American Option Pricing Using Computational Intelligence Methods

Michael Maio Pires

A research report submitted to the Faculty of Engineering and the Built Environment, of University of the Witwatersrand, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, April 2005

Declaration

I declare that this is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg, South Africa. It has not been submitted before for any degree or examination at any other university.

Signed this _____ day of _____ 2005

_____

Michael Maio Pires

Summary

An option is the right to buy or sell an underlying asset at a future date by fixing the price now. The field of option pricing produces a challenge because of the complexity with pricing American styled options which cannot be done by the Black-Scholes equations. Neural Networks and Machine Learning techniques are predictors based on past data and it is intuitive to believe that they can model American options as they are non-linear instruments. Call option data on the South African All Share Index (ALSI) was used for testing of the techniques. These two different techniques were compared. What was also done was the comparison of Bayesian techniques applied to both the techniques. What this provided was confidence levels for the predictions. The investigations showed that Machine Learning techniques out-performed Neural Networks. The investigations also showed that there is scope for work to be done to improve the model.

Acknowledgements

I wish to thank the following person for his contributions to this project:

Prof. Tshilidzi Marwala, School of Electrical and information Engineering, University of the Witwatersrand. *For his guidance through the process, for his insight into computational intelligence and for his recommendations on the publications for the conferences attended and the Journal submitted to.*

I would also like to thank the *National Research Foundation (NRF)* for their financial support throughout the coding and writing up of the research.

Foreword

This dissertation is presented to the University of the Witwatersrand, Johannesburg, South Africa for the degree of Master of Science in Engineering.

The dissertation is entitled "American Option Pricing Using Computational Intelligence Methods." It is comprised of three Sections (all of which were written as papers for submissions to conferences and journals), "American Option pricing Using Multi-Layer Perceptron and Support Vector Machine", "American Option Pricing Using Bayesian Multi-Layer Perceptrons and Bayesian Support Vector Machines" and "Computational Intelligence Methods for American Option Pricing". The first paper was presented at the *IEEE International Conference on Systems, Man and Cybernetics* in 2004 at the Hague in the Netherlands, the second paper has been presented at the *IEEE 3rd International Conference on Computational Cybernetics* in 2005 in Mauritius and the third paper has been submitted to *the Journal of Derivatives* (a financial journal) for a journal publication. The first two papers present two different topics and the third paper consolidates the first two topics and also delves into the implications of the results from a financial point of view.

This document complies with the university's so-called "paper model" format. The Appendices provide some financial background as well as an additional paper that was written for a conference. This fourth paper is included for the purposes of reference when consolidating results for all Bayesian techniques that were attempted with Neural Networks. There is also an Appendix explaining the software written which is on the attached CD.

Appendix A is a background Section on options and how they can be used to hedge against financial risk.

Appendix B is a paper written for the *Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa* and is entitled "Option Pricing Using Bayesian Neural Networks". It was presented at this conference between the 25th and 26th of November 2004 in Cape Town, South Africa.

Appendix C introduces the files written for all the implementations, the result files that were obtained and code that was written to sort the data obtained into the form required. These are all stored on the attached CD. Included on the CD are the NETLAB and SVM toolboxes used with MATLAB® and the raw data obtained off the South African Futures Exchange (SAFEX) website.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

An option is a financial derivative of another financial security such as, a stock, an interest rate, a commodity or an exchange rate. Either one of these four securities can be the underlying asset of an option. That is why options are known as derivatives because they are derived from other financial securities [1]. An option gives the owner of the option the right, not the obligation, to buy or sell the underlying asset at a later date (known as the maturity date) but by agreeing on a price for the asset now (known as the strike price). Options are very valuable and to have this right, of buy or sell, people have to pay a premium known as the price of the option. If someone wishes to purchase the underlying asset then the option is known as a call option and if he/she wishes to sell the underlying asset then this is known as a put option [2].

Options are used on a daily basis by large firms to hedge their financial risk and maybe save millions of rands. For example, consider a firm with a large amount of long-term debt. This company's financial risk is governed by the interest rate that the bank charges the firm and in most cases this interest rate is governed by the prime rate. This prime rate changes according to the financial policy of the Reserve Bank of South Africa and so if the interest rate increases then the firm will have to pay more money in interest rate payments. What if the interest rate keeps climbing? The company will then carry on paying more and more in interest payments. A firm can generally use an option to protect itself by purchasing a call option and thus giving the firm the option of a lower interest rate. This also allows the company to not exercise the option if the interest rate drops below current levels and thus allows the company to enjoy the benefits of the favourable situation in the market. This process is known as hedging and makes options very valuable and thus companies have to pay a premium known as the price of the option. There are other financial instruments that can be used for hedging. These are forwards and futures [1].

There are two classes of options. There are European and American options. European options only allow the owner to exercise them on the expiry date of the contract (the maturity date). American options allow the owners to exercise the option on any date between accepting the contract and the expiry date of the contract. This makes American options much more valuable and also this introduces a second random process into the model. The second random process comes about because it is not known when it will be optimal to exercise the option [3].

In 1973, Fischer Black, Myron Scholes and Robert Merton formulated what is known as the Black-Scholes model for option pricing. The paper launched the field of financial engineering and provided the original option pricing formula. The model has one inherent assumption: it only holds for European options for dividend paying assets [4]. However, it did bring several insights. Most importantly, it allowed people to see what it is that influences option prices and it is assumed in this work that the same factors would influence American options. These factors are: price of the underlying asset at time zero, the option's strike price, the time to maturity from when the option is purchased, the volatility of the underlying asset and the continuously compounded risk-free rate of interest [2]. The risk-free rate of interest is the return that investors can expect risk-free and is usually denoted by the rate offered by government bonds. What was significant about the factors extracted is that the expected return of the underlying asset does not affect the option's price and this lead to the useful result known as risk-neutral valuation [2]. With these factors in mind, the model provided specific equations to price European put and call options. The Black-Scholes model also provided a means of estimating the underlying asset's volatility which can be estimated using the Black-Scholes equations. Significantly the Black-Scholes model laid the foundation for future work and this was shown as now the model has been extended for dividend paying assets [2]. Most importantly the model provided the first accurate options pricing formula and made a huge influence on the way that traders price and hedge options all over the world even today. Option pricing factors and hedging theory are explained in more detail in Appendix A. American options are what are used mostly by traders in South Africa and there is no standard for pricing them. In fact many traders use the Black-Scholes model together with a sampling process (again showing the significance of the Black-Scholes model). This means that different options traders will provide different prices and this makes the price of American options susceptible to market forces.

Computational Intelligence has been used recently in many fields and is principally used in any field where there is a degree of prediction required. For example, in an automated control environment, a controller may be required to perform some control by knowing what may happen in the future and then controlling that, based on what is known. This is known as predictive control. It has been used extensively in pattern recognition especially in the imaging sector [5]. It has also been used extensively to predict human behaviour such as predicting interstate conflict [6]. Computational Intelligence methods have a good ability to model non-linear data and it has been stated by Hornick et. al. that neural networks are universal approximators under certain conditions [7]. Computational Intelligence methods comprise machine learning techniques such as neural networks and other machine learning techniques such as Support Vector Machines. Both techniques predict the future based on the past data that they are fed and are thus able to make predictions based on past patterns.

What is proposed here is to use various computational intelligence methods in options pricing. This is a financial field that has drawn much interest especially since pricing options can be very difficult given their nature and the fact that the data is very non-linear and the fact that American options are susceptible to market forces and are not easily priced. Chapter 2 introduces two computational intelligence methods used to price options (namely the Multi-Layer Perceptron and Support Vector Machines) and shows the results obtained. Chapter 3 introduces the Bayesian framework and its application to both the techniques introduced in Chapter 2. Chapter 4 rounds up all the findings of the various techniques as well as the Bayesian framework and also introduces the usefulness and advantages of the Bayesian framework. Chapters 2, 3 and 4 are written in paper format and

have their own reference lists. Chapter 5 concludes with the techniques that showed the best results in terms of error analysis and time to train the techniques. Appendix A provides some background for options and how they are used. Appendix B provides another paper that was written and presented and is included as a means of reference for Chapter 3. Appendix C provides an explanation of the software written and how it can be used to produce the results obtained. From this thesis the following papers were published or submitted for publication:

1. M. M. Pires and T. Marwala, "American Option Pricing Using Multi-Layer Perceptron and Support Vector Machine", *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* October 10-13 2004, Amsterdam, Netherlands, pp. 1279-1285.
2. M. M. Pires and T. Marwala, "American Option Pricing Using Bayesian Multi-Layer Perceptrons and Bayesian Support Vector Machines", *Proceedings of the IEEE 3rd International Conference on Computational Cybernetics,* April 13-16 2005, Hotel Le Victoria, Mauritius, pp. 219-224.
3. M. M. Pires and T. Marwala, "American Option Pricing Using Bayesian Neural Networks", *Proceedings of the Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa,* November 25-26 2004, Cape Town, South Africa, pp. 161-166.
4. M. M. Pires and T. Marwala, "Computational Intelligence Methods for American Option Pricing", *Journal of Derivatives*, submitted December 2004.

REFERENCES

[1] S. Ross, R. W. Westerfield, B. D. Jordan and C. Firer, *Fundamentals of Corporate Finance 2nd South African Edition*, McGraw-Hill Book Company, Sydney, Australia: 2001.

[2] J. C. Hull, *Options, Futures and Other Derivatives*, Prentice Hall, Upper Saddle River, New Jersey, U.S.A.: 2003.

[3] R. A. Jarrow and S. M. Turnbull, *Derivative Securities 2nd Edition*, South-Western College Publishing, U.S.A.: 2000.

[4] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, vol. 81, 1973, pp. 637-659.

[5] B. D. Ripley, *Pattern Recognition and Neural Networks,* Cambridge University Press, 1996.

[6] T. Marwala and M. Lagazio, "Modelling and Controlling Interstate Conflict", *IEEE International Joint Conference on Neural Networks,* July 25-29 2004, Budapest, Hungary, pp. 1233-1238.

[7] K. M. Hornick, M. Sticnhcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol. 2, No. 5, 1999, pp. 359-366.

Chapter 2: American Option Pricing Using Multi-Layer Perceptron and Support Vector Machine

M. M. PIRES
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg, South Africa
m.pires@ee.wits.ac.za

Abstract – **An option is the right to buy or sell an underlying asset at a future date. The field of option pricing produces a challenge because of the complexity with pricing American styled options which cannot be done by the Black-Scholes equations for option pricing. A Multi-Layer Perceptron neural network has been used before to price these options with limited success. In this paper we will compare the performance of a Multi-Layer Perceptron neural network and a Support Vector Machine in pricing American styled options. It was found that a Support Vector Machine approach provided much better results than that found with Multi-Layer Perceptrons.**

I. INTRODUCTION

This paper deals with the application of a Multi-Layer Perceptron (MLP) neural network and a Support Vector Machine (SVM), trained using the maximum likelihood approach, for pricing an American styled option.

