$$

$$vl_{5day} = (vl_{t-20}, vl_{t-15}, vl_{t-10}, vl_{t-5}, vl_t), \quad (5.15)$$

$$vt_{5day} = (vt_{t-20}, vt_{t-15}, vt_{t-10}, vt_{t-5}, vt_t), \quad (5.16)$$

$$id_{5day} = (sp_t, ny_t, dj_t), \quad (5.17)$$

and

$$dt_{5day} = (d_t, m_t, wd_t). \quad (5.18)$$

Note that here the lagged series ( $r_{5day}$ ,  $vl_{5day}$  and  $vt_{5day}$ ) includes temporal information spanning 20 days of trading. Note also that we use 5-day returns (i.e.  $r_t = \log(p_t) - \log(p_{t-5})$ ) and 5-day (weekly) volatilities. These changes are necessary to enable the ensembles to identify 5-day temporal structures (if they exist) in the training data.

The 10-day ahead training vector set-up is

$$r_{10day}, vl_{10day}, vt_{10day}, id_{10day}, dt_{10day}, r_{t+10} \quad (5.19)$$

where

$$r_{10day} = (r_{t-40}, r_{t-30}, r_{t-20}, r_{t-10}, r_t), \quad (5.20)$$

$$vl_{10day} = (vl_{t-40}, vl_{t-30}, vl_{t-20}, vl_{t-10}, vl_t), \quad (5.21)$$

$$vt_{10day} = (vt_{t-40}, vt_{t-30}, vt_{t-20}, vt_{t-10}, vt_t), \quad (5.22)$$

$$id_{10day} = (sp_t, ny_t, dj_t), \quad (5.23)$$

and

$$dt_{10day} = (d_t, m_t, wd_t). \quad (5.24)$$

For reasons identical to those for the 5-day ahead prediction horizon the lagged series

includes temporal information spanning 40 days of trading and we use 10-day returns and volatilities.

The 20-day ahead training vector set-up is

$$r_{20day}, vl_{20day}, vt_{20day}, id_{20day}, dt_{20day}, r_{t+20} \quad (5.25)$$

where

$$r_{20day} = (r_{t-80}, r_{t-60}, r_{t-40}, r_{t-20}, r_t), \quad (5.26)$$

$$vl_{20day} = (vl_{t-80}, vl_{t-60}, vl_{t-40}, vl_{t-20}, vl_t), \quad (5.27)$$

$$vt_{20day} = (vt_{t-80}, vt_{t-60}, vt_{t-40}, vt_{t-20}, vt_t), \quad (5.28)$$

$$id_{20day} = (sp_t, ny_t, dj_t), \quad (5.29)$$

and

$$dt_{20day} = (d_t, m_t, wd_t). \quad (5.30)$$

Here the lagged series includes temporal information spanning 80 days of trading and we use 20-day returns and volatilities.

The results of all experiments performed on the stock-market data-sets are summarised in tables 5.1, 5.2, 5.3 and 5.4 and analysed in section 5.4.1. Note that the predicted returns for these (and experiments performed on all other data-sets) are transformed back to prices for reporting the results.

### 5.3.2 Stock-market indices

We also test the prediction performance of the ensembles on 11 years (1/9/88-1/9/99) of 2 stock-market index (S&P500 and NYSE) data-sets. The training vector set-up is very similar to that used above for the stock-market data-sets. For example, the 1-day ahead training vector set-up for the S&P500 data-set is

$$sp_{1day}, vl_{1day}, vt_{1day}, id_{1day}, dt_{1day}, sp_{t+1} \quad (5.31)$$



where

$$sp_{1day} = (sp_{t-4}, sp_{t-3}, sp_{t-2}, sp_{t-1}, sp_t), \quad (5.32)$$

$$vl_{1day} = (vl_{t-4}, vl_{t-3}, vl_{t-2}, vl_{t-1}, vl_t), \quad (5.33)$$

$$vt_{1day} = (vt_{t-4}, vt_{t-3}, vt_{t-2}, vt_{t-1}, vt_t), \quad (5.34)$$

$$id_{1day} = (ny_t, dj_t), \quad (5.35)$$

and

$$dt_{1day} = (d_t, m_t, wd_t). \quad (5.36)$$

The training vectors for the other index data-sets and for all the prediction horizons can be trivially derived from this.

The results of all experiments performed on the stock-market index data-sets are summarised in tables 5.5 and 5.6 and analysed in section 5.4.2.

### 5.3.3 Foreign exchange

We test the ensembles using 5 years (20/5/92 - 20/5/97) of 2 foreign exchange market data-sets; CHF/JPY and USD/JPY. Again, the training vector set-up is very similar to that used for the stock-market data-sets. However, the dates spanning the training and test sets will be different but can be trivially derived. The 1-day ahead training vector set-up for each foreign exchange data-set is

$$r_{1day}, vt_{1day}, i_{1day}, dt_{1day}, r_{t+1} \quad (5.37)$$

where

$$r_{1day} = (r_{t-4}, r_{t-3}, r_{t-2}, r_{t-1}, r_t), \quad (5.38)$$

$$vt_{1day} = (vt_{t-4}, vt_{t-3}, vt_{t-2}, vt_{t-1}, vt_t), \quad (5.39)$$

$$i_{1day} = (i_{t-4}, i_{t-3}, i_{t-2}, i_{t-1}, i_t), \quad (5.40)$$

and

$$dt_{1day} = (d_t, m_t, wd_t). \quad (5.41)$$

As for the stock-market index data-sets we will not detail the exact make-up of each training vector set-up for each prediction horizon as it can be trivially derived from this.

The results of all experiments performed on the foreign exchange data-sets are summarised in tables 5.7 and 5.8 and analysed in section 5.4.3.

## 5.4 Analysis of results

In this section we discuss and analyse the results of all the experiments described in this chapter. The aim is to establish the overall predictive accuracy of the ensembles and to highlight important properties of the ensembles such as dependancies of predictive accuracy on the volatility of the test sets and on the prediction horizons. Determining what constitutes “good” prediction performance is not easy in the context of financial time-series prediction. However, as a guideline we expect the ensembles to at least out-perform the martingale random walk model and achieve a directional prediction accuracy of above 50%<sup>2</sup>.

Each table (5.1-5.8) consists of results for a single data-set (e.g. Coca-Cola) across 4 prediction horizons – 1-day, 5-days, 10-days and 20-days ahead. For each prediction horizon we include the results of 6 experiments – simulated-live (SIMLIVE) and back-tests 1-5 (BACKTEST1-5). The evaluation metrics that we include in each table are the root mean squared error (*RMSE*), the root mean squared error of the martingale random walk model (*RW*), the information coefficient (*IC*), the correlation coefficient squared (*CC*<sup>2</sup>) and the directional prediction accuracy (expressed as a percentage) (*DC*). We also include the average volatility of the market over a test set (*VT*) and the interval non-coverage (expressed as a percentage) for 80%, 90%, 95% and 99% intervals.

Note that for all experiments we trained ensembles of 200 networks. Each ensemble required approximately 1 hour of compute time to be trained using the C-MPI

---

<sup>2</sup>Anything above a directional prediction accuracy of 52% should recover costs and generate a profit. The best traders predict directional change correctly 55-60% of the time (personal communication *Beacon Systems Ltd.*).

implementation of NeuralBAG and 8 processing nodes on the TCD CS Department SCI Cluster.

### 5.4.1 Stocks

In this section we discuss the results of the experiments performed on the stock-market data-sets (tables 5.1-5.4).

Overall the results of the Coca-Cola experiments are quite promising. The 1-day ahead and 5-day ahead predictions are consistent and the BACKTEST3 and BACKTEST4 experiments over these horizons are particularly good. The average interval non-coverage is also very good over these horizons. The results for the 10-day ahead and 20-day ahead predictions (although on average are roughly equivalent to the 5-day and 10-day ahead results) are highly variable however – although the BACKTEST3 and BACKTEST4 results are excellent, the SIMLIVE, BACKTEST1 and BACKTEST2 results are poor. Note that the very poor results e.g. BACKTEST2 correspond to periods of very high volatility. This is a phenomenon we will see recurring in the results of experiments performed on the other data-sets also – it confirms what one would expect – more volatility implies more noise variance (randomness) and therefore poorer predictions. Another significant feature of the Coca-Cola results is that the quality of the prediction intervals are not sensitive to the quality of the predictions. This also confirms what one would expect – the quality of a volatility prediction does not deteriorate as the magnitude of the volatility increases. Overall these predictions if traded live on the market could generate a reasonable return.

The GEC results are more promising than the Coca-Cola results. They are generally more consistent and over the 5-day, 10-day and 20-day prediction horizons are excellent in terms of directional prediction accuracy, peaking twice at 76% for the 20-day ahead predictions. Again the poor results (e.g. BACKTEST2, 20-day ahead) correspond to periods of very high volatility. Also, the prediction intervals are not sensitive to the magnitude of the volatility. However, the prediction intervals do seem to be sensitive to the prediction horizon – although on average the quality of all the intervals are roughly equivalent, the 10-day and 20-day intervals are significantly more



variable. We will re-visit this issue in section 5.4.3. Overall, these predictions could generate a profit, especially the 10-day and 20-day ahead predictions.

The IBM results are excellent. The 5-day ahead predictions are particularly promising – an average directional prediction accuracy of 70% is exceptional, especially given the consistency of the results across all 6 of the 5-day ahead experiments. The BACKTEST3 results for the 10-day ahead predictions which have a directional prediction accuracy of 81% are also exceptional. A significant feature of all the IBM results is that there are no very poor results. This is due (in large part at least) to the absence of periods of very high volatility – volatility instead is relatively stable. This introduces consistency to the results and perhaps even predictable structure that the ensembles model to generate the excellent predictions<sup>3</sup>. The prediction intervals for the 1-day, 5-day and 10-day ahead predictions are similar in quality to those for the previous data-sets. However the 20-day predictions are consistently too wide. As previously mentioned we will re-visit this issue in section 5.4.3. Overall, the IBM results are excellent and could generate a significant profit if applied to real-world trading.

The Microsoft Corporation results are quite promising. The BACKTEST3 and BACKTEST4 experiments across all prediction horizons are particularly good. These experiments correspond to periods of low volatility. The prediction intervals are very good for the 1-day, 5-day and 10-day ahead prediction horizons but quite poor for the 20-day ahead horizon. However, they still remain good enough to be useful.

For illustrative purposes, in figure 5.2 we plot the results of the first 20 days of the GEC SIMLIVE 20-day ahead experiment. In this illustration the correspondance between the quality of the predictions and the width of the prediction intervals is clearly observable. In practical trading scenarios this is very valuable – if a trader can observe the quality of a prediction before its horizon expires then he can manage risk more effectively e.g. if an interval is relatively narrow he can place a larger trade to bet on the corresponding prediction than if the interval is relatively wide.

---

<sup>3</sup>This is merely speculation - significantly more analysis would be needed to confirm this.

Table 5.1: Coca-Cola 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	1.00	1.00	1.00	0.82	52	25.50	16	10	3	0
BACKTEST1	1.26	1.26	1.00	0.86	56	30.36	23	9	3	2
BACKTEST2	1.67	1.66	1.01	0.97	55	34.76	18	11	5	3
BACKTEST3	1.04	1.05	0.98	0.95	60	22.64	19	8	5	2
BACKTEST4	1.09	1.10	0.98	0.90	53	29.24	17	11	5	3
BACKTEST5	1.10	1.11	1.00	0.93	53	28.54	16	8	4	2
AVG	1.19	1.20	1.00	0.90	54.83	28.51	18.17	9.50	4.17	2.00
STDEV	0.25	0.24	0.01	0.06	2.93	4.16	2.64	1.38	0.98	1.10
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	1.94	1.92	1.01	0.62	55	44.34	23	14	7	0
BACKTEST1	2.91	2.93	0.99	0.37	47	61.43	23	11	8	0
BACKTEST2	4.37	4.09	1.07	0.83	50	68.87	24	13	8	2
BACKTEST3	2.09	2.25	0.93	0.82	62	41.17	24	13	6	2
BACKTEST4	2.05	2.18	0.94	0.65	72	49.33	22	13	6	3
BACKTEST5	2.37	2.37	1.00	0.72	52	47.60	24	14	5	0
AVG	2.62	2.62	0.99	0.67	56.33	52.12	23.33	13.00	6.67	1.17
STDEV	0.92	0.79	0.05	0.17	9.22	10.73	0.82	1.10	1.21	1.33
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	2.66	2.61	1.02	0.32	50	57.72	24	14	9	2
BACKTEST1	3.44	3.62	0.95	0.38	42	72.89	23	13	7	3
BACKTEST2	6.84	6.41	1.07	0.62	36	113.47	21	11	4	1
BACKTEST3	2.75	3.31	0.83	0.85	80	57.95	16	6	3	1
BACKTEST4	2.85	3.23	0.88	0.63	61	60.98	23	10	7	2
BACKTEST5	3.53	3.62	0.97	0.67	50	70.50	17	9	6	2
AVG	3.68	3.80	0.95	0.58	53.17	72.25	20.67	10.50	6.00	1.83
STDEV	1.59	1.33	0.09	0.20	15.63	21.19	3.39	2.88	2.19	0.75
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	3.33	3.15	1.06	0.07	47	75.80	21	7	5	1
BACKTEST1	3.07	3.56	0.86	0.13	43	75.52	24	9	4	1
BACKTEST2	12.52	10.76	1.16	0.19	33	188.53	16	6	3	0
BACKTEST3	3.86	4.94	0.78	0.54	81	87.20	16	7	3	1
BACKTEST4	5.57	14.95	0.37	0.76	92	93.08	16	8	3	0
BACKTEST5	4.48	5.64	0.79	0.33	57	119.99	17	9	4	0
AVG	5.47	7.17	0.84	0.34	58.83	106.69	18.33	7.67	3.67	0.50
STDEV	3.57	4.69	0.27	0.27	23.03	43.28	3.39	1.21	0.82	0.55



Table 5.2: General Electric Corporation 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	2.00	2.01	0.99	0.85	44	29.01	19	8	3	1
BACKTEST1	1.81	1.85	0.98	0.94	52	28.58	17	10	7	1
BACKTEST2	1.83	1.84	0.99	0.91	54	33.56	16	9	5	3
BACKTEST3	0.95	0.97	0.98	0.99	51	19.43	16	9	3	2
BACKTEST4	1.26	1.28	0.98	0.96	58	30.04	18	7	5	3
BACKTEST5	1.03	1.06	0.98	0.96	57	26.07	22	8	4	2
AVG	1.38	1.40	0.98	0.95	52.67	27.53	18.00	8.50	4.50	2.00
STDEV	0.42	0.42	0.01	0.03	5.05	5.28	2.28	1.05	1.52	0.89
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	4.43	4.48	0.99	0.36	50	56.82	22	7	4	1
BACKTEST1	3.87	3.95	0.98	0.74	66	54.41	23	12	4	0
BACKTEST2	4.39	4.23	1.04	0.55	60	65.39	16	7	3	0
BACKTEST3	1.96	1.96	1.00	0.76	57	32.36	16	7	3	1
BACKTEST4	2.51	2.53	0.99	0.96	61	47.51	19	8	4	0
BACKTEST5	2.28	2.39	0.95	0.86	60	52.78	15	7	2	0
AVG	3.00	3.01	0.99	0.77	59.00	50.49	18.50	8.00	3.33	0.33
STDEV	1.06	1.01	0.03	0.15	5.29	12.04	3.39	2.00	0.82	0.52
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	5.77	5.71	1.01	0.10	54	71.08	25	10	4	1
BACKTEST1	5.14	5.68	0.91	0.58	74	70.47	16	11	5	0
BACKTEST2	6.24	6.38	0.98	0.23	50	99.01	14	6	3	1
BACKTEST3	2.57	2.81	0.92	0.58	60	45.08	16	3	2	0
BACKTEST4	2.50	2.81	0.89	0.52	69	53.13	14	3	2	0
BACKTEST5	3.35	3.69	0.91	0.76	65	81.57	16	5	2	0
AVG	3.96	4.27	0.92	0.53	62.00	69.85	16.83	6.33	3.00	0.33
STDEV	1.66	1.66	0.03	0.19	9.10	21.69	4.12	3.44	1.26	0.52
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	7.73	8.01	0.97	0.02	52	96.12	22	11	5	2
BACKTEST1	5.45	7.52	0.73	0.59	76	84.07	19	7	6	4
BACKTEST2	7.67	7.59	1.01	0.03	45	101.41	25	15	8	4
BACKTEST3	4.08	4.37	0.93	0.41	76	66.06	15	5	2	0
BACKTEST4	3.90	3.96	0.98	0.23	65	74.03	19	9	6	1
BACKTEST5	4.94	5.54	0.89	0.56	50	134.70	16	4	3	1
AVG	5.21	5.80	0.92	0.37	60.67	92.05	19.33	8.50	5.00	2.00
STDEV	1.52	1.71	0.10	0.24	13.59	27.25	3.72	4.09	2.19	1.67



Table 5.3: International Business Machines 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	2.85	2.89	0.99	0.95	54	40.46	16	6	4	2
BACKTEST1	1.98	1.99	1.00	0.90	57	34.41	16	8	6	1
BACKTEST2	1.27	1.34	0.95	0.91	56	31.99	19	7	5	1
BACKTEST3	0.88	0.88	1.00	0.89	58	30.14	22	8	7	2
BACKTEST4	1.28	1.29	0.99	0.97	52	37.32	17	6	3	0
BACKTEST5	0.85	0.88	0.96	0.98	68	31.50	16	6	4	0
AVG	1.52	1.54	0.98	0.93	57.50	34.30	17.67	6.83	4.83	1.00
STDEV	0.77	0.77	0.02	0.04	5.58	3.94	2.42	0.98	1.47	0.89
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	6.79	6.76	1.00	0.76	64	73.37	24	14	7	3
BACKTEST1	4.15	4.23	0.98	0.82	64	64.54	24	14	7	1
BACKTEST2	2.85	3.08	0.93	0.81	65	63.03	22	12	5	1
BACKTEST3	1.99	2.03	0.98	0.95	82	54.36	22	13	5	0
BACKTEST4	2.20	2.37	0.93	0.91	75	61.69	23	10	3	1
BACKTEST5	1.84	1.93	0.95	0.92	72	56.73	20	9	4	0
AVG	3.30	3.40	0.96	0.86	70.33	62.29	22.50	12.00	5.17	1.00
STDEV	1.91	1.86	0.03	0.07	7.34	6.67	1.52	2.10	1.60	1.10
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	9.33	9.30	1.00	0.66	61	105.97	19	14	7	1
BACKTEST1	5.44	5.52	0.99	0.72	62	85.50	18	13	7	0
BACKTEST2	4.25	4.28	0.99	0.71	54	89.57	22	12	7	1
BACKTEST3	2.69	2.82	0.95	0.95	81	54.36	17	8	4	1
BACKTEST4	2.78	2.80	0.99	0.91	53	73.54	16	9	5	1
BACKTEST5	2.85	2.87	0.99	0.87	51	85.78	22	11	7	2
AVG	4.56	4.60	0.99	0.80	60.33	82.45	19.00	11.17	6.17	1.00
STDEV	2.58	2.55	0.02	0.12	11.06	17.28	2.53	2.32	1.33	0.63
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	14.08	14.03	1.00	0.50	66	155.02	13	4	3	3
BACKTEST1	7.47	7.49	1.00	0.55	60	125.59	13	6	5	1
BACKTEST2	6.01	6.03	1.00	0.41	62	116.33	12	5	3	0
BACKTEST3	3.11	4.00	0.78	0.92	76	54.36	16	8	2	1
BACKTEST4	2.85	3.08	0.93	0.66	69	79.48	17	7	3	2
BACKTEST5	4.83	4.83	1.00	0.21	54	146.38	16	6	3	2
AVG	6.39	6.57	0.95	0.54	64.50	112.86	14.50	6.00	3.17	1.50
STDEV	4.15	3.97	0.09	0.24	7.64	39.02	2.07	1.41	0.98	1.05

Table 5.4: Microsoft Corporation 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	2.03	2.05	0.99	0.87	54	37.59	17	9	5	1
BACKTEST1	2.01	1.99	1.01	0.81	50	42.07	19	8	3	0
BACKTEST2	1.38	1.40	0.98	0.91	53	41.42	22	9	5	1
BACKTEST3	0.91	0.91	1.00	0.84	57	30.86	18	11	4	2
BACKTEST4	0.59	0.63	0.93	0.97	65	30.05	21	8	4	1
BACKTEST5	0.68	0.70	0.97	0.95	64	34.67	17	9	3	0
AVG	1.27	1.28	0.98	0.89	57.17	36.11	19.00	9.00	4.00	0.83
STDEV	0.64	0.63	0.03	0.06	6.11	5.14	2.10	1.10	0.89	0.75
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	4.10	4.25	0.97	0.47	50	65.10	21	9	3	0
BACKTEST1	4.41	4.40	1.00	0.52	52	80.00	20	7	3	1
BACKTEST2	3.16	3.16	1.00	0.43	49	80.97	22	11	5	2
BACKTEST3	1.89	2.11	0.90	0.86	77	47.16	17	8	6	0
BACKTEST4	1.36	1.37	0.99	0.81	66	52.49	16	8	6	1
BACKTEST5	1.55	1.58	0.98	0.83	59	63.72	18	9	5	0
AVG	2.75	2.81	0.97	0.65	58.83	64.91	19.00	8.67	4.67	0.67
STDEV	1.33	1.33	0.04	0.20	10.98	13.83	2.37	1.37	1.37	0.82
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	5.82	5.92	0.98	0.17	52	94.36	23	14	8	0
BACKTEST1	6.49	6.49	1.00	0.16	49	121.92	16	5	3	0
BACKTEST2	4.72	4.74	1.00	0.15	51	125.87	17	9	4	1
BACKTEST3	2.65	2.81	0.94	0.83	81	43.04	20	7	6	1
BACKTEST4	1.89	1.91	0.99	0.67	76	67.57	18	11	3	0
BACKTEST5	2.13	2.13	1.00	0.18	54	85.25	16	7	3	0
AVG	3.95	4.00	0.98	0.36	60.50	89.67	18.33	8.83	4.50	0.33
STDEV	1.99	1.99	0.02	0.31	14.12	31.80	2.73	3.25	2.07	0.52
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	7.27	8.45	0.86	0.01	54	135.14	22	11	5	2
BACKTEST1	10.01	10.10	0.99	0.02	50	199.79	18	8	4	1
BACKTEST2	6.15	6.18	1.00	0.15	51	147.58	10	2	0	0
BACKTEST3	3.95	3.97	0.99	0.19	58	127.14	12	3	1	1
BACKTEST4	2.22	2.28	0.98	0.52	64	79.92	16	8	3	2
BACKTEST5	3.01	3.10	0.97	0.21	53	137.50	19	9	4	1
AVG	5.43	5.68	0.96	0.18	55.00	137.85	16.17	6.83	2.83	1.17
STDEV	2.94	3.12	0.05	0.19	5.22	38.49	4.49	3.54	1.94	0.75