Firstly what needs to be dealt with is what an option is. An option is the right to buy or sell some underlying asset at a later date (known as the maturity date) by fixing the price of the asset now [5]. If the person is given the right to sell the asset then the option is known as a put option and if the person is given the right to buy the asset then the option is known as a call option [5]. Thus a person can make or lose money but that is the risk taken. To be able to have this option, the person must pay a fee and the price of the actual option is something that is not easily obtained. Black et. al. [1] made a breakthrough in the options pricing field and established what is known as the Black-Scholes model for option pricing. They formulated the option price but certain assumptions were made, the most important of which is that the pricing only holds for European styled options [5]. A European option only allows the person to buy or sell the underlying asset at the maturity date of the contract. An American option allows the person to buy or sell the asset at any date up until the maturity date [5]. Thus American options allow the buyer or seller to have extra flexibility and thus are never worth less than European options [7]. Because the date that may be chosen to exercise the option, by the buyer or seller, is completely random, this introduces a second random process and thus makes the pricing of American options difficult [5].

So what is proposed here is to compare the use of Multi-Layer Perceptron neural network and Support Vector Machine approach to pricing American styled options. These methods are formulated in the maximum likelihood framework.

An MLP is a type of neural network (NN). Neural networks are a form of prediction tools based on the trends that have occurred in the past. For supervised learning the outputs of the neural network are predicted from the inputs. The inputs are chosen as variables that affect the outputs and this is conducted by understanding the real world interrelationships. An SVM is a machine learning technique that maps certain inputs to outputs with the use of a function and its inputs and outputs can be chosen to be the same as those for an MLP. The nature of an MLP and an SVM make them useful for research in pricing options because options rely on certain parameters that influence their price.

Comparisons between SVM's and MLP's have been made in other fields. One example is an application to financial forecasting [13]. The finding was that SVM provided a robust technique for function approximation and results were more promising than those found by Radial Basis Function (RBF) and MLP networks [13]. Another comparison was made between MLP and SVM in Glaucoma diagnosis [3]. It was found in this case that both SVM and MLP techniques gave an improvement over traditional methods used to diagnose Glaucoma.

Neural Networks have been used many times before in the pricing of options. Hutchinson, et. al. [6] were one of the first people to attempt the use of NN's with options pricing but it was attempted for European options only and it was found that the prices predicted by the Black-Scholes equations could be recovered with daily data going back two years. The model types used were multi-layer perceptrons, radial basis functions, projection pursuit and ordinary least squares. Morelli, et. al. [11] also attempted pricing with NN's and in particular MLP's and RBF's. They tried pricing European and American styled options but only published there findings for the European styled options thus indicating that their findings for the American styled options were not as promising as hoped. Kelly [10] attempted valuing American put options through NN's as well but the accuracy found was to be proportional to the actual option price and it was found that there were errors as high as 60% (which occurred at higher price values) but the results were reported to be better than certain other models used [10]. Kelly did conclude that the NN approach was able to approximate an American put option price. The purpose of this work is to see if SVM can outperform MLP and if so then it can be further researched for pricing American styled options. It is conceivable to use SVM's given their success in other fields and the fact that they have not been extensively used in the options pricing field.

II. COMPUTATIONAL INTELLIGENCE METHODS

A. The Multi-Layer Perceptron

MLP is the most widely used architecture for neural networks. In this paper a NETLAB toolbox that runs in MATLAB® was used to implement the MLP neural network [12].

In this paper two layers with full connectivity between hidden units and inputs, and between hidden units and outputs [12] were used. For the particular application, a network with three inputs and one output was constructed as can be viewed in Figure 1.

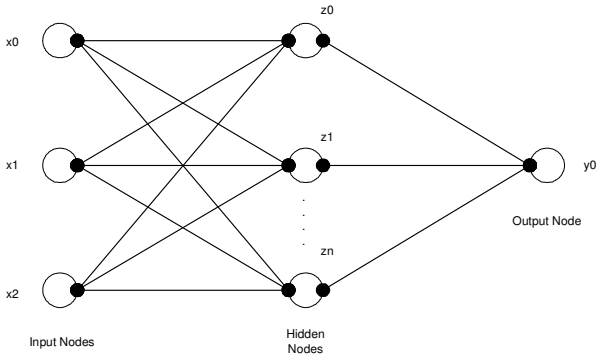

Figure 1: Architecture of the Multi-Layer Perceptron.

As can be seen in Figure 1, the number of hidden nodes is unknown and a few different amounts of hidden nodes were tried. For the first layer of the network we have activation parameters and bias variables ($a_j^{(1)}$ and $b_j^{(1)}$ respectively) associated with each hidden node. Each line (joining to different nodes) in Figure 1 has a weight associated with it ($w_{ji}^{(1)}$) and so we have:

$$a_j^{(1)} = \sum_{i=1}^{3} w_{ji}^{(1)} x_i + b_j^{(1)} \tag{1}$$

where j is the number of hidden units (n from Figure 1) and i is the number of input units. Data is fed into the network in terms of training inputs and training outputs. The activation parameters are then transformed by the non-linear activation functions of the hidden layer. There are many activation functions that can be used and in this study the hyperbolic tangent function is used for the hidden layer. The outputs of the hidden units (z_j) can thus be written as follows:

$$z_j = \tanh(a_j^{(1)}) \tag{2}$$

The z_j are then transformed by the second layer of weights ($w_{kj}^{(2)}$) and biases ($b_k^{(2)}$) to give the second layer activation functions ($a_k^{(2)}$)

$$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)} \tag{3}$$

where k is the number of output nodes (in this case $k = 1$). The second layer activation functions are then transformed through a linear function into the output and thus the output is:

$$y_k = a_k^{(2)} \tag{4}.$$

The weights at each layer are chosen using some optimisation method. The error function between the output predicted by the neural network and the target vector is constructed as follows:

$$E = \sum_{k=1}^{1} \sum_{g=1}^{N} (y_{kg} - y_{k,t\arg et})^2 \tag{5}$$

where E is the error to be minimized and N is the number of sets of training data given. The sets of outputs y_{kg} are the outputs predicted by the network given the training inputs and $y_{k,target}$ are the training outputs. The weights at each layer are changed until the error shown in equation (5) is as low as possible. Several optimisation techniques such as the conjugate gradient method, the scaled conjugate gradient method, the Quasi-Newton method and the gradient descent method may be used to minimize equation (5) [12]. Each of these methods was tried and the results of each network with different optimisation techniques were analysed.

B. Support Vector Machines

The basic idea behind support vector regression is to map the input space to an output space. Suppose we have the training data set with one input and one output being considered: $\{(x_1,y_1),\ldots,(x_l,y_l)\} \subset \chi \times \Re$, where χ is the space of the input parameters and $\Re$ denotes the real number set. We wish to find a function $f(x)$ that will map the training inputs to the training outputs. In Support Vector (SV) regression we wish to find this function that has at most ε deviation from the actual training targets y_l. We can fit several kinds of functions $f(x)$ to map training inputs to training outputs. These functions are known as kernel functions but these cannot just be any functions, kernel functions have to adhere to some criteria [8]. For the purposes of explanation we will consider a linear kernel function

$$f(x) = \langle w, x \rangle + b \qquad with \quad w \in \chi \quad , \quad b \in \Re \tag{6}$$

where $\langle .,. \rangle$ denotes the dot product.

We seek to find small values for w and one way to do this is to minimize the Euclidean norm $\|w\|^2$ [8]. We then include slack variables ξ_i, ξ_i^* so that certain infeasible constraints in the minimization of the Euclidean norm can be used and the minimization problem then becomes

$$\begin{aligned} &minimize \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &subject \quad to \quad \begin{cases} y_i - \langle w, x_i \rangle - b \le \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases} \end{aligned} \tag{7}$$

where l is the number of training points used. The constraints above deal with an ε-insensitive loss function used to penalize certain training points that are outside of the bound given by ε which is a value chosen by the user. There are various other loss functions such as the Huber loss function which can also be used but the most common is the ε-insensitive loss function [4]. The loss function is given by

$$|\xi|_\varepsilon = \begin{cases} 0 & if \quad |\xi| \le \varepsilon \\ |\xi| - \varepsilon & othererwise \end{cases} \tag{8}.$$

The value for C in equation (7) is used as the amount to which deviations from ε is tolerated [13]. It can be seen as a measure of over fitting a function too well to its training points. If the value of C is set too high then the function found ($f(x)$) will be too well fitted to the training data and will not predict very well on data that isn't seen by the training of the function. It means that points lying outside of the bounds given by ε are not penalized enough and this results in the function being too well fitted to the training points [2]. A sketch of a linear function being fitted to training data can be seen in Figure 2 with the bounds being shown.

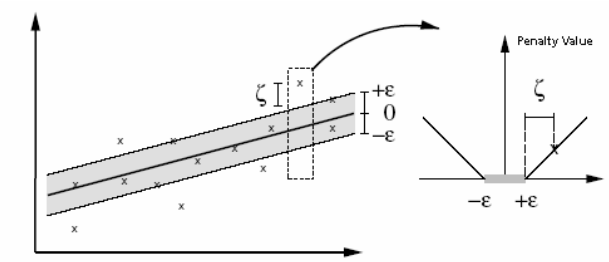

Figure 2: Linear SVM regression for a set of data (left) and the ε-insensitive loss function (right) [8].

The function on the right in Figure 2 is used to penalize those points that lie outside of the bounds shown on the left. The more a point lies outside of one of the bounds (either below or above), the more the point is penalized and thus plays less of a role in the determination of the function. Those points that fall within the bounds of the function are not penalized at all and their corresponding slack variable values (ξ_i, ξ_i^*) are given zero and thus these points will play a major contribution in the determination of the function $f(x)$.

The optimisation problem of equation (7) is then set up to be a quadratic programming problem by first finding the Lagrangian multiplier and applying the Karush-Kuhn Tucker (KKT) conditions [8]. Then the values for w and b can be found so that the linear function to fit the training data of equation (6) can be explicitly found. Note that this example using the constrained optimisation problem of equation (7) is for a linear kernel function and the constrained optimisation problem of equation (7) is different for different kernel functions.