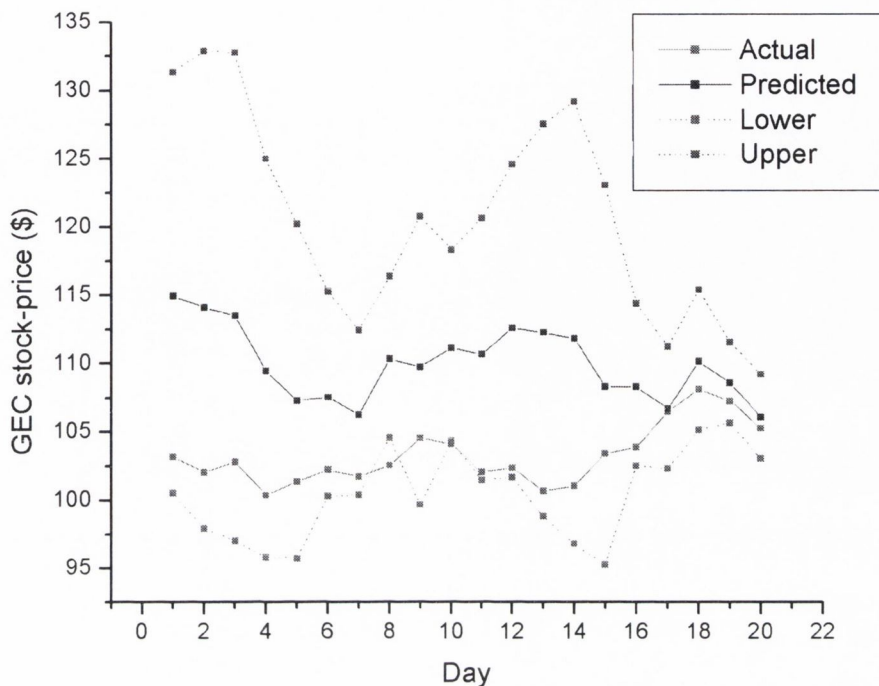


Figure 5.2: 20 days of GEC SIMLIVE 20-day ahead predictions with 90% prediction intervals.

### 5.4.2 Foreign exchange and stock-market indices

In this section we discuss the results of the experiments performed with the the foreign exchange and stock-market index data-sets (tables 5.4-5.8). In general, all of these results have properties very similar to those observed for the stock-market results and so do not merit much extra discussion.

One unique feature of these results however is their consistency. This is no surprise for the stock-market index results – the fact that a daily stock-market index observation is essentially an aggregate of a large number of daily stock prices stabilises the returns series somewhat which reduces volatility. However, consistency across most of the foreign exchange results is a surprise given that foreign exchange



markets are traditionally very volatile. However, the data-sets chosen for the experiments (CHF/JPY and USD/JPY) are widely accepted as amongst those with the lowest volatility given the relative stability of the underlying economies that drive the exchange rates. Experiments performed on other exchange rate data-sets (especially those subject to occasional government intervention e.g. the Thai Bhat, Brazilian Real) would not likely be as fruitful.

### 5.4.3 Prediction interval performance

Given that the prediction intervals are an important focus of this thesis, in this section we discuss their performance in more detail.

Overall, the quality of the prediction intervals is very high. This is especially significant given the number of possible sources of error e.g. bias in the ensemble predictions (the mean of the predicted conditional distribution), bias in the volatility predictions e.g. caused by a poorly chosen decay parameter in the EWMA model, bias in the model variance estimate and “extreme” or non-Gaussian market conditions (which can occur quite frequently in most stock and foreign exchange financial markets).

The most encouraging result is the performance of the intervals over the SIMLIVE experiments. This is important – as discussed in section 5.3 these are the only truly “out-of-sample” experiments for the prediction intervals. These results are summarised in table 5.9. Note that although, on average, the performance of the SIMLIVE intervals is roughly equivalent across all horizons there is more variability across the 10-day and 20-day ahead horizons. This phenomenon was also observed for the BACKTEST intervals as discussed in section 5.4.1. Also, for some of the BACKTEST experiments, the interval non-coverage for the 10-day and 20-day ahead horizons is poorer.

The reason for this poorer performance of the intervals over longer horizons can be attributed (at least in large part) to two related factors. Firstly, the ensemble predictions over the longer horizons are likely to be more biased than over the shorter horizons. This is simply because it is a more difficult prediction task. Secondly, for

Table 5.5: CHF/JPY 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	0.46	0.47	0.98	0.97	52	31.06	22	11	5	1
BACKTEST1	0.44	0.46	0.97	0.98	58	29.46	18	7	3	1
BACKTEST2	0.41	0.41	1.00	0.99	71	33.09	17	6	2	0
BACKTEST3	0.45	0.45	1.00	0.89	62	35.20	19	8	5	2
BACKTEST4	0.46	0.47	0.98	0.89	53	31.60	22	9	7	1
BACKTEST5	1.89	1.88	1.00	0.88	49	41.91	21	10	3	1
AVG	0.69	0.69	0.99	0.93	57.50	33.72	19.83	8.50	4.17	1.00
STDEV	0.59	0.59	0.01	0.05	8.07	4.46	2.14	1.87	1.83	0.63
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	1.10	1.10	1.00	0.82	52	52.64	20	9	3	1
BACKTEST1	0.89	0.88	1.01	0.54	48	75.81	19	8	2	0
BACKTEST2	1.01	1.04	0.97	0.91	62	36.39	21	7	4	1
BACKTEST3	1.00	1.11	0.90	0.93	59	37.90	19	9	6	2
BACKTEST4	1.25	1.26	1.00	0.65	63	41.21	23	8	5	1
BACKTEST5	1.28	1.28	0.99	0.69	54	37.13	20	8	6	2
AVG	1.09	1.11	0.98	0.76	56.33	46.85	20.33	8.17	4.33	1.17
STDEV	0.15	0.15	0.04	0.16	5.96	15.41	1.51	0.75	1.63	0.75
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	1.57	1.58	1.00	0.85	52	23.81	17	9	4	0
BACKTEST1	1.27	1.27	0.99	0.72	56	18.07	15	5	2	0
BACKTEST2	1.49	1.55	0.96	0.81	76	19.12	17	7	3	1
BACKTEST3	1.60	1.62	0.99	0.76	61	24.79	19	6	2	0
BACKTEST4	2.02	2.01	1.00	0.53	52	31.23	21	9	5	0
BACKTEST5	1.66	1.67	0.99	0.63	54	26.16	19	8	3	1
AVG	1.60	1.62	0.99	0.72	58.50	23.86	18.00	7.33	3.17	0.33
STDEV	0.25	0.24	0.02	0.12	9.20	4.83	2.10	1.63	1.17	0.52
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	2.02	2.03	1.00	0.58	51	31.21	24	12	7	3
BACKTEST1	1.57	1.57	1.00	0.79	69	23.70	23	8	4	1
BACKTEST2	1.53	1.55	0.99	0.67	56	34.42	27	13	6	2
BACKTEST3	2.28	2.31	0.99	0.81	59	36.31	24	11	4	1
BACKTEST4	3.52	3.51	1.00	0.16	54	55.91	22	10	6	2
BACKTEST5	3.51	3.53	1.00	0.44	68	51.98	23	12	4	1
AVG	2.41	2.42	0.99	0.58	59.50	38.92	23.83	11.00	5.17	1.67
STDEV	0.91	0.90	0.01	0.25	7.45	12.47	1.72	1.79	1.33	0.82



Table 5.6: USD/JPY 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	0.62	0.62	0.99	0.91	55	7.79	22	11	4	1
BACKTEST1	0.44	0.46	0.97	0.98	62	6.41	20	10	6	0
BACKTEST2	0.50	0.50	1.00	0.96	53	7.57	21	13	4	2
BACKTEST3	0.59	0.59	1.00	0.92	51	9.84	19	11	3	1
BACKTEST4	0.95	0.93	1.02	0.85	50	15.36	21	8	4	0
BACKTEST5	0.76	0.76	1.00	0.91	50	11.53	17	7	3	0
AVG	0.64	0.64	1.00	0.92	53.50	9.75	20.00	10.00	4.00	0.67
STDEV	0.18	0.18	0.02	0.05	4.59	3.30	1.79	2.19	1.10	0.82
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	1.48	1.50	0.99	0.65	53	16.43	23	13	6	2
BACKTEST1	1.03	1.04	0.99	0.72	72	13.11	22	12	6	2
BACKTEST2	1.11	1.11	0.99	0.69	66	13.91	19	8	3	0
BACKTEST3	1.14	1.15	0.99	0.65	58	14.51	24	7	6	1
BACKTEST4	2.21	2.17	1.02	0.23	45	29.98	22	8	4	0
BACKTEST5	1.88	1.87	1.01	0.38	50	24.53	20	11	6	2
AVG	1.47	1.47	1.00	0.55	57.33	18.74	21.67	9.83	5.17	1.17
STDEV	0.48	0.46	0.01	0.20	10.15	6.90	1.86	2.48	1.33	0.98
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	2.29	2.30	1.00	0.51	53	24.88	18	9	3	1
BACKTEST1	1.47	1.48	0.99	0.46	55	18.68	15	5	2	0
BACKTEST2	1.62	1.65	0.98	0.39	57	22.16	15	6	3	0
BACKTEST3	1.48	1.52	0.98	0.72	63	18.62	17	7	2	0
BACKTEST4	3.22	3.20	1.01	0.08	47	44.38	16	5	4	1
BACKTEST5	2.90	2.82	1.03	0.12	51	36.42	18	9	3	0
AVG	2.16	2.16	1.00	0.38	54.33	27.52	16.50	6.83	2.83	0.33
STDEV	0.76	0.73	0.02	0.24	5.47	10.55	1.38	1.83	0.75	0.52
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	3.70	3.68	1.00	0.13	50	41.44	16	7	2	0
BACKTEST1	1.87	1.95	0.96	0.65	68	24.67	17	8	3	1
BACKTEST2	1.51	1.65	0.91	0.68	72	26.50	15	7	3	2
BACKTEST3	2.23	2.26	0.99	0.59	65	28.51	19	8	3	1
BACKTEST4	4.79	4.78	1.00	0.02	47	69.87	22	7	4	2
BACKTEST5	4.75	4.73	1.00	0.05	51	64.52	12	4	2	0
AVG	3.14	3.18	0.98	0.35	58.83	42.59	16.83	6.83	2.83	1.00
STDEV	1.46	1.41	0.04	0.32	10.72	20.02	3.43	1.47	0.75	0.89