III. Implementation and Results

A. The Multi-Layer Perceptron

Historical data was obtained from the JSE securities exchange of South Africa [9]. Call, put and future option data were obtained for the period January 2001 to December 2003. The inputs used into the network were the stock volatility, the time in days to maturity and the strike price for the option. All these factors affect European option prices [5] and so it was felt that these three parameters will also affect American styled option prices. Therefore this is the justification for these parameters being chosen as inputs to the NN remembering that inputs to a NN have to be that which affect what is trying to be predicted (see Section I). The average of the high and low prices for the option was set as the output for the network. It was decided to only consider the call option data for a particular stock as there was much data available over the three year period and not all of it was needed.

Certain parameters were changed during the training of the MLP network so that the best results could be obtained. These were: the number of training cycles (because if a network is allowed to train for too long then the network may be over trained and thus not predict outputs very well because the network will be too tuned to the training data, at the same time if too few cycles were used then the network may be under trained), the number of hidden nodes (if too few nodes or too many nodes are used then outputs are also affected because the number of nodes is an indication of the network's ability to learn but if there are too many then the NN may not learn too well) and the number of training data points used (the more data used the better but if there is too much data then training of the network can be very slow because of the optimisation process). Different optimisation techniques can also be used and the weight decay value can also be changed.

It was found that training a network with 400 data points was suitable given that the number of inputs was only 3 (if more training data was used then the training of the network seemed to produce a high error due to the optimisation not being able to handle a training set that is too big) and it was found that 10 hidden nodes sufficed for the implementation. If the number of hidden nodes was made too high then it produced quite high errors in the pricing. The number of training cycles used (to produce the best results) was found to be 300. The optimisation technique that produced the best results was the Scaled Conjugate Gradient technique [12]. The other techniques all required more iterations to obtain a network that converged to a minimum in the error function and the worst performing technique was the Gradient Descent method because it didn't converge to a result and the outputs, given out by the network trained, were not numbers at times. The weight decay value was kept at 0.05 as the network was found to perform the best with it at this value.

The results for the network trained using MLP can be seen in Table 1. As can be seen from Table 1, the maximum error and mean error parameters are quite high and may not be good measures of performance and thus the last two parameters are much more encouraging with 46% of the outputs having an error of less than 10% and 25% of the outputs having an error less than 5%. A plot of the targets and predicted values can be seen in Figure 3 but for only 100 of the test points used.

Table 1: MLP Results

TP	Max Error (%)	Mean Error (%)	Error less than 10%	Error less than 5%
300	2585	52	139	76

TP = no. of test points used, Max Error = Maximum error between the actual output and that predicted by the network in the *TP* point test set used, Mean Error = average error of the size of the test set used (TP), Error less than 10% = amount of outputs given by the network that fell within an error of 10%, Error less than 5% = amount of outputs given by the network that fell within an error of 5%.

Figure 3 shows that the exact trends (i.e. if the target value goes up then so does the predicted value) are followed and that the maximum error (in terms of price) was R1409.00 or $216.77 (at R6.50 to the U.S $). Compared to the actual price of R54.50 for the option, this error is high (but the low price for the option can be considered to be an outlier and thus the actual price of the option may also be too low) but as can be seen in Figure 3 the biggest errors occur at the peaks of the plots otherwise the trends are followed almost exactly.

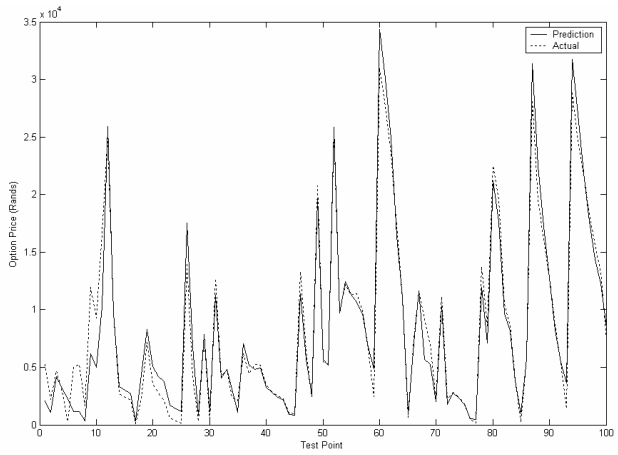

Figure 3: Outputs of the MLP NN and the actual target values used in the testing.

It was also found that due to the initial values for the weights of each layer and the initial values for the bias parameters for each layer being chosen purely randomly, the algorithm to train the network had to be run a few times with the same parameters before the solution in Table 1 was found. Initially the results were worse than that shown in Table 1 but once the algorithm was run a few times the results were found to be better.

B. Support Vector Machines

The same historical data was used as that for the MLP training. With regards to SVM there are different parameters that can be changed namely the capacity and the e-insensitivity (ε, see Section II.B), the amount of training inputs and the function to be used for the kernel.

It was found that due to the computational power needed by the SVM network training, 400 points of data sufficed for the network training otherwise it took too long to train the network. At first a polynomial kernel of degree 2 was used with 1000 data points being used to train the data but this caused the network to take 1 hour and 40 minutes to train (on a Pentium 4 1.8GHz laptop with no other software running) and the results were poorer than with only 400 training points used (with the same values for capacity and ε being used with 400 and 1000 training points). At 500 training points, the time to train the network was double that at 400 points and the results were also poorer than at 400 points.

To find the most appropriate kernel function for this problem, all the kernel functions with the same values for capacity and ε were used and the best performing kernel function was chosen.

As was stated in Section II.B, the value for capacity had to be chosen carefully so as not to over-train or under train the SVM. Various values for the capacity and for ε were experimented with and the results for the different values can be seen in Table 2.

Table 2: SVM Results

TP	Max Error (%)	Mean Error (%)	Error less than 10%	Error less than 5%	C	ε
300	3353	61	135	80	1	0.1
300	1282	35.7	135	92	10	0.01
300	1152	34.4	138	94	10	0.005
300	1356	35	140	91	10	0.001
300	1339	34.9	141	93	10	0.0005
300	1540	38.16	137	92	15	0.01
300	1338	36.2	134	95	15	0.005
300	1518	36.4	133	93	15	0.001
300	1524	36.5	131	94	15	0.0005
300	1707	38.8	128	92	20	0.001

TP = no. of test points used, Max Error = Maximum error between the actual output and that predicted by the network in the *TP* point test set used, Mean Error = average error of the size of the test set used (TP), Error less than 10% = amount of outputs given by the network that fell within an error of 10%, Error less than 5% = amount of outputs given by the network that fell within an error of 5%, C = capacity, ε = e-insensitivity.

From Table 2 there are many deductions that can be made:

1. With the capacity at 1, the network was under trained.
2. With the capacity at 20, the network was over trained.
3. The network performed the best with the capacity at 10.
4. If we consider the measure of 'mean error' (to measure performance), then the e-insensitivity should be set to 0.005.
5. If we consider the number of predicted outputs that are less than 10% and 5% (to measure performance) then the value for the e-insensitivity should be 0.0005.

When the data was normalised, the results of Table 2 were obtained. The data normalizing approach used may be represented as follows:

$$data = \frac{data - \mu}{\sigma} \tag{9}$$

where *data* is the training data set to be normalized, μ is the mean of the training data set and σ is the standard deviation of the training data set. This is done for each input and output data sets.

This normalizing approach was the same approach that was used with the training of the MLP network. The average error was found to be 34.4% at best with 400 training inputs. There were significant improvements with the use of SVM's over MLP's as will be discussed.

A plot of the SVM outputs against actual outputs (for the third network in Table 2) can be seen in Figure 4. The plot

shown in Figure 4 can show that the predictions obtained can be quite impressive. These errors are clear from Table 2.

As can be seen form Figure 4 the trends of the actual prices are followed accurately by the SVM.

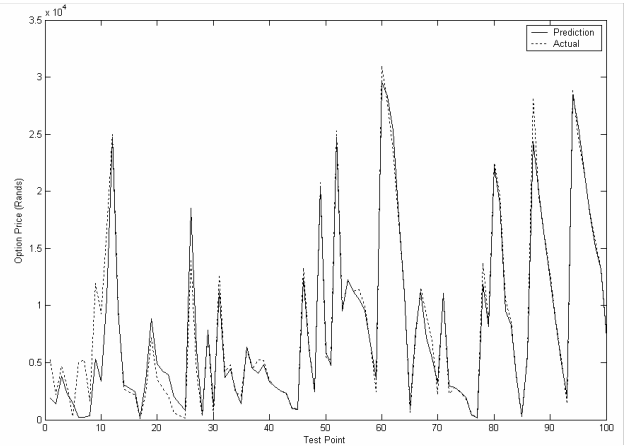

Figure 4: Outputs of the SVM and the actual target values used in the testing.

IV. COMPARISON OF MLP AND SVM

As can be seen from Table 1 and Table 2 the SVM network out-performed the MLP network because the average error was brought down to 34.4% with SVM and the number of outputs predicted that were within 5% were more than the that found with the best MLP network found. The maximum error found by the SVM for the data set was also less than that found by the MLP. The same test and training points were used for both the MLP and the SVM. Therefore the SVM outperformed the MLP even with the same data used. The MLP network took only 5 seconds to train whereas all the networks trained for SVM (from Table 2) took on average about 6 minutes and 40 seconds. This is a small price to pay for obtaining a better pricing model. Thus SVM is said to outperform MLP in the case of pricing stock options as has been shown here.

As with MLP, the outputs predicted by the SVM network also followed the exact trends of that of the actual outputs but at times some of the peaks didn't correspond and that is the reason for there being an error at times.

To improve the performance of SVM, what can be done is to use an optimisation technique to find the best possible values for capacity (C) and ε-insensitivity (ε). Due to the computational time needed to find a solution using SVM, it might be feasible to use less training points to find the best values for C and ε and then try to use SVM with more training points. To use optimisation methods with MLP (to find the best values for the number of hidden nodes and possibly some other parameters) may cause a bit of a problem because the MLP has to be trained a few times to give its best predictions given certain parameters because of its random nature (as stated in Section III.A). Therefore to use optimisation with MLP, some means of training the NN a few times and then taking its best prediction (for each set of parameters tried in the optimisation) will have to be devised thus making the problem more complicated.

Given the fact that Kelly concluded that a NN approach was successful in approximating American put options [10], it is conceivable that SVM would approximate the put option more accurately than the NN approach used by Kelly (if it were used the same way as Kelly used the NN approach). Therefore the results show that SVM can more accurately price American styled options.