Table 5.7: S&amp;P500 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	14.48	14.51	1.00	0.98	65	17.12	19	10	4	2
BACKTEST1	15.31	15.31	1.00	0.90	57	19.30	23	13	6	2
BACKTEST2	17.26	17.18	1.00	0.93	56	24.54	21	11	3	1
BACKTEST3	8.90	8.92	1.00	0.96	75	12.78	19	9	4	0
BACKTEST4	12.67	12.67	1.00	0.95	53	20.35	21	7	2	1
BACKTEST5	9.14	9.15	1.00	0.98	73	16.44	23	11	7	1
AVG	12.96	12.96	1.00	0.95	63.17	18.42	21.00	10.17	4.33	1.17
STDEV	3.39	3.37	0.00	0.03	9.30	3.98	1.79	2.04	1.86	0.75
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	32.23	33.47	0.96	0.54	56	34.89	22	11	8	2
BACKTEST1	31.54	31.45	1.00	0.27	46	47.66	19	10	6	1
BACKTEST2	37.97	39.04	0.97	0.77	69	24.23	23	13	7	2
BACKTEST3	18.99	19.07	1.00	0.63	54	32.93	22	11	7	0
BACKTEST4	23.98	23.99	1.00	0.68	53	29.10	24	12	5	1
BACKTEST5	19.39	19.38	1.00	0.72	55	34.45	23	11	7	2
AVG	27.35	27.73	0.99	0.60	55.50	33.88	22.17	11.33	6.67	1.33
STDEV	7.73	8.17	0.02	0.18	7.50	7.85	1.72	1.03	1.03	0.82
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	41.00	41.06	1.00	0.56	52	41.02	29	16	7	7
BACKTEST1	39.18	39.19	1.00	0.67	55	41.43	23	12	6	3
BACKTEST2	57.98	56.93	1.02	0.21	49	74.44	24	13	8	2
BACKTEST3	24.19	24.91	0.97	0.87	71	32.55	23	11	7	1
BACKTEST4	24.13	24.57	0.98	0.72	63	33.83	24	12	6	2
BACKTEST5	28.23	28.24	1.00	0.82	54	43.71	23	12	7	2
AVG	35.79	35.82	0.99	0.64	57.33	44.50	24.33	12.67	6.83	2.83
STDEV	13.11	12.55	0.02	0.24	8.16	15.33	2.34	1.75	0.75	2.14
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	38.34	39.69	0.97	0.55	54	54.01	24	8	6	2
BACKTEST1	56.35	56.16	1.00	0.09	49	65.26	12	6	3	0
BACKTEST2	74.36	72.93	1.02	0.13	50	91.17	15	5	2	0
BACKTEST3	40.13	42.87	0.94	0.79	62	52.80	14	4	1	0
BACKTEST4	26.93	32.34	0.83	0.83	75	45.93	19	7	3	0
BACKTEST5	47.15	46.07	1.02	0.20	54	76.59	21	7	3	1
AVG	47.21	48.34	0.96	0.43	57.33	64.29	17.50	6.17	3.00	0.50
STDEV	16.50	14.36	0.07	0.34	9.79	17.03	4.59	1.47	1.67	0.84

Table 5.8: NYSE 1-day, 5-days, 10-days and 20-days ahead prediction results.

<b>1-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	5.53	5.67	0.97	0.98	68	14.16	21	11	4	1
BACKTEST1	6.04	6.05	1.00	0.92	54	16.10	22	9	6	0
BACKTEST2	7.99	7.93	1.01	0.88	48	22.66	22	11	6	2
BACKTEST3	4.26	4.27	1.00	0.96	58	11.76	20	9	5	1
BACKTEST4	6.03	6.02	1.00	0.91	52	18.20	21	8	4	0
BACKTEST5	4.15	4.17	1.00	0.95	56	14.60	20	9	5	2
AVG	5.67	5.69	1.00	0.93	56.00	16.25	21.00	9.50	5.00	1.00
STDEV	1.41	1.38	0.01	0.04	6.81	3.80	0.89	1.22	0.89	0.89
<b>5-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	13.46	13.48	1.00	0.68	58	29.13	23	13	7	1
BACKTEST1	12.91	12.93	1.00	0.72	57	30.08	22	11	8	2
BACKTEST2	18.90	18.90	1.00	0.43	51	45.39	22	11	7	1
BACKTEST3	9.47	9.48	1.00	0.76	63	23.03	19	12	6	0
BACKTEST4	11.98	11.96	1.00	0.40	52	31.02	21	11	5	1
BACKTEST5	9.11	9.12	1.00	0.65	60	26.07	22	10	6	2
AVG	12.64	12.64	1.00	0.61	56.83	30.79	21.50	11.33	6.50	1.17
STDEV	3.55	3.54	0.00	0.15	4.62	7.73	1.38	1.03	1.05	0.75
<b>10-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	17.16	17.20	1.00	0.53	62	36.25	22	12	7	1
BACKTEST1	16.18	16.44	0.98	0.65	63	35.62	21	11	6	1
BACKTEST2	29.99	28.06	1.07	0.30	50	72.56	23	12	7	2
BACKTEST3	11.89	12.59	0.94	0.87	71	31.95	19	9	4	0
BACKTEST4	12.01	12.22	0.98	0.77	56	32.79	22	11	7	2
BACKTEST5	13.53	13.57	1.00	0.66	53	40.41	23	12	6	1
AVG	16.79	16.68	1.00	0.63	59.17	41.60	21.67	11.17	6.17	1.17
STDEV	6.81	5.94	0.04	0.20	7.68	15.46	1.51	1.17	1.17	0.75
<b>20-DAY</b>	RMSE	RW	IC	$CC^2$	DC	VT	80%	90%	95%	99%
SIMLIVE	38.89	37.09	1.05	0.03	50	49.58	25	13	8	2
BACKTEST1	21.45	22.10	0.97	0.69	62	49.73	24	13	8	1
BACKTEST2	39.56	37.30	1.06	0.12	48	92.02	22	11	6	2
BACKTEST3	19.57	21.58	0.91	0.82	80	51.59	23	14	7	2
BACKTEST4	15.29	16.41	0.93	0.75	67	44.56	22	11	7	1
BACKTEST5	23.92	22.31	1.07	0.09	51	71.63	24	11	7	1
AVG	26.45	26.13	1.00	0.42	59.67	59.85	23.33	12.17	7.17	1.50
STDEV	10.29	8.84	0.07	0.37	12.47	18.35	1.21	1.33	0.75	0.55



Table 5.9: Average interval non-coverage across all of the SIMLIVE experiments for the 80%, 90%, 95% and 99% prediction intervals.

	80%	STDEV	90%	STDEV	95%	STDEV	99%	STDEV
1-day	19.00	2.51	9.5	1.77	4	0.76	1.13	0.64
5-day	22.25	1.28	11.25	2.66	5.63	1.99	1.25	1.04
10-day	22.13	4.02	12.25	2.66	6.13	2.17	1.63	2.26
20-day	20.88	4.22	9.13	3.09	5.13	1.96	1.88	0.99

essentially the same reason the volatility predictions are likely to be more biased – volatility predictions over longer horizons are never as accurate as those over shorter horizons – see (Alexander, 1998) for a discussion. However, despite this occasional poor performance of the prediction intervals, on average they are excellent and potentially very valuable in real-world trading scenarios.

## 5.5 Summary

In this chapter we evaluated the prediction performance of our ensembles. We used a number of popular objective statistical measures to do this including the root-mean-squared-error, the correlation coefficient, the information coefficient and the d-statistic. We also evaluated the ability of our ensembles to predict uncertainty by measuring the prediction interval non-coverage.

Overall, the performance of the ensembles is promising. Although there are examples of poor prediction performance, we showed how these are mostly caused by excessive volatility in the corresponding test sets. However, we also showed how this minority of poor predictions are anticipated by the prediction intervals and so in a real-world trading scenario a trader can adjust his trading behaviour accordingly – essentially manage risk more effectively. The overall quality of the prediction intervals is excellent. Although there is some degradation in their quality over the longer prediction horizons on average even the performance of these intervals is good.

In summary, we can conclude from the results presented in this chapter that the ensembles do provide valuable insights into future market behaviour (future prices and directional change) if market volatility is not excessive. However, a unique feature of



the prediction methodology that we propose is that the risk that traders are exposed to during such periods of excessive volatility can at least be more effectively managed using the prediction intervals.

# Chapter 6

## Conclusion

### 6.1 Introduction

In this chapter we conclude the thesis. In section 6.2 we discuss the main contributions or novel aspects of the thesis by identifying the contributions to the individual disciplines from which new techniques in the thesis were derived. In section 6.3 we discuss future work i.e. approaches that could be taken to improve the performance of the techniques proposed in the thesis. Finally, in section 6.4 we summarise the main conclusions of the thesis.

### 6.2 Thesis contributions

In this section we attempt to identify the main contributions of the thesis. The approach is different to that taken in section 1.8 – given the multi-disciplinary nature of the thesis we attempt to identify contributions to the individual disciplines from which techniques in the thesis were inspired or adapted.

#### 6.2.1 Machine learning

The main contribution of the thesis to the discipline of machine learning is the development and evaluation of the NeuralBAG ensemble technique proposed in chapter 3.

Although most work on neural network ensembles has recognised the importance of diversity in ensembles, very little work has attempted to explicitly *tune* diversity as the NeuralBAG technique does. The simplicity and stability of NeuralBAG is particularly attractive, especially in the context of difficult, noisy, real-world prediction tasks.

Another key feature of the NeuralBAG algorithm is that estimates of model variance are readily available from the outputs of the individual networks in the ensemble. In chapter 4 we show how these estimates can be adapted to provide better estimates of ensemble model variance which can be used to generate confidence intervals with very good coverage. This method for generating confidence intervals can be applied to any ensemble technique that uses the bootstrap to generate the training sets for the ensemble e.g. balancing (Heskes, 1997a).

## 6.2.2 Time-series prediction

It is difficult to identify any specific contribution to the field of (modern) time-series prediction as it is itself very multi-disciplinary. However it is useful to position our prediction methodology relative to some classical time-series prediction approaches.

Using the classical time-series prediction terminology, the models that we build are low-variance, non-linear, multi-variate and semi-parametric (the neural networks and bootstrapping techniques are non-parametric, but the volatility models and interpretation of the predicted conditional distributions are parametric). Classical time-series prediction techniques e.g. auto-regressive moving-average (ARMA) methods are also typically low-variance but they are also usually linear, univariate and parametric.

The big difference here is that the classical time-series prediction techniques do not have the attractive universal approximation properties that neural networks have. Also, most will not have prediction intervals of the sophistication and accuracy of those proposed in this thesis. The big criticism of universal approximators such as neural networks from the classical time-series prediction and statistics communities has always been related to issues of variance or instability. However, this problem has been largely solved using ensemble techniques. In this context, the contribution



of this thesis to the time-series prediction community is significant – much more powerful and accurate models of time-series can be built without any significant loss in stability.

### 6.2.3 Econometrics

The main contribution of this thesis to the field of econometrics is our prediction interval technique proposed in chapter 4. Here we show how classical parametric econometric (EWMA) volatility predictions can be combined with non-parametric (bootstrap) predictions for model variance and non-parametric (neural network) predictions for price to produce estimates for the future conditional distribution of market prices. The novelty here is the successful combination of econometric techniques with techniques from machine learning and statistics.

Another contribution is the set of decay factor estimates used for the EWMA models. Unlike previous attempts (see e.g. Zangari, 1996), we estimate a different decay factor for each data-set and volatility horizon. These could be used in a variety of other econometric studies.

### 6.2.4 Finance

The contributions of the thesis to the field of finance are easy to identify. Firstly, the ensembles generate predictions of financial market movements that could be applied in real-world trading to generate a significant profit. Secondly, each prediction generated by the ensembles has a quantifiable measure of confidence associated with it. This is potentially very valuable for managing risk in real-world trading scenarios.

## 6.3 Future work

In this section we identify the areas of research that could be pursued to improve the predictions generated by the techniques described in this thesis.

### 6.3.1 Feature selection

One area of research that was not pursued in any significant depth in this thesis is *feature selection* i.e. determining what are the best inputs or training vectors for the ensembles. There are two approaches to solving this problem. The first is to use expert knowledge e.g. in the context of financial markets an experienced trader or econometrician who has a feel for what influences or drives a specific market. This is essentially the approach taken in this thesis i.e. established relationships between important financial market variables were identified by surveying the econometrics literature.

The second is much more systematic and essentially attempts to “learn” which variables are important. Examples of this approach applied to neural networks include the *automatic relevance determination* (ARD) technique of (Neal, 1996). This rather more principled approach should at least be investigated as a more convenient technique for feature selection and compared to the expert based approach.

### 6.3.2 Ensembles

Although the issue of stabilising the neural networks using ensemble techniques was pursued in depth in this thesis, there is still scope for improving the techniques. For, example more novel combination techniques than averaging could be investigated. We do not expect the improvements in performance to be dramatic (such sophisticated combination techniques usually only yield significant improvements for small ensembles – we use ensembles of 200 networks). However, a small change in performance can translate to large sums of money being lost or earned in financial prediction.

### 6.3.3 Prediction intervals

This is possibly the area where there is most scope for improvement. There are two possible approaches that could be taken to improve performance. The first is to refine the current technique. The most obvious improvement that could be made here is to use a more sophisticated technique for predicting volatility. For example, some GARCH techniques can provide predictions of volatility over long horizons that are

significantly better than those provided by EWMA models – see (Alexander, 1998) for a discussion. Another way in which the current technique could be refined is to investigate using  $t$ -distribution tables instead of Gaussian distribution tables for the  $z$ -values used in the prediction intervals. These  $t$ -distributions have fatter tails than Gaussian distributions which should better model the occurrence of very large movements in prices which are often observed in financial markets. However, it is not clear how many degrees of freedom should be chosen for the  $t$ -distributions.

The second is to interpret the problem in a completely different way and use *mixture density networks* (MDNs) (Bishop, 1994) to predict entire conditional distributions, not just means and variances. Here no prior assumptions are made about the form of the conditional distribution and rare or extreme market events can (in theory at least) be captured by the predictions (e.g. by predicting a very fat tailed distribution). However, this is not a mature area of research and such MDNs have serious limitations when applied to anything other than toy problems e.g. they need very large quantities of data and have serious stability (local minima) problems (see (Husmeier, 1999) for a discussion). However, they do seem very attractive as a technique for predicting financial time-series and managing risk and will certainly be pursued in the future.

## 6.4 Summary

In this chapter we outlined the main contributions of the thesis and suggested some possible future areas of research that could be pursued to improve the performance of the techniques introduced in the thesis.

Overall, the research was a success – its ultimate aim i.e. to develop a new neural network prediction methodology that generates accurate, stable, risk-adjusted (interval) predictions of financial market movements was achieved. Significantly, a large part of this success can be attributed to the multi-disciplinary approach that was taken to solve the difficult specific problems.



# Appendix A

## NeuralBAG C-MPI code

```
/*
 *
 * NeuralBAG v1.0 - C-MPI cluster version
 * -----
 *
 * bagmain.c : Mainline of NeuralBAG (C-MPI)
 *
 * c. John Carney 13/5/99
 *
 */

#include "/CAGcluster1/CAG32/Software/Ensemble/
MPI/MPICH/MPICH1_1_1/Binaries/include/mpi.h"
#include "bagmain.h"
#include <stdio.h>
#include <stdlib.h>

#define MASTER 0
```

```
/* Variables global to all files */
```

```
double **train_array;
```

```
double **test_array;
```

```
int **bs_indices;
```

```
int **bs_waste_indices;
```

```
int *count_waste;
```

```
int *nodes;
```

```
int *rand_index;
```

```
int *stop_epoch;
```

```
double **idiff;
```

```
double **agg_errors;
```

```
double *agg_errors1d;
```

```
double **inter;
```

```
double ***wgts;
```

```
double ***batchbuf;
```

```
double ***wdiff;
```

```
int *count_voccur;
```

```
int ntrainpat;
```

```
int ntestpat;
```

```
int ninput;
```

```
int nhidden_nodes;
```

```
int maxepochs;
```

```
int num_bs;
```

```
double lrate;
```

```
double mom;
```

```
double offset;
```

```
double scale;
```

```
int nlayer=3;
```

```
/******
```

```
*
* main() : This is the main parallel
* engine function
*
*****/
```

```
void main(int argc, char *argv[])
{
```

```
/** DECLARATIONS **/
/*-----*/
```

```
MPI_Status status;
```

```
/* Mainline variables */
```

```
int numtasks;
int numworkers;
int taskid;
int indexmsg=1;
int arraymsg1d=3;
int rc;
int start_index;
int dest;
int source;
int bs_partit;
double *W_agg_errors1d;
```

```
/* Miscellaneous */
```



```
char db_filename[32];
char pr_filename[32];
char *task;
int i, j, k;

/** INITIALISE COMM WORLD **/
/*-----*/

rc = MPI_Init (&argc, &argv);
rc = MPI_Comm_size (MPI_COMM_WORLD, &numtasks);
rc = MPI_Comm_rank (MPI_COMM_WORLD, &taskid);

/* Check this worked */
if (rc != 0)
{
printf ("Error initialising MPI...exiting\n");
MPI_Finalize ();
exit (0);
}

/** READ DATA AND PARAMS **/
/*-----*/
```

```
/* Each task will have its own copy */
/* of data and params      */
task = "input"; /* This is temporary */
sprintf (pr_filename, "%s%s", task, ".par");
sprintf (db_filename, "%s%s", task, ".dat");
read_params (pr_filename);
read_patterns (db_filename);
```

```
/* Error checking */
if (num_bs%numtasks != 0)
{
printf ("Error with num_bs...exiting\n");
MPI_Finalize ();
exit (0);
}
bs_partit = num_bs/numtasks;
numworkers = numtasks-1;
```

```
/* Create bootstrap resampled datasets */
create_bs_datasets ();
```

```
/**/ MASTER ***/
/*-----*/
```

```
if (taskid == MASTER)
{
```

```
/** CALCULATE OUT-OF-BAG ERRORS **/  
/*-----*/  
  
/* Send slaves their share of the work */  
start_index = 0;  
for (dest=1; dest <= numworkers; dest++)  
{  
start_index += bs_partit;  
MPI_Send (&start_index, 1, MPI_INT,  
dest, indexmsg, MPI_COMM_WORLD);  
}  
  
/* Master must do work too */  
start_index = 0;  
printf ("\nCreating error matrix...");  
create_error_matrix (start_index,  
bs_partit, taskid);  
  
/* Receive back work done by slaves */  
W_agg_errors1d = (double *)  
malloc (sizeof (double) * (ntrainpat*maxepochs));  
for (i=1; i <= numworkers; i++)  
{  
source = i;  
MPI_Recv (&W_agg_errors1d[0],  
ntrainpat*maxepochs, MPI_DOUBLE,  
source, arraymsg1d, i  
MPI_COMM_WORLD, &status);
```



```
/* Merge */
for (j=0; j < (ntrainpat*maxepochs); j++)
agg_errors1d[j] += W_agg_errors1d[j];
}

/* Convert 1-d back to 2-d for convenience */
k = 0;
for (i=0; i < ntrainpat; i++)
for (j=0; j < maxepochs; j++)
{
agg_errors[i][j] = agg_errors1d[k];
k++;
}

/* Combine out-of-bag errors */
combine_errors ();

/** TRAIN NETWORKS **/
/*-----*/

/* Send slaves their share of the work */
start_index = 0;
for (dest=1; dest <= numworkers; dest++)
{
start_index += bs_partit;
MPI_Send (&stop_epoch[0],
```

```
num_bs, MPI_INT, dest,
arraymsgid, MPI_COMM_WORLD);
MPI_Send (&start_index,
1, MPI_INT, dest,
indexmsg, MPI_COMM_WORLD);
}

/* Master works too */
start_index = 0;
printf ("\nTraining networks");
train_bs_ensemble (start_index, bs_partit, taskid);

calc_val_res (numtasks);
}

/** WORKERS **/
/*-----*/

if (taskid > MASTER)
{
/** CALCULATE OUT-OF-BAG ERRORS **/
/*-----*/

source = MASTER;
MPI_Recv (&start_index,
1, MPI_INT, source,
indexmsg, MPI_COMM_WORLD, &status);
create_error_matrix (start_index,
bs_partit, taskid);
```

```

/* Send agg_errors1d versions
back to master to be merged */
dest = MASTER;
MPI_Send (&agg_errors1d[0],
ntrainpat*maxepochs, MPI_DOUBLE,
dest, arraymsg1d, MPI_COMM_WORLD);

/**/ TRAIN NETWORKS /**/
/*-----*/
stop_epoch = (int *) malloc
(sizeof (int) * num_bs);
MPI_Recv (&stop_epoch[0], num_bs,
MPI_INT, source, arraymsg1d,
MPI_COMM_WORLD, &status);
MPI_Recv (&start_index,
1, MPI_INT, source,
indexmsg, MPI_COMM_WORLD, &status);

/* Train */
train_bs_ensemble (start_index,
bs_partit, taskid);
}

MPI_Finalize ();

}

/*****