V. CONCLUSION

In conclusion it can be stated that SVM networks out-perform MLP networks in the problem of American options pricing. If more data could be incorporated with the training of the network and then values for capacity and ε-insensitivity could be found (by training more networks) then the results of the network training would be more satisfactory. This can be very computationally intensive and so a more computational power may be needed. What can also be done is to use some optimisation technique with SVM to optimise the values for capacity and ε-insensitivity. This will be very computationally intensive as well but will produce better results. Another suggestion is to use more inputs in the training of an SVM. With European options, other parameters that affect price are the risk free rate of interest and the underlying asset price at the outset of purchasing the option. These two could be added as inputs to an SVM to try improve the results found but data for this would also have to be found. If other parameters could also be obtained (other than these extra two) that affect the option's price then these could also be incorporated as inputs but studies would have to be done to find extra parameters that affect American option prices.

REFERENCES

[1] F. Black, and M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, Vol 81, pp. 637-659, 1973.

[2] C. J. C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition*, Kluwer Academic Publishers, Boston, pp 1-43.

[3] K. Chan, T. Lee, P. A. Sample, M. H. Goldbaum, R. N. Weinreb, and T. J. Sejnowski, "Comparison of Machine Learning and Traditional Classifiers in Glaucoma Diagnosis", *IEEE Transaction on Biomedical Engineering*, Vol 49, No. 9, pp. 963-974, Sep. 2002.

[4] S. R. Gunn, *Support Vector Machines for Classification and Regression.* Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, University of Southampton, U.K., 1998.

[5] J. C. Hull, *Options, Futures and Other Derivatives, 5th Edition.* Prentice Hall, Upper Saddle River, New Jersey, U.S.A., 2003.

[6] J. M. Hutchinson, A. W. Lo, and T. Poggio, "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks," *Journal of Finance*, Vol 9, No. 3, pp. 851-889, 1994.

[7] R. A. Jarrow, and S. M. Turnbull, *Derivative Securities, 2nd Edition.* South-Western College Publishing, U.S.A, 2000.

[8] J. Joachims, "Making large-scale SVM learning practical" In B. Scholkopf, C.J.C. Burges and A.J. Smola, editors, *Advances in Kernel Methods-Support Vector Learning,* pp. 169-184, Cambridge, MA, 1999, MIT Press.

[9] *South African Futures Exchange,* Available: http://www.safex.co.za.

[10] D. L. Kelly, "Valuing and Hedging American Put Options Using Neural Networks," Working Paper, Carnegie Mellon University, PA, December 1994.

[11] M. J. Morelli, G. Montagna, O. Nicrosini, M. Treccani, M. Farina, and P. Amato, "Pricing Financial Derivatives with Neural Networks," *Physica A: Statistical Mechanics and Its Applications*, 1 July 2004, Vol. 338, Issues 1-2, pp. 160-165.

[12] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition,* Springer-Verlag, London, Great Britain, 2003.

[13] T. B. Trafalis, and H. Ince, "Support Vector Machine for Regression and Applications to Financial Forecasting," IEEE-INNS-ENNS International Joint Conference on Neural Networks, Como, Italy, July 24-27, 2000.

Chapter 3: American Option Pricing Using Bayesian Multi-Layer Perceptrons and Bayesian Support Vector Machines

M. M. PIRES
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg, South Africa
m.pires@ee.wits.ac.za

Abstract – **An option is the right, not the obligation, to buy or sell an underlying asset at a later date but by fixing the price of the asset now. There are European and American styled options. European styled options can be priced using the Black-Scholes equations but American options are more complex and valuable due to the second random process they introduce. Multi-Layer Perceptrons and Support Vector Machines (formulated in the maximum likelihood framework) have been used previously to price American options and what is introduced here is Bayesian Techniques to both these approaches. Bayesian techniques used with both these approaches are compared in terms of pricing accuracy and time to train each of the learning algorithms. It was found that Bayesian SVM's out-performed Bayesian MLP's and that there is scope for further work. However, Bayesian SVM's took much longer to train than Bayesian MLP's even though they produced better error results.**

I. INTRODUCTION

This paper deals with the problem of American option pricing using Bayesian techniques applied to Neural Networks (NN's) and Support Vector Machines (SVM's). Firstly what needs to be done is to introduce options and their use.

Around the world, companies are exposed to risk in a number of ways. If a company exports then they are exposed to the volatility of the exchange rate. For example, a gold mining company in South Africa is exposed to risk from the gold price because if the gold price drops then the mining company may lose money. In addition most of the gold production in South Africa is exported and so if the Rand strengthens against the dollar then the mining company will receive less Rand for the gold sold and so it is exposed to exchange rate risk as well. Mining companies wish to protect themselves against this risk and so what they do is have an agreement to sell gold at a particular fixed exchange rate and gold price for the future months. This contract is fixed and the company will not make any extra money or lose any extra money. The two contracts the company undertakes are known as futures contracts. Since the contract doesn't allow the owner to make any additional money (if the gold price increases or the Rand weakens) then the company doesn't pay a premium for the contracts. This reduction of risk is known as hedging [1]. Another way in which companies hedge against this risk is by using options.

An option is the right, not the obligation, to buy or sell an underlying asset at a later date (known as the maturity date) by fixing the price of the asset now [2]. When the option gives the owner the right to buy the asset then the option is known as a call option and when option gives the owner the right to sell the asset then the option is known as a put option. There are two kinds of options. They are European and American styled options. European options only allow exercise of the option on the maturity date and American options allow exercise on any date leading up to the maturity date. In the example above, with the gold mining company, the company could buy a commodity put option (allowing the company to sell gold at a particular price form a fixed time) and purchase an exchange rate call option (allowing the company to trade at a particular exchange rate at a later date) [3].

Options are different from futures in that the owner of the option is given the right, and not the obligation, to exercise and thus make them valuable and so companies can benefit from desirable situations in the market and still protect themselves from undesirable effects in the market [3]. The main difference from futures is that if an undesirable situation does occur then the owner will lose the premium that he/she paid to have the option. Because of this, options are bought with a premium and there is a difficulty with finding the value of this premium. Black et. al. [4] formulated pricing options and obtained equations for pricing options in 1973 and this was a very significant finding in finance. The problem with their model was that one of the assumptions was that the model only was appropriate for pricing European options [2]. American options are more difficult to price [2] because there is a second random process in the contract (as we do not know when it will be optimal to exercise the option) and they give the owner of the option an extra flexibility and are thus more valuable than European options [5].

Multi-Layer Perceptrons (MLP's) are a type of NN. NN's are a prediction tool and predict based on trends that are found from past data. It learns the trends form the information it is given and then tries and predict on data that it hasn't seen before (unseen data) based on the information it is fed. The inputs to the network are chosen as variables that could possibly affect what it is that is to be predicted (in this case what is trying to be predicted is the option price). An SVM is a machine learning technique different from NN's. It predicts also based on inputs but is rather done by finding a function that maps the inputs to the outputs. Their inputs can be chosen in the same way as that from NN's. MLP's and SVM's are an appropriate method for pricing options because as option prices depend on several factors that influence them and these can be used as inputs to the NN's and SVM's. A comparison has been made between MLP's and SVM's and it was found that SVM's performed better, for the pricing problem, than MLP's [6].

Bayesian techniques provide confidence levels in the predictions made by either NN's or SVM's so that the predictions are known to be within a certain confidence interval (68% sure if the interval is one standard deviation away form the mean (i.e. 68% sure that the actual value is within this interval), 95% sure if the interval is taken as two standard deviations from the mean and 99.7% sure for three standard devia-

tions from the mean [7]). Bayesian techniques applied to MLP's have been used in many other fields but applying the techniques to SVM's is new and has mainly been used in general applications more to test the application of the Bayesian framework to SVM's rather than actually using them for the application [8]. The sampling technique used for the Bayesian inference is the Metropolis-Hastings algorithm. An introduction to MLP's and SVM's is given and how Bayesian techniques can be applied to them is explained. The findings are then provided with the comparison between the techniques.

II. Computational Intelligence Methods

A. The Multi-Layer Perceptron

The MLP is the most widely used architecture for NN's. In this paper a toolbox was used to implement the MLP architecture. The toolbox used was the NETLAB toolbox developed by Ian Nabney [9] and was a toolbox developed for use in MATLAB®. In this toolbox the NN has two layers with full connectivity between both layers. For this particular application there were three inputs used and one output.

The number of hidden nodes to be used can vary and the performance of the network is dependent on how many are used. Therefore different amounts of hidden nodes were used until the best performing network was found.

The process of training a network is very mathematical and involves random values being given two first layer weights and bias variables. First layer activation parameters are then found from these variables and then transformed to the second layer through an activation function (in this case it was a hyperbolic tangent function). The second layer activation parameters are then found from a function of second layer bias parameters, second layer weights and the values from the activation function. These second layer activation parameters are then transformed to the output in terms of an output activation function. In this toolbox the output activation function was linear and it just involved making the output equal to the second layer activation parameters. The training of the network involves a minimization of an error function by finding the best weight values so that the predicted value of the network is close to the actual values being fed to the network as training data. The error function to be minimized is [9]:

$$E = \sum_{k=1}^{1} \sum_{g=1}^{N} (y_{kg} - y_{k,t\arg et})^2 \qquad (1)$$

where E is the error to be minimized and N is the number of sets training data given. The sets of outputs y_{kg} are the outputs predicted by the network when it is given the training inputs and $y_{k,target}$ are the actual values for the training outputs fed into the network.

B. Support Vector Machines

SVM's are a machine learning technique that attempt to map the inputs of the training data to the outputs of the training data. In this problem we wish to find the function $f(x)$ that has at most ε deviation form the training targets. There are several functions that can be fitted to map the inputs to the outputs but the functions used have to fit to certain criteria [10]. The function used is a function of two or more variables and the training inputs. One of the variables is w. It is important to seek the smallest possible values for w and we then wish to minimize the Euclidean norm $\|w\|^2$ [10]. If we say wished to use a linear function to fit the data as follows [10]:

$$f(x) = \langle w, x \rangle + b \quad with \quad w \in \chi \quad , \quad b \in \Re \qquad (2)$$

where $\langle .,. \rangle$ denotes the dot product. A constrained optimisation problem is then set up so that by the inclusion of certain slack variables ξ_i, ξ_i^*. The constrained optimisation problem is [10]:

$$minimize \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$
$$subject \quad to \quad \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \qquad (3)$$

where l is the number of training points. Equation (3) is solved by using quadratic programming, finding the Lagrangian Multiplier and applying the Karush-Kuhn Tucker (KKT) conditions [10]. The variable C is known as the capacity value and is used so that the function found does not over fit to the training data and is used as the amount to which deviations from ε are tolerated [11]. The higher the value of C the worse the algorithm will perform on unseen data. The slack variables are introduced also to prevent over training. They are given values according to a loss function. There are different loss functions used and the one most popular is the ε-insensitive loss function. Basically, if certain training inputs map to outputs that are more than ε deviation from the function (trying to be found) then those points are penalized and the more they exceed ε the more those points are penalized and considered less important by the optimisation when trying to find $f(x)$. It must be remembered that the problem provided by (3) if is a linear function (as in (2)) is used to fit the data and that (3) would be different if a different function is used to fit the data.