```



```
*
* bootstrap.c: The bootstrap data-sets to be used
* for the ensemble are created here
*
* c. John Carney 11/9/98
*
*****/

#include <stdio.h>
#include <time.h>
#include <stdlib.h>

/* Prototypes */
static void seed_randnums ();
void create_bs_datasets ();

/* External variables */
extern int **bs_indices;
extern int *count_waste;
extern int **bs_waste_indices;
extern int ntrainpat;
extern int num_bs;

/*****
*
* create_bs_datasets(): Create the bootstrap indices
*     These are int *[]
*
*****/
```

```
void create_bs_datasets ()

{

int i, j, b;
int rand_num;
int index;
int **waste_tag;

/* Allocate bs_indices memory */
bs_indices = (int **)
malloc (sizeof (int *) * num_bs);
for (i=0; i < num_bs; i++)
bs_indices[i] = (int *)
malloc (sizeof (int) * ntrainpat);

/* Allocate waste_tag memory and initialise it */
waste_tag = (int **)
malloc (sizeof (int *) * num_bs);
for (i=0; i < num_bs; i++)
waste_tag[i] = (int *)
malloc (sizeof (int) * ntrainpat);
for (b=0; b < num_bs; b++)
for (i=0; i < ntrainpat; i++)
waste_tag[b][i] = 0;
```

```
/* Generate bootstrap indices */
seed_rands ( );
for (b=0; b < num_bs; b++)
{
for (i=0; i < ntrainpat; i++)
{
rand_num = rand() % (ntrainpat);
bs_indices[b][i] = rand_num;
waste_tag[b][rand_num] = 1;
}
}

/* Allocate count_waste memory
and bs_waste_indices memory */
count_waste = (int *)
malloc (sizeof (int) * num_bs);
bs_waste_indices = (int **)
malloc (sizeof (int *) * num_bs);
for (i=0; i < num_bs; i++)
bs_waste_indices[i] = (int *)
malloc (sizeof (int) * (ntrainpat/2));

/* Generate waste indices */
for (b=0; b < num_bs; b++)
{
count_waste[b] = 0;
index = 0;
for (i=0; i < ntrainpat; i++)
```



```
if (waste_tag[b][i] == 0)
{
count_waste[b]++;
bs_waste_indices[b][index] = i;
index++;
}
}
```

```
/* Free waste_tag memory */
for (i=0; i < num_bs; i++)
free (waste_tag[i]);
free (waste_tag);

}
```

```
/******
*
* seed_randnums (): Seed the random number generator
*
*****/
```

```
static void seed_randnums ()
```

```
{
```

```
int ltime;
```

```
int utime;
```

```
ltime = time (NULL);
```

```
utime = (unsigned int) ltime/2;
```

```
srand (utime);
```

```
}
```

```
/*  
*****
```

```
*
```

```
* combine.c : File for code to combine agg_errors
```

```
*
```

```
* John Carney 18/3/98
```

```
*
```

```
*****  
*/
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <math.h>
```

```
/* Prototypes */
```

```
void combine_errors();
```

```
/* External variables */
extern int *stop_epoch;
extern int *count_waste;
extern int **bs_waste_indices;
extern double **train_array;
extern double **agg_errors;
extern int *count_voccur;
extern int maxepochs;
extern int num_bs;
extern int ntrainpat;
extern int ninput;

/*****
*
* combine_errors () : combine agg_errors
*
*****/

void combine_errors ()

{

int i, b, e, j;
int found;
double b_fx;
double y;
double min_err;
double *error;
```



```
FILE *fp;
```

```
/* Allocate memory */
```

```
error = (double *)
```

```
malloc (sizeof (double) * maxepochs);
```

```
stop_epoch = (int *)
```

```
malloc (sizeof (int) * num_bs);
```

```
count_voccur = (int *)
```

```
malloc (sizeof (int) * ntrainpat);
```

```
/* Make count_voccur */
```

```
for (i=0; i < ntrainpat; i++)
```

```
count_voccur[i] = 0;
```

```
for (i=0; i < ntrainpat; i++)
```

```
for (b=0; b < num_bs; b++)
```

```
{
```

```
found = 0;
```

```
j = 0;
```

```
while ((j < count_waste[b]) && (found == 0))
```

```
{
```

```
if (bs_waste_indices[b][j] == i)
```

```
{
```

```
count_voccur[i]++;
```

```
found = 1;
```

```
}
```

```
j++;
```

```
}
```

```
}
```

```
/* Combine errors */
for (b=0; b < num_bs; b++)
{
for (e=0; e < maxepochs; e++)
{
error[e] = 0.0;
for (j=0; j < count_waste[b]; j++)
{
b_fx =
agg_errors[(bs_waste_indices[b][j])][e] /
count_voccur[(bs_waste_indices[b][j])];
y =
train_array[(bs_waste_indices[b][j])[ninput];
error[e] += pow ((y-b_fx), 2.0);
}
error[e] /= count_waste[b];
}
stop_epoch[b] = 0;
min_err = error[0];
for (e=1; e < maxepochs; e++)
{
if (error[e] < min_err)
{
min_err = error[e];
stop_epoch[b] = e;
}
}
}
```

```

/* Free memory */
free (error);

fp = fopen ("stopepochs.out", "w+");
for (b=0; b < num_bs; b++)
fprintf (fp, "%d\n", stop_epoch[b]);
    fclose (fp);

}

/*****
*
* errors.c: Header file of code for calculating validation
*   errors for each training vector
*
* c. John Carney 21/8/98
*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "neural.h"

/* Function prototypes */

```



```
void create_error_matrix (int start_index, int bs_partit, int taskid);
static void allocate_mem (int ntrainpat, int maxepochs);
```

```
/* External variables */
extern int *rand_index;
extern int *count_waste;
extern int *nodes;
extern double **idiff;
extern double **agg_errors;
extern double *agg_errors1d;
extern int **bs_indices;
extern double **train_array;
extern int **bs_waste_indices;
extern double **inter;
extern double ***wgts;
extern double ***batchbuf;
extern double ***wdiff;
extern int ntrainpat;
extern int maxepochs;
extern int ninput;
extern double lrate;
extern double mom;
```

```
/******
```

```
*
```

```
* create_error_matrix: create the error
```

```
* validation matrix
```

```
*
```

```
*****/
```

```
void create_error_matrix (int start_index, int bs_partit, int taskid)

{

int i,j,b,k;
int epoch;
double actual_out;
double net_out;
double *in;

/* Allocate memory */
allocate_mem (ntrainpat, maxepochs);
in = (double *) malloc (sizeof (double) * ninput);

/* Initialise agg_errors */
for (i=0; i < ntrainpat; i++)
for (j=0; j < maxepochs; j++)
agg_errors[i][j] = 0.0;

/* Train each bootstrap re-sample */
for (b=start_index; b < (start_index+bs_partit); b++)
{
if (taskid == 0)
printf ("\nMASTER: Error matrix %d", (b+1));

init_weights (ninput);
epoch = 0;
```

```
for (i=0; i < ntrainpat; i++)
rand_index[i] = bs_indices[b][i];

do
{
/* Present training patterns */
for (j=0; j < ntrainpat; j++)
{
for (k=0; k < ninput; k++)
in[k] =
train_array[(rand_index[j])][k];
actual_out =
train_array[(rand_index[j])][ninput];
net_out = feed_forward (in);
compute_difference (actual_out);
propagate_backwards ();
update_weights (epoch);
}

/* Create errors matrix */
for (j=0; j < count_waste[b]; j++)
{
for (k=0; k < ninput; k++)
in[k] =
train_array[(bs_waste_indices[b][j])][k];
net_out = feed_forward (in);
agg_errors[(bs_waste_indices[b][j])][epoch] += net_out;
}
}
```



```

} while (++epoch < maxepochs);
}

/* Convert agg_errors to 1-d for message passing */
/* This is done to overcome apparent bug in n-d */
/* array message passing */
k = 0;
for (i=0; i < ntrainpat; i++)
for (j=0; j < maxepochs; j++)
{
agg_errors1d[k] = agg_errors[i][j];
k++;
}

}

/*****
*
* allocate_mem (): allocate memory
*
*****/

static void allocate_mem (int ntrainpat, int maxepochs)

{

int i, j;

```

```
/* Allocate memory */
rand_index = (int *) malloc (sizeof (float) * ntrainpat);

inter = (double **) malloc (sizeof (double *) * 3);
for (i=0; i < 3; i++)
inter[i] = (double *) malloc (sizeof (double) * (nodes[i]+1));

wgts = (double ***) malloc (sizeof (double **) * 2);
for (i=1; i < 3; i++)
{
wgts[i-1] = (double **) malloc (sizeof (double *) * nodes[i]);
for (j=0; j < nodes[i]; j++)
wgts[i-1][j] = (double *) malloc (sizeof (double) * (nodes[i-1]+1));
}

batchbuf = (double ***) malloc (sizeof (double **) * 2);
wdiff = (double ***) malloc (sizeof (double **) * 2);
for (i=1; i < 3; i++)
{
batchbuf[i-1] = (double **) malloc (sizeof (double *) * nodes[i]);
wdiff[i-1] = (double **) malloc (sizeof (double *) * nodes[i]);
for (j=0; j < nodes[i]; j++)
{
batchbuf[i-1][j] = (double *)
malloc (sizeof (double) * (nodes[i-1]+1));
wdiff[i-1][j] = (double *)
malloc (sizeof (double) * (nodes[i-1]+1));
}
}
```

```
}

idiff = (double **) malloc (sizeof (double *) * 3);
for (i=0; i < 3; i++)
idiff[i] = (double *) malloc (sizeof (double) * (nodes[i]+1));

agg_errors = (double **) malloc (sizeof (double *) * ntrainpat);
for (i=0; i < ntrainpat; i++)
agg_errors[i] = (double *) malloc (sizeof (double) * maxepochs);

agg_errors1d = (double *) malloc (sizeof (double) * (ntrainpat*maxepochs));

}

/*****
*
* neural.c : File of NN functions
*
* John Carney 16/3/98
*
*****/

#include <math.h>
#include <stdlib.h>

/* Function prototypes */
```

```

void init_weights (int ninput);
double feed_forward (double *in);
void compute_difference (double actual_out);
void propagate_backwards ();
void update_weights (int epoch);
static double momentum (int epoch);
void setup_index ();

/* External variables */
extern int ntrainpat;
extern double mom;
extern double lrate;
extern double **inter;
extern double ***wgts;
extern double ***wdiff;
extern double **idiff;
extern int *nodes;
extern double ***batchbuf;
extern int *rand_index;

/*****
*
* init_weights () : Initialise network weights
*
*****/

void init_weights (int ninput)

```



```
{

int i, j, k;
double beta, v_old;

/* Basis for learning thresholds */
for (i=0; i < 3; i++)
inter[i][nodes[i]] = 1.0;

/* Randomise weights */
beta = 0.7*( pow((double)nodes[1], (1.0/((double)ninput))) );

for (i=1; i < 3; i++)
for (j=0; j < nodes[i]; j++)
for (k=0; k < nodes[i-1]; k++)
{
wgts[i-1][j][k] =
(((double) (random()%65535))/65535)-0.5);
wdiff[i-1][j][k] = 0.0;
}

for (j=0; j < nodes[1]; j++)
{
v_old = 0.0;
for (i=0; i < nodes[0]; i++)
v_old += pow( wgts[0][j][i], 2.0 );
v_old = sqrt(v_old);

for (k=0; k < nodes[0]; k++)
wgts[0][j][k] = (beta * wgts[0][j][k]) / v_old;
```

```
}
```

```
}
```

```
/*  
*  
* feed_forward () : Feed input values forward  
*   to output  
*  
******/
```

```
double feed_forward (double *in)
```

```
{
```

```
int i, j, k;
```

```
double net;
```

```
double net_out;
```

```
for (k=0; k < nodes[0]; k++)
```

```
inter[0][k] = in[k];
```

```
/* Feed forward */
```

```
for (i=1; i < 3; i++)
```

```
{
```

```
for (j=0; j < nodes[i]; j++)
```

```
{
```

```
net = 0.0;
```

```
for (k=0; k < nodes[i-1]+1; k++)
```

```
net += wgts[i-1][j][k] * inter[i-1][k];
/* Thresholding */
inter[i][j] = (1.0-exp(-2.0*net))/(1.0+exp(-2.0*net));
}
}
for (k=0; k < nodes[2]; k++)
net_out = inter[2][k];

return (net_out);

}
```

```
/******
```

```
*
```

```
* compute_difference () : Compute difference
```

```
* between desired and
```

```
* actual output
```

```
*
```

```
*****/
```

```
void compute_difference (double actual_out)
```

```
{
```

```
int i;
```

```
double val;
```

```
for (i=0; i < nodes[2]; i++)
```

```

{
val = inter[2][i];
idiff[2][i] = (1+val) * (1-val) * (actual_out - val);
}
}

```

```

/*****
*
* propagate_backwards () : Propagate errors
* backward
*
*****/

```

```

void propagate_backwards ()

{

int i, j, k;
double val;

for (i=1; i >= 0; i--)
{
/* Compute weight differences */
for (j=0; j < nodes[i+1]; j++)
{
for (k=0; k < nodes[i]+1; k++)
{
batchbuf[i][j][k] = wdiff[i][j][k];

```



```

wdiff[i][j][k] = lrate*idiff[i+1][j]*inter[i][k];
}
}
/* Compute deltas */
for (j=0; j < nodes[i]; j++)
{
val = 0.0;
for (k=0; k < nodes[i+1]; k++)
val += idiff[i+1][k] * wgts[i][k][j];
idiff[i][j] = val * (1+inter[i][j]) * (1-inter[i][j]);
}
}
}

/*****
*
* update_weights () : update the weights after
*   backpropagation
*
*****/

void update_weights (int epoch)

{

int i, j, k;

for (i=1; i>=0; i--)

```

```
for (j=0; j<nodes[i+1]; j++)
for (k=0; k < nodes[i]+1; k++)
{
wgts[i][j][k] +=
wdiff[i][j][k]+(momentum(epoch) * batchbuf[i][j][k]);
}
}

/*****
*
* momentum () : incremental momentum value
*
*****/

static double momentum (int epoch)
{
if (epoch < 100) return (((double) (epoch))/100.0) * mom);
return (mom);
}

/*****
*
* setup_index ()
*
*****/
```

```
void setup_index ()
{
int firstIndex;
int secondIndex, r;
int holdingTank;

for (firstIndex = 0; firstIndex < ntrainpat; firstIndex++)
{
secondIndex = rand()%ntrainpat;
holdingTank = rand_index[firstIndex];
rand_index[firstIndex] = rand_index[secondIndex];
rand_index[secondIndex] = holdingTank;
}
}
```

```
/*
*****
*
* read.c: File for reading the parameters and
* database files
*
* c. John Carney 21/8/98
*
*****/
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Function Prototypes */
```

```
void read_params (char *pr_filename);
void read_patterns (char *db_filename);

/* External variables */
extern double **train_array;
extern double **test_array;
extern int *nodes;
extern int ntrainpat;
extern int ntestpat;
extern int ninput;
extern int nhidden_nodes;
extern int maxepochs;
extern int num_bs;
extern double lrate;
extern double mom;
extern double offset;
extern double scale;

/*****
*
* read_params () : process parameter file
*
*****/

void read_params (char *pr_filename)

{
```



```
FILE *fp;

/*Open parameter file*/
fp = fopen (pr_filename, "r");

/* Read parameters */
fscanf (fp, "%d", &ntrainpat);
fscanf (fp, "%d", &ntestpat);
fscanf (fp, "%d", &ninput);
fscanf (fp, "%d", &nhidden_nodes);
fscanf (fp, "%d", &maxepochs);
fscanf (fp, "%d", &num_bs);
fscanf (fp, "%lf", &lrate);
fscanf (fp, "%lf", &mom);
fscanf (fp, "%lf", &scale);
fscanf (fp, "%lf", &offset);

fclose (fp);

}

/*****
*
* read_patterns () : read patterns from database
*
*****/
```

```
void read_patterns (char *db_filename)
{

FILE *fp;

int i, j;

/* Allocate memory */
train_array = (double **) malloc (sizeof (double *) * (ntrainpat));
for (i=0; i < ntrainpat; i++)
train_array[i] = (double *) malloc (sizeof (double) * (ninput+1));
test_array = (double **) malloc (sizeof (double *) * (ntestpat));
for (i=0; i < ntestpat; i++)
test_array[i] = (double *) malloc (sizeof (double) * (ninput+1));
nodes = (int *) malloc (sizeof (int) * 3);

/* Initialise nodes */
nodes[0] = ninput;
nodes[1] = nhidden_nodes;
nodes[2] = 1;

/* Read data */
fp = fopen (db_filename, "r");
for (i=0; i < ntrainpat; i++)
```

```
{
for (j=0; j < (ninput+1); j++)
{
fscanf (fp, "%lf", &train_array[i][j]);
}
}
fclose (fp);

fp = fopen ("test.dat", "r");
for (i=0; i < ntestpat; i++)
{
for (j=0; j < (ninput+1); j++)
{
fscanf (fp, "%lf", &test_array[i][j]);
}
}

fclose (fp);

}

/*****
*
* train.c : Once we have found the
*           optimal epochs we train the nets
*
* c. John Carney 21/8/98
*
*****/
```

```
*****/
```

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "neural.h"
```

```
/* Function prototypes */
```

```
void train_bs_ensemble (int start_index, int bs_partit, int taskid);
```

```
/* External variables */
```

```
extern int *rand_index;
extern int **bs_indices;
extern double **train_array;
extern double **test_array;
extern int *stop_epoch;
extern int *nodes;
extern double ***wgts;
extern int ntrainpat;
extern int ntestpat;
extern int ninput;
extern int nlayer;
extern double lrate;
extern double mom;
```

```
*****/
```



```
*
* train_bs_ensemble ()
*
*****/

void train_bs_ensemble (int start_index, int bs_partit, int taskid)

{

int b, i, j, k;
int epoch;
double actual_out;
double net_out;
double *in;
FILE *fp;

char savefile[32];

/* We need a different save_weights file for each task */
sprintf (savefile, "%s.%d", "weights", taskid);

/* Open save network weights file */
fp = fopen (savefile, "w");

/* Allocate memory for in */
in = (double *) malloc (sizeof (double) * ninput);

/* Train each bootstrap re-sample */
```

```
for (b=start_index; b < (start_index+bs_partit); b++)
{

if (taskid == 0)
printf ("\nMASTER: Training network %d", (b+1));

init_weights (ninput);
epoch = 0;

for (i=0; i < ntrainpat; i++)
rand_index[i] = bs_indices[b][i];

/* Train and test 1 network */
do
{
/* Present training patterns */
for (j=0; j < ntrainpat; j++)
{
for (k=0; k < ninput; k++)
in[k] = train_array[(rand_index[j])][k];
actual_out = train_array[(rand_index[j])][ninput];
net_out = feed_forward (in);
compute_difference (actual_out);
propagate_backwards ();
update_weights (epoch);
}
} while (++epoch < stop_epoch[b]);

/* Save weights */
```

```
for (i=1; i < nlayer; i++)
{
for (j=0; j < nodes[i]; j++)
{
for (k=0; k < nodes[i-1]+1; k++)
{
fprintf (fp, "%lf ", wgts[i-1][j][k]);
}
fprintf (fp, "\n");
}
fprintf (fp, "\n");
}

}

fclose (fp);

}

/*****
*
* bagmain.h : header for bagmain.c
*
* c. John Carney 10/9/98
*
*****/

void create_bs_datasets ();
void create_error_matrix (int start_index, int bs_partit, int taskid);
```

```
void read_params (char *pr_filename);
void read_patterns (char *db_filename);
void combine_errors ();
void train_bs_ensemble (int start_index, int bs_partit, int taskid);
void calc_val_res (int numtasks);
```

```
/******
```

```
*
```

```
* errors.h: Header file for errors.c
```

```
*
```

```
* c. John Carney 21/8/98
```

```
*
```

```
*****/
```

```
void init_weights (int ninput);
double feed_forward (double *in);
void propagate_backwards ();
void update_weights (int epoch);
void compute_difference (double actual_out);
```