III. Bayesian Techniques

A. Bayesian Techniques for Neural Networks

With NN's there is always an error in the predictions made and we thus have [9]:

$$y = f(x; w) + \varepsilon \qquad (4)$$

where y is the actual output desired, f is the output predicted by the network, ε is the error, w are the weights [9] and x is a vector of inputs. Even if we are given ε and the same network is run twice with the same parameters, we will obtain different weights w both times and thus there is an uncertainty in the training of the networks [9] and this can be attributed to the randomness in the assignment of weights. Generally some complex models try to fit the noise into the predictions which

cause problems when trying to predict with unseen inputs (the problem of over training) and thus cause there to be even more error in the predictions [9].

The parameter *p(.)* wherever used from now on denotes the probability function from statistics. In the Bayesian approach, the uncertainty in the parameters estimated when training a network is assumed to follow a particular distribution. We first start with a prior distribution *p(w)* which gives us an idea of the parameters before the data is used [9] but this only give us a vague idea as the distribution is quite broad. The prior distribution can be of any kind, for example Poisson or Geometric. In this case we will only use a Geometric distribution. We then wish to narrow this distribution down by finding the posterior probability density of the parameters *w* given a particular dataset *D*, *p(w|D)* where [9]:

$$p(w \mid D) = \frac{p(D \mid w)p(w)}{p(D)} \tag{5}$$

and *p(D|w)* is the dataset likelihood and *p(D)* is the evidence and ensures that the posterior integrates to 1 and is calculated by an integral over the parameter space. Once the posterior:

$$p(D) = \int p(D \mid w')p(w')dw' \tag{6}$$

is calculated we can then make a prediction for a new input by first calculating the prediction distribution [9]:

$$p(y \mid x^*, D) = \int p(y \mid x^*, w)p(w \mid D)dw \tag{7}$$

where *y* is the predicted values and then the actual prediction is found by [9]:

$$E(y \mid x^*, D) = \int yp(y \mid x^*, w)p(w \mid D)dw \tag{8}.$$

Here *E(.)* is the expected value in statistical terms. As can be seen from equations (6) and (8), there is an integral involved and the dimensionality of the integral is given by the number of network parameters (weights) and this is not analytically possible and simple numerical algorithms break down [9]. Therefore approximations to the posterior distribution are made (the toolbox used to train Bayesian Neural Networks is the NETLAB toolbox used with MATLAB®) and in the toolbox there are various different methods that can be used to approximate this integral [9]. There is the Metropolis-Hastings algorithm, the Hybrid Monte Carlo (HMC) algorithm and Gibbs sampling [9].

The main reason for the use of Bayesian techniques is simply to reduce the uncertainty in the weights and thus try to reduce the problem of over-fitting (i.e. over-fitting occurs when a network predicts badly because it is trained too much to its training data and predicts badly with unseen inputs [9]) and to ensure confidence intervals. Bayesian techniques do reduce the problem of over-fitting as has been proved by Nabney [9]. In NN's there is a need to optimise the network and thus reduce the error function [9]. In Bayesian techniques this is done by obtaining a posterior distribution for the weights so that they can only be found within a particular distribution thus narrowing the search for the optimal weight values [9]. Bayes' theorem helps us to do this but there are large integrals and there are several ways of evaluating these integrals as stated before. The sampling method that was used to approximate the integral is the Metropolis-Hastings Algorithm. It is not clear how the Bayesian framework can be used for SVM's and this is discussed next with a description of the Metropolis-Hastings Algorithm following that.

B. Bayesian Techniques for Support Vector Machines

It may not be clear as to why Bayesian Inference would be needed with SVM's. SVM's do produce a unique solution and are thus not exposed to errors resulting from the same randomness as NN's where the first layer weights and biases are assigned random values. However, the fact remains that with support vector machines the best parameters are not always found when trying to fit a curve to the data and there are still errors inherent in the predictions they provide. The parameters found are Beta, Bias and the number of support vectors (NSV) and are found from the quadratic programming optimisation problem. In NN's the bias values and weights (collectively referred to as weights in Section III.A.) are what are sampled for so that a distribution of NN's can be found and so that error bars can be established. With SVM's the same can be done and what will be sampled for is the Beta values. What could also be sampled for in SVM's is the bias value. Many support vectors have been trained in a previous implementation [6] and in each of the findings the bias values were found to be zero. Therefore it was decided to keep the bias at this value and only sample for the beta values. The number of support vectors is found from these values and so will not be sampled for [12]. In conclusion what is sampled for in both processes are the variables that are found from the optimisation processes either for NN's or SVM's and so these variables will be collectively called optimisation variables from now on.

C. The Metropolis-Hastings Algorithm

As stated before, Monte Carlo methods can be used to approximate the integrals involved in Bayesian techniques rather than using a Gaussian approximation [9].

Since there is an uncertainty in the process, we need to find the predictive distribution, i.e. the distribution that represents the possible outcomes of the MLP or SVM due to the uncertainty in the optimisation variables [9]. This distribution is an integral but in Monte Carlo methods it is approximated to be [9]:

$$p(y \mid x, D) = \frac{1}{N}\sum_{n=1}^{N} p(y \mid x, w_n) \tag{9}$$

where *N* is the number of samples chosen by the trainer of the network and w_n is the sample of optimisation variable vectors. These samples of optimisation variables can be chosen through different methods. A Hybrid Monte Carlo (HMC) method can be used to sample for these weights. HMC makes use of gradient information in its sampling and so may be very computationally intensive if the Hessian matrix (the derivative matrix) is difficult to compute. In addition to this, the derivative of the SVM is very difficult to find analytically and in fact the loss function used by the toolbox (the ε-insensitive

loss function) cannot be differentiated [8]. In light of this what is needed is to use the Metropolis-Hastings algorithm as it doesn't require any derivative information in its sampling. One drawback of this approach is that many more samples are required to gain an accurate distribution when compared to the HMC algorithm.

The principle behind the Metropolis-Hastings algorithm method is to generate a new state $x^{(n+1)}$ (in this problem it is a vector of the optimisation variables) from the old state $x^{(n)}$ by first generating a candidate state from a particular statistical distribution and then deciding whether or not to accept the candidate state. The procedure follows the following steps [9]:

1) *Generate the Candidate State x*:* The candidate state is generated with proposal distribution $S(x^*,x^{(n)})$ and an example distribution is a Boltzmann distribution.
2) *Acceptance and Rejection: E* is defined as the error or cost function. In this case it is an error function and is the objective function of the optimisation. If $E(x^*) < E(x^{(n)})$ then accept x^*, else accept x^* with probability $e^{(E(x^{(n)})-E(x^*))\frac{S(x^{(n)},x^*)}{S(x^*,x^{(n)})}}$. The 'else' part is done by generating a random number from the distribution chosen and this is the number to compare with the probability given by the 'else' part. If the random number is less than the probability given then accept x^*.
3) If x^* is accepted then let $x^{(n+1)} = x^*$ otherwise $x^{(n+1)} = x^{(n)}$.

This process is repeated for a certain number of samples which is chosen by the user. These three steps, in essence, describe how the sampling is done so that the summation of equation (9) can be accomplished and so that the posterior distribution can be found and thus allowing the optimisation of the NN or SVM. The sets of optimisation variables are thus selected or rejected according to the three steps above and the numbers of samples that are retained are the number of sets of optimisation variables accepted. For each set of optimisation variables there is a corresponding NN or SVM output. The prediction of the NN or SVM is the average of the outputs.

The usefulness of the Bayesian approach is that the prediction comes with certain confidence levels (upper and lower bounds). In fact the prediction mathematically is the same as that of the standard MLP or SVM. If we plot the prediction and upper and lower bounds (where the upper bound is the prediction plus the standard deviation of the outputs and the lower bound is the prediction minus the standard deviation of the outputs of the NN or SVM) then we say that the prediction is known to be within a certainty of 68% (because in the normal distribution 1 standard deviation form the mean constitutes 68% of the possible outcomes [7]).

IV. Implementation and Results

The data used for the option pricing problem was obtained form the South African Futures Exchange (SAFEX) website [13]. SAFEX is the body in South Africa that deals with futures and options and guarantees the integrity of the contracts entered on a daily basis. Data was obtained for futures and options from January 2001 to December 2003 for options and futures on the All Share Index (ALSI) of South Africa. What was attempted was to price the call options on the index. There were high and low prices available on the options contracts and what was decided was to use the average of these two and this would be the variable try and predict.

There are various factors that influence options prices. In the Black-Scholes equations, there are five variables that are used in pricing the options. Even though these equations only hold for European options, it was still felt that the same factors would influence the American version of the options prices. The five factors are strike price (price that the owner can exercise the contract), the time to maturity (the date at which the contract expires), the risk-free rate of interest (the return that investors can obtain risk free in the market), the spot price (the price of the underlying asset today) and the volatility of the underlying asset. The spot price was not used as one of the inputs for the prediction as it was felt that the strike price is close to this value and so would not help in the training of the MLP's or SVM's. The risk free rate was also not used as an input as it was found that this would be fairly constant in the period of the data and so wouldn't help with the predictions. The other three factors were used as inputs in the training of the MLP's and SVM's. In both implementations 300 test points were used to evaluate the errors in the predictions and the bounds.

A. Bayesian Neural Networks

The NETLAB toolbox was used with MATLAB®. There were several examples that came with the toolbox but mainly for use of the HMC algorithm in the sampling for the Bayesian Neural Networks. One of these examples was edited and modified for the option pricing problem and the sampling method used was the Metropolis-Hastings algorithm described in Section III.C.

What was attempted with the NN's was to obtain the bounds from the data as this is the most useful application of Bayesian techniques. This would then give us the price predicted and within certain confidence levels. Factors in the training algorithms that would influence this were the number of samples omitted at the start of the chain and accepted and the step size used by the algorithm. It was found that the best step size to use when sampling was 0.0005. If a step size less than this was used then there were too many samples being accepted (the bounds were too small showing that they were not an accurate reflection of what they should be) and if the step size was made larger then it was found that there weren't enough samples being accepted and so too wide a search space was being explored. The number of hidden units in the MLP was also experimented with but this was just so that we could get the average prediction error to be as low as possible. The results of the algorithm can be seen in Table 3.