```
/******
```

```
*
```

```
* train.h: Header file for train.c
```

```
*
```

```
* c. John Carney 21/8/98
```

```
*
```

```
*****/
```



```
void init_weights();  
double feed_forward();  
void compute_difference(double actual_out);  
void propagate_backwards();  
void update_weights(int epoch);
```

# Bibliography

(Alexander, 1998) C. Alexander, editor. *Risk Management and Analysis*, John Wiley and Sons, Chichester, England, 1998.

(Aussem *et al.*, 1998) A. Aussem, J. Campbell and F. Murtagh. S&P500 forecasting using a neuro-wavelet approach. *Journal of Computational Intelligence in Finance* 6:5-12, 1998.

(Bachelier, 1900) L. Bachelier. Theory of Speculation. In P. Cootner editor, *The Random Character of Stock Market Prices*, MIT Press, Cambridge, MA, 1964, reprint.

(Baum and Haussler, 1988) E. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation* 1:151-160, 1988.

(Bickel and Freedman, 1981) P. Bickel and D. Freedman. Some asymptotic theory for the bootstrap, *Annals of Statistics*, 9:1196-1217, 1981.

(Bishop, 1994) C.M. Bishop. Mixture density networks. Technical report, Department of Computer Science and Applied Mathematics, Aston University, Birmingham, UK, 1994.

(Blume *et al.*, 1994) L. Blume, D. Easley and M. O'Hara. Market statistics and technical analysis: The role of volume, *Journal of Finance*, 49:153-181, 1994.

(Breiman, 1996a) L. Breiman. Bagging predictors. *Machine Learning*, 24:123-140, 1996.

(Breiman, 1996b) L. Breiman. Out-of-bag estimation. Technical report, Statistics Department, University of California at Berkeley, California, 1996.

(Breiman, 1996c) L. Breiman. Bias, variance and arcing classifiers. Technical report, Statistics Department, University of California at Berkeley, California, 1996.

(Breiman, 1994) L. Breiman. Heuristics of instability in model selection. Technical report, Statistics Department, University of California at Berkeley, California, 1994.

(Brock *et al.*, 1992) W. Brock, J. Lakonishok and B. LeBaron. Simple technical trading rules and the stochastic properties of stock returns, *Journal of Finance*, 47:1731-1764, 1992.

(Campbell *et al.*, 1997) J. Campbell, A. Lo and A.C. MacKinlay. *The Econometrics of Financial Markets*, Princeton University Press, Princeton, New Jersey, 1997.

(Carney and Cunningham, 1999a) J.G. Carney and P. Cunningham. The NeuralBAG algorithm: Optimizing generalization performance in bagged neural networks. In M. Verleysen, editor, *Proceedings of the 7th European Symposium on Artificial Neural Networks*, 135-140, D-Facto, Brussels, 1999.

(Carney and Cunningham, 1999b) J.G. Carney and P. Cunningham. Tuning diversity in bagged ensembles. Technical report TCD-CS-1999-44, Department of Computer Science, University of Dublin, Trinity College, 1999. To appear in *International Journal of Neural Systems*.

(Carney *et al.*, 1999) J.G. Carney, P. Cunningham and U. Bhagwan. Confidence and prediction intervals for neural network ensembles. In *Proceedings of the International Joint Conference on Neural Networks 1999*, paper 2090 (CD-ROM volume), Washington DC, 1999.

(Carret, 1997) P. Carret. *The Art of Speculation*, John Wiley and Sons, New York, 1997.

(Corden, 1995) W.M. Corden. *Economic Policy, Exchange Rates and the International System*, University of Chicago Press, Chicago, 1995.

(Cowles, 1933) A. Cowles. Can stock-market forecasters forecast? *Econometrica*, 1:309-324, 1933.

(Cybenko, 1989) G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematical Control Signals Systems*, 2:303-314, 1989.

(Davidson and Hinkley, 1997) A. Davidson and D. Hinkley. *Bootstrap Methods and their Application*, Cambridge University Press, 1997.

(DiCicco and Tibshirani, 1987) T. DiCicco and R. Tibshirani. Bootstrap confidence intervals and bootstrap approximations, *Journal of the American Statistical Association*, 82:163-170, 1987.

(Drucker *et al.*, 1993) H. Drucker, R. Schairpe and P. Simard. Improving performance in neural networks using a boosting algorithm. In S.J. Hanson, J.D. Cowen and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, 42-49, Morgan Kaufman, 1993.



(Efron, 1979) B. Efron. Bootstrap methods: Another look at the jackknife, *Annals of Statistics*, 7:1-26, 1979.

(Efron and Tibshirani, 1993) B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*, Chapman and Hall, London, 1993.

(Efron and Tibshirani, 1995) B. Efron and R. Tibshirani. Cross-validation and the bootstrap: Estimating the error rate of a prediction rule. Technical report, Statistics Department, Stanford University, 1995.

(Einstein, 1905) A. Einstein. Ueber die von der molekular-kinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Annalen der Physik*, 17:549-560, 1905.

(Farmer, 1998) D. Farmer. Market force, ecology and evolution. Technical report 98-12-119E, Santa Fe Institute, 1998.

(Fraser and Dimitriadis, 1994) A.M. Fraser and A. Dimitriadis. Forecasting probability densities by using hidden Markov models with mixed states. In A. Weigend and N. Gershenfeld, editors, *Time-Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley, Reading, MA, 1994.

(Freidman, 1991) J. Freidman. Multivariate adaptive regression splines (with discussion), *Annals of Statistics*, 19:1-141, 1991.

(Freund and Schapire, 1996) Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, 148-156, Morgan Kaufman, 1996.

(Freund and Schapire, 1995) Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory* , Springer-Verlag, 23-37, 1995.

(Gallant *et al.*, 1991) R. Gallant, P. Rossi and G. Tauchen. Stock prices and volume, *Review of Financial Studies*, 5:199-242, 1991.

(Geman *et al.*, 1992) S. Geman, E. Bienenstock and Rene Doursat. Neural networks and the bias/variance dilemma, *Neural Computation*, 4:1-58, 1992.

(Granger, 1989) C.W. Granger. Combining forecasts – twenty years later, *Journal of Forecasting* 8:167-173, 1989.

(Granger and Morgenstern, 1970) C. Granger and O. Morgenstern. *Predictability of Stock-Market Prices*, D.C. Heath and Company, Lexington, MA, 1970.

(Hamilton, 1989) J. Hamilton. A new approach to the economic analysis of nonstationary time-series and the business cycle, *Econometrica*, 57:257-384, 1989.

(Hassibi and Stork, 1993) B. Hassibi and D. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In S.J. Hanson, J.D. Cowen and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5* , 164-171, Morgan Kaufman, 1993.

(Hertz *et al.*, 1991) J. Hertz, A. Krogh and R. Palmer. *Introduction to the Theory of Neural Computation* , Addison-Wesley, Redwood City, CA, 1991.

(Heskes, 1997a) T. Heskes. Balancing between bumping and bagging. In M. Mozer, M. Jordan and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, 466-472, MIT Press, 1997.

(Heskes, 1997b) T. Heskes. Practical confidence and prediction intervals. In M. Mozer, M. Jordan and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, 176-182, MIT Press, 1997.

(Hjorth, 1994) U. Hjorth. *Computer Intensive Statistical Methods*, Chapman and Hall, London, 1994.

(Huang *et al.*, 1990) X. Huang, K.F. Lee and H.W. Hon. On semi-continuous hidden Markov modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 689-692, 1990.

(Husmeier, 1999) D. Husmeier. *Neural Networks for Conditional Probability Estimation*, Springer-Verlag, London, 1999.

(Ito, 1993) Y. Ito. Extension of approximation capability of three layered neural networks to derivatives. In *Proceedings of the International Conference on Neural Networks 1993*, 377-381, 1993.

(Karpoff, 1986) J. Karpoff. The relation between orice changes and trading volume: A survey, *Journal of Quantitative and Financial Analysis*, 22:109-126, 1986.

(Keim, 1989) D. Keim. Trading patterns, bid-ask spreads and estimated security returns: The case of common stocks and at calendar turning points, *Journal of financial Economics*, 37:371-398, 1989.

(Kendall, 1953) M.G. Kendall. The analysis of economic time-series – part I: Prices, *Journal of the Royal Statistical Society*, 96:11-25, 1953.

(Kennan and O'Brien, 1993) D. Kennan and M. O'Brien. Competition, collusion and chaos, *Journal of Economic Dynamics and Control*, 17:327-353, 1993.

(Krogh and Vedelsby, 1995) A. Krogh and J. Vedelsby. Neural network ensembles, cross-validation and active learning. In G. Tesauero, D. Touretzky and T. Lean, editors, *Advances in Neural Information Processing Systems 7*, 231-238, MIT Press, 1995.

(LeBaron, 1996) B. LeBaron. Technical trading rule profitability and foreign exchange intervention, working paper 5505, NBER, Cambridge, MA, 1996.

(Le Cun *et al.*, 1990) Y. Le Cun, J. Denker and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, 598-605, Morgan Kaufman, 1990.

(Lorenz, 1963) E. Lorenz. Deterministic aperiodic flow, *Journal of atmospheric sciences*, 20:130-141, 1963.

(Lucas, 1978) R. Lucas. Asset prices in an exchange economy, *Econometrica*, 46:1429-1446, 1978.

(Maclin and Opitz, 1997) R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, Rhode Island, 1997.



(Maclin and Shavlik, 1995) R. Maclin and J. Shavlik. Using competitive learning to initialize neural networks. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* , 524-530, 1995.

(Malkiel, 1992) B. Malkiel. Efficient market hypothesis. In P. Newman, M. Milgate and J. Eatwell editors, *New Palgrave Dictionary of Money and Finance*, Macmillan, London, 1992.

(Mandelbrot, 1963) B. Mandelbrot. The variation of certain speculative prices. *Journal of Business*, 36:394-419, 1963.

(Mukherjee *et al.*, 1997) S. Mukherjee, E. Osuna and F. Girosi. Nonlinear prediction of chaotic time-series using support vector machines. In *Proceedings of Neural Networks for Signal Processing 1997*, Amelia Island, FL, 1997.

(Murphy, 1986) J. Murphy. *Technical Analysis of the Futures Markets*, New York Institute of Finance, New York, 1986.

(Murphy and Aha, 1995) P. Murphy and D. Aha, librarians, The UCI Repository of Machine Learning Databases and Domain Theories, <http://www.ics.uci.edu/mlearn/MLRepository.html> , 1995.

(Neal, 1996) R.M. Neal. *Bayesian Learning for Neural Networks*, Springer-Verlag, New York, 1996.

(Nilsson, 1965) N.J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems* , McGraw Hill, New York, 1965.

(Opitz and Shavlik, 1996) D. Opitz and J. Shavlik. Generating accurate and diverse members of a neural network ensemble. In D. Touretzky, M. Mozer and M. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, 534-541, MIT Press, 1996.

(Pesaron and Potter, 1992) M. Pesaron and S. Potter. Nonlinear dynamics and econometrics: An introduction, *Journal of Applied Econometrics*, 7(supp):S1-S8, 1992.

(Prechelt, 1994a) L. Prechelt. Proben1: A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, Germany, 1994.

(Prechelt, 1994b) L. Prechelt, librarian, Proben1: A set of benchmarks and benchmarking rules for neural network training algorithms, <ftp://ftp.ira.uka.de/pub/neuron>, 1994.

(Rao and Tibshirani, 1997) J. Rao and R. Tibshirani. The out-of-bootstrap method for model averaging and selection. Technical Report, Statistics Department, Stanford University, 1997.

(Reed, 1993) R. Reed. Pruning algorithms – a survey, *IEEE Transactions on Neural Networks* 4:740-747, 1993.

(Refenes, 1994) A. Refenes, editor. *Neural Networks in the Capital Markets*, 1994, John Wiley and Sons, Chichester.

(Samuelson, 1965) P. Samuelson. Proof that properly anticipated prices fluctuate randomly. *Industrial Management Review*, 6:41-49, 1965.

(Scheinkman and Woodford, 1994) J. Scheinkman and M. Woodford. Self-organized criticality and economic fluctuations, *American Economic Review*, 84:417-421, 1994.

(Sclove, 1983) S. Sclove. Time-series segmentation: A model and a method, *Information Sciences*, 29:7-25, 1983.

(Sharkey, 1999) A.J. Sharkey, editor. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, Springer-Verlag, London, 1999.

(Sollich and Krogh, 1996) P. Sollich and A. Krogh. Learning with ensembles: How over-fitting can be useful. In D. Touretzky, M. Mozer and M. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, 190-196, MIT Press, 1996.

(Soros, 1987) G. Soros. *The Alchemy of Finance*, John Wiley and Sons, New York, 1987.

(Stone, 1974) M. Stone. Cross-validatory choice and assesment of statistical prediction, *Journal of the Royal Statistical Society* 36:111-147, 1974.

(Starck *et al.*, 1998) J.L. Starck, F. Murtagh and A. Bijaoui. *Image Processing and Data Analysis: The Multi-Scale Approach*, Cambridge University Press, 1998.

(Vapnik, 1995) V. Vapnik. *The Nature of Statistical Learning Theory*, Springer-Verlag, London, 1995.

(Wahba and Wold, 1975) G. Wahba and S. Wold. A completely automatic French curve, *Commun. Statist.*, 4:1-17, 1975.

(Wang *et al.*, 1994) C. Wang, S.S. Venkatesh and J.S. Judd. Optimal stopping and effective machine complexity in learning. In *Advances in Neural Information Processing Systems 6*, 303-310, MIT Press, 1994.

(Weigend and Shi, 1998) A. Weigend and S. Shi. Predicting daily probability distributions of S&P500 returns. Working paper IS-98-33, Department of Information Systems, Stern School of Business, New York University, 1998.

(White, 1988a) H. White. Multilayer feedforward networks can learn arbitrary mappings: Connectionist non-parametric regression with automatic and semi-automatic determination of network complexity. University of California, San Diego, Department of Economics discussion paper, 1988.

(White, 1988b) H. White. Economic prediction using neural networks: the case of IBM daily stock returns. University of California, San Diego, Department of Economics discussion paper, 1988.

(Wolpert and Macready, 1996a) D. Wolpert and W. Macready. Combining stacking with bagging to improve a learning algorithm. Technical report, Santa Fe Institute, 1996.

(Wolpert and Macready, 1996b) D. Wolpert and W. Macready. An efficient method to estimate bagging's generalization error. Technical report 96-06-038, Santa Fe Institute, 1996.

(Wolpert, 1992) D. H. Wolpert. Stacked generalization, *Neural Networks* , 8:1341-1390, 1996.



(Zangari, 1996) P. Zangari. Estimation and forecast. In *Riskmetrics – Technical Document*, J.P. Morgan, New York, 1996.

(Zapranis and Refenes, 1999) A. Zapranis and A.P. Refenes. *Principles of Neural Model Identification, Selection and Adequacy*, Springer-Verlag, London, 1999.

(Zenobi, 1999) G. Zenobi. Technical report TCD-CS-1999-76, Department of Computer Science, University of Dublin, Trinity College, 1999.

(Zhang, 1999) J. Zhang. Developing robust non-linear models through bootstrap aggregated neural networks, *Neurocomputing* , 25:93-113, 1999.