Table 3: Bayesian Neural Networks Results

Omit	Max	Mean	Samples	Time	Dev	Hid
500	10234	95	1000	25	442	10
500	12229	122	1000	33	327	20
500	3855	64	1000	44	281	40
1000	7933	79	1000	31	289	10
1000	7921	85	1000	42	265	20
1000	4730	90	1000	57	271	40
500	14960	110	2000	64	363	10
500	12250	109	2000	74	385	20
500	9896	102	2000	93	441	40
1000	9615	89	2000	65	233	10
1000	7334	88	2000	79	260	20
1000	8962	119	2000	104	339	40
2500	6076	99	5000	276	211	10
2500	7759	102	5000	310	163	20
2500	8805	95	5000	369	221	40

Omit = number of samples omitted at the start of the chain, Max = the maximum error found in %, Mean = the average error found in %, Samples = the number of samples not omitted in the chain, Time = the time taken for training in seconds, Dev = the average deviation and thus the magnitude of the bounds above and below the prediction, Hid = the number of hidden units used in the MLP.

The algorithm makes use of an omitted amount of samples at the start of the chain because the initial values for the sampling are set to random values. Therefore the initial samples will in fact be way out of where the optimal weight values should be. The predictions are found from the average values of all the predictions given by the samples and if the first samples produce big error values then it is desirable to omit these initial samples until more optimal weight values are sampled for. There are several conclusions that can be drawn from Table 3. Firstly, intuitively it was thought that the more samples there were then the better the predictions would be. This was not the case and the average error was lower with 1000 samples than with 2000 or 5000 samples. As the number of samples grew it was also found that the bounds were smaller which was desired but the average error was too high. Smaller bounds are desired because it would mean that the price is known to be with a smaller range with a particular certainty [7] (68% certainty if one standard deviation is used). In conclusion we can say that 1000 samples sufficed for this algorithm with 500 omitted samples at the start of the chain.

B. Bayesian Support Vector Machines

The same data that was used to test the Bayesian NN implementation was also used to test the Bayesian SVM implementation and the exact same test points were used to evaluate the implementation. There was a SVM toolbox obtained and it had to be used in conjunction with the NETLAB toolbox (because this toolbox provided the Metropolis-Hastings algorithm) and so some coding had to be done to incorporate the two toolboxes in MATLAB®.

With Bayesian MLP's, random values were initially set to the weights and then the Metropolis-Hastings algorithm sampled with this as the starting point. This was attempted with SVM's as well but it was found that with as much as 5000 samples the average error was about 100% which is much too high. It was therefore felt that many more samples (say 30000) were needed but this would take the algorithm approximately 7.5 hours to finish. It was rather decided to provide a starting point to the Metropolis-Hastings algorithm rather than randomly generating one. What was done was thus to actually train an SVM and these values were used as the starting value for the sampling. With SVM's there are optimal values for capacity and ε-insensitivity when training the SVM. The best values were found before [6] and were set at 10 for capacity and 0.005 for ε-insensitivity and these were the values used to train the SVM so that the starting values for the betas could be found and used in the Metropolis-Hastings algorithm.

In light of this, when too high a step value was used then no samples were accepted and so there was no value for the bounds which thus doesn't help in the analysis. Also if too few samples were used with too high a step value then the average error was raised and so the number of samples and step value were experimented with. The number of omitted samples was always set at zero because the initial sample value given to the Metropolis-Hastings algorithm produced good predictions and so samples around this point would be useful and it was thus not desired to omit these. It was felt that as many new samples that were retained produced more realistic values in terms of finding the values for the bounds. The results of the implementation can be seen in Table 4.

It must be stated, firstly, that we would like the deviation to be low but if the deviation was too low then it can be argued that not enough of the search space has been sampled for the deviation to be indicative.

From Table 4 it can be seen that at 10000 samples the prediction did improve if the step was larger. It is believed that if more samples were used, that the eventual 34% [6] error mark would be reached and with the step value at this large value, that sufficient search space would be explored to gain an indicative value for the bounds. The value for the bounds at this point was still lower than that found by the Bayesian MLP and the average error found was also much lower than that found by the Bayesian MLP.

Table 4: Results of Bayesian Support Vector Machines

Max	Mean	Samples	Time	Dev	Step
3607	62	1000	849	348	0.01
3177	53	1000	863	167	0.001
486	36	1000	861	77	0.0001
865	35	1000	853	41	0.00001
959	35	1000	854	32	0.000001
1091	53	10000	8729	238	0.01
5425	71	10000	8811	131	0.001
649	35	10000	8889	88	0.0001
955	35	10000	8811	37	0.00001
1036	37	10000	8811	24	0.000001

Max = the maximum error found in %, Mean = the average error found in %, Samples = the number of samples not omitted in the chain, Time = the time taken for training in seconds, Dev = the average deviation and thus the magnitude of the bounds above and below the prediction, Step = the step size used by the Metropolis-Hastings algorithm to search for new samples.

V. Comparison of Bayesian Techniques and Standard Computational Intelligence Methods

A comparison has been made between MLP's and SVM's without Bayesian techniques and it was found that SVM's out-performed MLP's [6]. The MLP's achieved, at best, an average error of 52% and the SVM' achieved an average error of 34%. The MLP error levels were not achieved here and at best the error was 64% but these levels would be achieved if many more samples were used. With the SVM's, a 35% error level was achieved which is very similar to that found before

[6]. However at this level, the error bound values were very small and were thus not very indicative of what they stand for. Again if more than 10000 samples were used then the 34% error mark would be reached again and this would produce bounds more reflective as is given by the 6th entry in Table 4. In conclusion, SVM's out-performed MLP's both in the Bayesian framework and with standard training methods, in terms of error performance.

VI. Conclusion

Bayesian MLP's and Bayesian SVM's were implemented and tested in MATLAB® using the necessary toolboxes and coding needed. It was found that Bayesian MLP's with the Metropolis-Hastings algorithm outperformed other Bayesian techniques such as the Hybrid Monte Carlo and Gaussian approximation sampling functions [14]. These algorithms produced bounds that were quite larger than that found by the Metropolis-Hastings algorithm with MLP's. The HMC algorithm took much longer to run and was found to produce bounds larger and the Gaussian approximation algorithm produced much larger bounds although it took much less time to train. Error values were also better with the Metropolis-Hastings algorithm but it must be said that all Bayesian NN's will tend to produce the same prediction as the normal NN if enough samples are used. The Bayesian framework obtains its usefulness by the bounds that it can produce and thus the confidence levels it provides with the predictions.

In terms of Bayesian SVM's, SVM's have been found to provide more accurate predictions [6] than NN's in the option pricing problem and it produced bounds, in the Bayesian framework, that were smaller than that provided by their NN counterparts. This however comes at an expense as it takes much longer for the Bayesian framework to be implemented with SVM's than NN's. It could take as long as 2.5 hours compared to 44 seconds for Bayesian NN's. Although this time is much longer it must be said that Bayesian SVM's still outperformed Bayesian MLP's as the error in the predictions and the bounds were much smaller.

With implementing Bayesian SVM's, it seems that there is more scope for work that can still be done to further improve the bounds found. Instead of training an SVM first and using these beta values as the starting point for the Metropolis-Hastings algorithm, what can rather be done is to only sample for some of the beta values as it was found that some of the beta values were better kept at the value of the capacity (as beta values range between negative and positive capacity) and then some of the other beta values could then be sampled for. This may make the sampling faster and also this would allow more search space to be explored in the pursuit of trying to find more samples.

VII. Acknowledgement

I would like to thank the National Research Fund (NRF) for their financial contribution in making this research possible.

References

[1] S. Ross, R. W. Westerfield, B. D. Jordan and C. Firer, *Fundamentals of Corporate Finance 2nd South African Edition*, McGraw-Hill Book Company, Sydney, Australia: 2001.

[2] J. C. Hull, *Options, Futures and Other Derivatives*, Prentice Hall, Upper Saddle River, New Jersey, U.S.A.: 2003.

[3] M. M. Pires, Masters Thesis, School of Electrical and Information Engineering, University of the Witwatersrand, 2005, Appendix A: Hedging with Options and Other Contracts.

[4] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, vol. 81, 1973, pp. 637-659.

[5] R. A. Jarrow and S. M. Turnbull, *Derivative Securities 2nd Edition*, South-Western College Publishing, U.S.A.: 2000.

[6] M. M. Pires and T. Marwala, "American Option Pricing Using Multi-Layer Perceptron and Support Vector Machine", in *Proceedings of the IEEE Conference in Systems, Man and Cybernetics,* The Hague, October 10-13 2004, pp. 1279-1285.

[7] T. H. Mirer, *Economic Statistics and Econometrics 3rd Edition*, Prentice Hall, U.S.A.: 1995, pp. 209-218.

[8] W. Chu, S. S. Keerthi and C. J. Ong, "Bayesian Support Vector Regression Using a Unified Loss Function", *IEEE Transactions on Neural Networks*, vol. 15, no. 1, 2004, pp. 29-44.

[9] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition*, Springer-Verlag, London, Great Britain: 2003.

[10] J. Joachims, "Making large-scale SVM learning practical" in B. Scholkopf, C. J. C. Burges and A. J. Smola, editors, *Advances in Kernel Methods-Support Vector Learning*, MIT Press, Cambridge, MA: 1999, pp. 169-184.

[11] T. B. Trafalis and H. Ince, "Support Vector Machine for Regression and Applications to Financial Forecasting", *IEEE-INNS-ENNS International Joint Conference on Neural Networks*, Como, Italy, July 24-27, 2000.

[12] S. R. Gunn, *Support Vector Machines for Classification and Regression,* Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, University of South Hampton, United Kingdom, 1998.

[13] *South African Futures Exchange*, Available: http://www.safex.co.za.

[14] M. M. Pires, Masters Thesis, School of Electrical and Information Engineering, University of the Witwatersrand, 2005, Appendix B: Paper for *Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa.*

Chapter 4: Computational Intelligence Methods for American Option Pricing

M. M. PIRES
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg, South Africa
m.pires@ee.wits.ac.za

Abstract – **An option is used, by many firms, to hedge their financial risk. Black and Scholes introduced an option pricing model but there were several underlying assumptions. The most important of which is that the model only holds for European options. In South Africa, options are all American type and so various option brokers use different methods to price the options. What is used here is computational intelligence predictive tools to price the options by inputting past data. There are various predictive tools used and it was found that support vector machines out-performed the neural network techniques in terms of error but not computational time when the maximum-likelihood framework is used. The Bayesian frameworks is also applied to both predictive tools and this introduced a new paradigm in option pricing as the Bayesian frameworks has not been applied in the option pricing field before. It was found that Bayesian Support Vector Machines out-performed Bayesian Multi-Layer Perceptrons in terms of error analysis but Bayesian Support Vector Machines took much longer to train.**

I. INTRODUCTION

Options are used extensively in South Africa by many large firms to hedge their financial risk and thus keep their profit margins at certain levels. An option gives the owner of the option the right, not the obligation, to buy (a call option) or sell (a put option) an underlying asset at a later date (known as the maturity date) but by deciding on the price of the asset now (known as the strike price). Owners of options don't have to exercise their options if a more favourable situation occurs in the market and this makes them more valuable than futures meaning that a premium has to be paid to have these options [1].

There have been many attempts in the past to price options and Black and Scholes [2] came up with what is known today as the Black-Scholes option pricing model. In this model there were several underlying assumptions and most importantly the pricing model only held for European type options. A European type option only allows the owner of the option to exercise the option on the expiry date of the contract. There are also American type options and these contracts can be exercised on any date from when they are taken up until the maturity date. This extra flexibility makes American type options either the same value or more valuable than European type options [3]. American options also have a second random process inherent in their model (as it is not known when it will be optimal to exercise the option) and thus makes them more difficult to price.

In South Africa, only American type options are used. There is no standard for pricing these options on the South African Futures Exchange (SAFEX). SAFEX is the body in South Africa that provides the market for the trading of derivatives and ensures the integrity of all derivative contracts. The various option brokers in the South African market use different techniques to price these options and what this means is that different prices can be obtained from different options brokers. As a result the actual option premium are susceptible to market forces. With this in mind, it seems intuitive to believe that a predictive tool can be used to price American type options.

Predictive tools are used in Computational Intelligence methods since the theory of Computational Intelligence was introduced. The first method of interest is Neural Networks (NN's) and these have been used in many fields such as pattern recognition (Ripley [4]), biomedical applications (Hudson and Cohen [5]) and even in predicting inter-state conflict (see Marwala and Lagazio [6]). Predictive tools can be used for classification and regression applications. Classification problems deals with the placing of classes of objects in specific categories and classifiers predict which class certain objects will fall under. Regression problems deal with finding an actual number output based on the inputs that the predictive tool is given. Predictive tools base their predictions on what has happened in the past and so typically they require a training dataset to train them appropriately and then produce predictions based on the test dataset. Here we attempt to use Support Vector Machines (SVM's) as well as Neural Networks. What is also presented here is the introduction of the Bayesian framework to both NN's and SVM's. What the Bayesian framework does is allow the predictions to be provided with certain confidence levels (more reasons for the use of the framework are outlined in Section V.).

II. PREVIOUS ATTEMPTS AT OPTION PRICING WITH PREDICTIVE TOOLS

NN's have been used before to price options. Hutchinson et. al. [7] were one of the first people to attempt to price options using NN's but this was only attempted for European options. It was also attempted by Morelli et. al. [8] for American and European options but with little success with the American option pricing portion of the research. In fact their findings were only published for European options. Kelly [9] attempted American option pricing with NN's but found that the error in the predictions was proportional to the actual option price and error values as high as 60% were found at higher option prices.

SVM's have not been used before in the option pricing field and so what is illustrated here is their performance against MLP's (which have been used before) as well as their performance in the Bayesian framework. SVM's have produced satisfactory results in other fields. Trafalis and Ince [10] applied them to financial forecasting and Chan et. al. [11] applied them to Glaucoma diagnosis and found that they outperformed the NN architectures used.

Bayesian techniques have not been used in the option pricing field either with NN's or SVM's and they have only been recently applied to SVM's by Chu et. al [12].

III. The Black-Scholes Model

With all predictive tools there is a need to distinguish between what will be used as inputs and what will be used as outputs. The inputs are chosen as those factors that would affect the outputs in the real world. In the option pricing problem we are attempting to price options and so the option price will be the only output of the network. We need to then decide to what to use as inputs and thus decide what affects option prices.

The Black-Scholes model is used together with some form of Markov chain sampling by many option brokers to price American type options. We can thus deduce that the same factors that affect European options may also affect American option prices. It must be noted that there are other factors that may affect American option prices but for these purposes we will only choose those found by the Black-Scholes model.

The Black-Scholes equations for put and call options can be found from Hull [13] and are stated as follows:

$$c = S_0 N(d_1) - Ke^{-rT} N(d_2) \quad (1)$$

and

$$p = Ke^{-rT} N(-d_2) - S_0 N(-d_1) \quad (2)$$

where

$$d_1 = \frac{\ln(S_0 / K) + (r + \sigma^2 / 2)T}{\sigma\sqrt{T}} \quad (3)$$

and

$$d_2 = \frac{\ln(S_0 / K) + (r - \sigma^2 / 2)T}{\sigma\sqrt{T}} \quad (4)$$

$N()$ is the cumulative probability distribution function, S_0 is the spot price (current price today) of the underlying asset, K is the strike price agreed upon, r is the risk-free rate of interest (this interest rate is usually that denoted by government bonds in a politically stable country, see Ross et. al [1]), T is the time to maturity in number of years and σ is the volatility of the underlying asset (volatility implies how much the underlying asset has deviated in the past, see Ross et. al. [1]).

From Equations (1) to (4) we can see that there are five factors that influence options prices. They are: spot price, strike price, the risk-free rate of interest, the time to maturity of the contract (the time difference between when the contract is taken and when the contract expires) and the volatility of the underlying asset. All these can be used as inputs to the predictive tools but it was chosen to eliminate two of these five factors. Firstly, the risk-free rate of interest was excluded. The risk-free rate of interest was excluded because it doesn't seem to vary too much with time in South Africa. In fact from January 2004 to December 2005 (projections were used for the period that hasn't occurred yet by the Standard Bank Group of South Africa) the interest rate was only found to fluctuate between 12% and 11%. A plot of interest rate can be seen in Figure 5 below:

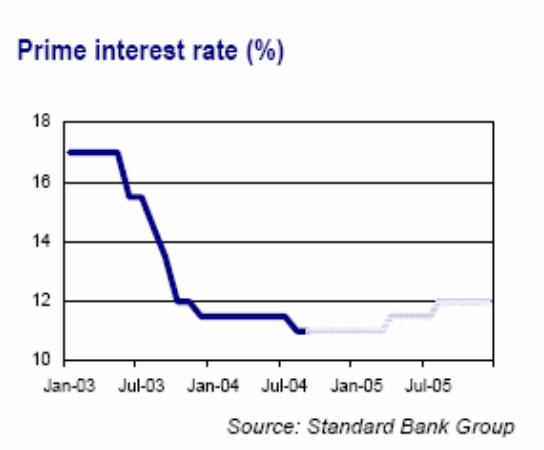

Figure 5: Interest Rates in South Africa

Therefore since the interest rates were fairly constant, including it would be equivalent to not having included it as an input, because it will not play a role in determining the predictions. Secondly, the spot price was not included. The quoted spot price (from the data provided) was always very close to the strike price provided and so the effect would have been the same as having two inputs that are similar. Therefore the spot price was also not included.

We can now discuss the mathematics behind the techniques used and show how predictions are obtained from raw data in each of the predictive techniques.

IV. Computational Intelligence Methods

A. The Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is the most widely used architecture for neural networks (Nabney [14]). In this paper two layers with full connectivity between hidden units and inputs, and between hidden units and outputs were used. Hornick et. al. [15] have shown that two-layer networks, with sigmoidal type activation functions in the hidden layer and linear activation functions in the output layer, are universal approximators, provided that there are a sufficient number of hidden layer neurons (Polycarpou [16]). This means that a simple two layer network is sufficient and that more hidden layers are not required. For the particular application, a network with three inputs and one output was constructed as can be viewed in Figure 6 and from now on we will only consider a network with three inputs and one output.

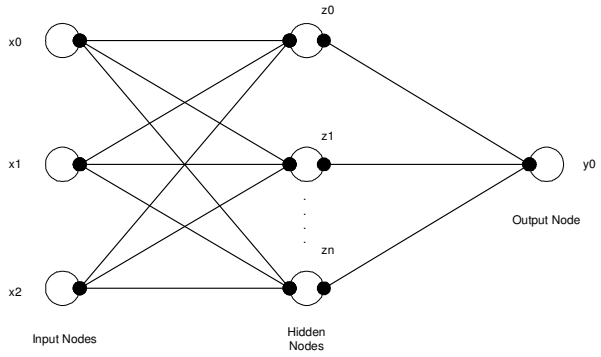

Figure 6: Two-Layer Architecture of the Multi-Layer Perceptron

As can be seen in Figure 6, the number of hidden nodes is unknown and a few different amounts of hidden nodes were tried. For the first layer of the network we have activation parameters and bias variables ($a_j^{(1)}$ and $b_j^{(1)}$ respectively) associated with each hidden node. Each line (joining to different nodes) in Figure 6 has a weight associated with it ($w_{ji}^{(1)}$) and so we have:

$$a_j^{(1)} = \sum_{i=1}^{3} w_{ji}^{(1)} x_i + b_j^{(1)} \tag{5}$$

where j is the number of hidden units (n from Figure 6) and i is the number of input units. Data is fed into the network in terms of training inputs and training outputs. The activation parameters are then transformed by the non-linear activation functions of the hidden layer. There are many activation functions that can be used and in this study the hyperbolic tangent function is used for the hidden layer. The outputs of the hidden units (z_j) can thus be written as follows and thus making the activation function sigmoidal and in keeping with Hornick et. al. [15]:

$$z_j = \tanh(a_j^{(1)}) \tag{6}$$

The z_j are then transformed by the second layer of weights ($w_{kj}^{(2)}$) and biases ($b_k^{(2)}$) to give the second layer activation functions ($a_k^{(2)}$)

$$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)} \tag{7}$$

where k is the number of output nodes (in this case k = 1). The second layer activation functions are then transformed through a linear function into the output in keeping with Hornick et. al. [15]:

$$y_k = a_k^{(2)} \tag{8}$$

The weights at each layer are chosen using some optimisation method. The error function between the output predicted by the neural network and the target vector is constructed as follows:

$$E = \sum_{k=1}^{1} \sum_{g=1}^{N} (y_{kg} - y_{k,t\arg et})^2 \tag{9}$$

where E is the error to be minimized and N is the number of sets of training data given. The sets of outputs y_{kg} are the outputs predicted by the network given the training inputs and $y_{k,target}$ are the training outputs. The weights at each layer are changed until the error shown in equation (9) is as low as possible. The weights in each layer are initially set to random values. Several optimisation techniques such as the conjugate gradient method, the scaled conjugate gradient method, the Quasi-Newton method and the gradient descent method may be used to minimize equation (9) (see Nabney [14]). Each of these methods was tried and the results of each network with different optimisation techniques were analysed.

B. Support Vector Machines

The basic idea behind support vector regression is to map the input space to an output space. Suppose we have the training data set with one input and one output being considered: $\{(x_1,y_1),\ldots,(x_l,y_l)\} \subset \chi \times \Re$, where χ is the space of the input parameters and $\Re$ denotes the real number set. We wish to find a function $f(x)$ that will map the training inputs to the training outputs. In Support Vector (SV) regression we wish to find this function that has at most ε deviation from the actual training targets y_l. We can fit several kinds of functions $f(x)$ to map training inputs to training outputs. These functions are known as kernel functions but these cannot just be any functions, kernel functions have to adhere to some criteria as was stated by Joachims [17]. For the purposes of explanation we will consider a linear kernel function

$$f(x) = \langle w, x \rangle + b \quad with \quad w \in \chi \quad , \quad b \in \Re \tag{10}$$

where $\langle .,. \rangle$ denotes the dot product.

We seek to find small values for w (w is known as the Beta parameter and b is the bias parameter from equation (10)) and one way to do this is to minimize the Euclidean norm $\|w\|^2$ as shown by Joachims [17]. We then include slack variables ξ_i, ξ_i^* so that certain infeasible constraints in the minimization of the Euclidean norm can be used and the minimization problem then becomes

$$minimize \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$
$$subject \quad to \quad \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{11}$$

where l is the number of training points used. The constraints above deal with an ε-insensitive loss function used to penalize certain training points that are outside of the bound given by ε which is a value chosen by the user. There are various other loss functions such as the Huber loss function which can also be used but the most common is the ε-insensitive loss function as stated by Gunn [18].

The loss function is given by

$$|\xi|_\varepsilon = \begin{cases} 0 & if \quad |\xi| \le \varepsilon \\ |\xi| - \varepsilon & othererwise \end{cases} \tag{12}$$

The value for C in equation (11) is used as the amount to which deviations from ε is tolerated (see Trafalis and Ince [10]). It can be seen as a measure of over-fitting a function too well to its training points. If the value of C is set too high then the function found ($f(x)$) will be too well fitted to the training data and will not predict very well on data that isn't seen by the training of the function. It means that points lying outside of the bounds given by ε are not penalized enough and this results in the function being too well fitted to the training points (see Burges [19]). A sketch of a linear function being fitted to training data can be seen in Figure 7 with the bounds being shown.

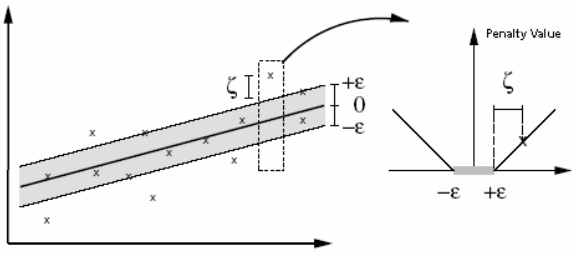

Figure 7: Linear SVM Regression for a Set of Data (left) and the ε-insensitive Loss Function [17].

The function on the right in Figure 7 is used to penalize those points that lie outside of the bounds shown on the left. The more a point lies outside of one of the bounds (either below or above), the more the point is penalized and thus plays less of a role in the determination of the function. Those points that fall within the bounds of the function are not penalized at all and their corresponding slack variable values (ξ_i, ξ_i^*) are given zero and thus these points will play a major contribution in the determination of the function $f(x)$.

The optimisation problem of equation (11) is then set up to be a quadratic programming problem by first finding the Lagrangian multiplier and applying the Karush-Kuhn Tucker (KKT) conditions (see Joachims [17]). Then the values for w and b can be found so that the linear function to fit the training data of equation (10) can be explicitly found. Note that this example using the constrained optimisation problem of equation (11) is for a linear kernel function and the constrained optimisation problem of equation (11) is different for different kernel functions.

V. Bayesian Techniques

We will first discuss Bayesian techniques applied to Neural Networks as this is what the purpose of the techniques were originally intended for. Section V.B then discusses how the Bayesian framework can be applied to SVM's.

A. Bayesian Techniques for Multi-Layer Perceptrons

Bayesian techniques are derived from Bayes' theorem from probabilistic mathematics. With NN's there is always an error in the predictions made and we thus have (see Nabney [14]):

$$y = f(x; w) + \varepsilon \tag{13}$$

where y is the actual output desired, f is the output predicted by the network, ε is the error, w are the weights and x is a vector of inputs. Even if we are given ε and the same network is run twice with the same parameters, we will obtain different weights w both times and thus there is an uncertainty in the training of the networks and this can be attributed to the randomness in the assignment of weights. Generally some complex models try to fit the noise into the predictions which cause problems when trying to predict with unseen inputs (the problem of over training) and thus cause there to be even more error in the predictions.

The parameter $p(.)$ wherever used from now on denotes the probability function. In the Bayesian approach, the uncertainty in the parameters estimated when training a network is assumed to follow a particular distribution. We first start with a prior distribution $p(w)$ which gives us an idea of the parameters before the data is used but this only give us a vague idea as the distribution is quite broad. The prior distribution can be of any kind, for example Poisson or Geometric. In this case we will only use a Geometric distribution. We then wish to narrow this distribution down by finding the posterior probability density of the parameters w given a particular dataset D, $p(w|D)$ where:

$$p(w \mid D) = \frac{p(D \mid w)\, p(w)}{p(D)} \tag{14}$$

and $p(D|w)$ is the dataset likelihood and $p(D)$ is the evidence and ensures that the posterior integrates to 1 and is calculated by an integral over the parameter space. Once the posterior:

$$p(D) = \int p(D \mid w')\, p(w')\, dw' \tag{15}$$

is calculated we can then make a prediction for a new input by first calculating the prediction distribution:

$$p(y \mid x^*, D) = \int p(y \mid x^*, w)\, p(w \mid D)\, dw \tag{16}$$

where y is the predicted values and then the actual prediction is found by:

$$E(y \mid x^*, D) = \int y p(y \mid x^*, w)\, p(w \mid D)\, dw \tag{17}$$

Here $E(.)$ is the expected value in statistical terms. As can be seen from equations (15) and (17), there is an integral involved and the dimensionality of the integral is given by the number of network parameters (weights) and this is not analytically possible and simple numerical algorithms break down. Therefore approximations to the posterior distribution are made. There are various different methods that can be used to approximate this integral such as the Metropolis-

Hastings algorithm, the Hybrid Monte Carlo (HMC) algorithm (together known as Monte Carlo methods) and Gibbs sampling.

The main reason for the use of Bayesian techniques is simply to reduce the uncertainty in the weights and thus try to reduce the problem of over-fitting (i.e. over-fitting occurs when a network predicts badly because it is trained too much to its training data and predicts badly with unseen inputs) and to ensure confidence intervals. Bayesian techniques do reduce the problem of over-fitting as has been proved by Nabney [14]. In NN's there is a need to optimise the network and thus reduce the error function shown in equation (9). In Bayesian techniques this is done by obtaining a posterior distribution for the weights so that they can only be found within a particular distribution thus narrowing the search for the optimal weight values. Bayes' theorem helps us to do this but there are large integrals and there are several ways of evaluating these integrals as stated before. The sampling method that was used to approximate the integral is the Metropolis-Hastings Algorithm. It is not clear how the Bayesian framework can be used for SVM's and this is discussed next with a description of the Metropolis-Hastings Algorithm following that.

B. Bayesian Techniques for Support Vector Machines

It may not be clear as to why Bayesian Inference would be needed with SVM's. SVM's do produce a unique solution and are thus not exposed to errors resulting from the same randomness as NN's where the first layer weights and biases are assigned random values. However the fact remains that with support vector machines the best parameters are not always found when trying to fit a curve to the data and there are still errors inherent in the predictions they provide. The parameters found are Beta, Bias and the number of support vectors (NSV) and are found from the quadratic programming optimisation problem. In NN's the bias values and weights (collectively referred to as weights in Section V.A.) are sampled for so that a distribution of NN's can be found and so that error bars can be established (see Section V.D). The number of support vectors is found from these values and so will not be sampled for. In conclusion what is sampled for in both processes are the variables that are found from the optimisation processes either for NN's or SVM's and so these variables will be collectively called optimisation variables from now on.

C. The Metropolis-Hastings Algorithm

As stated before, Monte Carlo methods can be used to approximate the integrals involved in Bayesian techniques.

Since there is an uncertainty in the process, we need to find the predictive distribution, i.e. the distribution that represents the possible outcomes of the MLP or SVM due to the uncertainty in the optimisation variables. This distribution is an integral but in Monte Carlo methods it is approximated to be:

$$p(y \mid x, D) = \frac{1}{N}\sum_{n=1}^{N} p(y \mid x, w_n) \qquad (18)$$

where N is the number of samples chosen by the trainer of the MLP or SVM and w_n is the sample of optimisation variable vectors. These samples of optimisation variables can be chosen through different methods. A Hybrid Monte Carlo (HMC) method can be used to sample for these weights. HMC makes use of gradient information (the gradient of the error function in equation (9)) in its sampling and so may be very computationally intensive if the Hessian matrix (the derivative matrix) is difficult to compute. In addition to this, the derivative of the SVM is very difficult to find analytically and in fact the loss function used by the toolbox (the ε-insensitive loss function) cannot be differentiated (see Chu et. al. [12]). In light of this what is needed is to use the Metropolis-Hastings algorithm as it doesn't require any derivative information in its sampling. One drawback of this approach is that many more samples are required to gain an accurate distribution when compared to the HMC algorithm.

The principle behind the Metropolis-Hastings algorithm method is to generate a new state $x^{(n+1)}$ (in this problem it is a vector of the optimisation variables) from the old state $x^{(n)}$ by first generating a candidate state from a particular statistical distribution and then deciding whether or not to accept the candidate state. The procedure follows the following steps:

1. *Generate the Candidate State x*:* The candidate state is generated with proposal distribution $S(x^*,x^{(n)})$ and an example distribution is a Boltzmann distribution.
2. *Acceptance and Rejection: E* is defined as the error or cost function (equation (9)). In this case it is an error function and is the objective function of the optimisation. If $E(x^*) < E(x^{(n)})$ then accept x^*, else accept x^* with probability $e^{(E(x^{(n)})-E(x^*))\frac{S(x^{(n)},x^*)}{S(x^*,x^{(n)})}}$. The 'else' part is done by generating a random number from the distribution chosen and this is the number to compare with the probability given by the 'else' part. If the random number is less than the probability given then accept x^*.
3. If x^* is accepted then let $x^{(n+1)} = x^*$ otherwise $x^{(n+1)} = x^{(n)}$.

This process is repeated for a certain number of samples which is chosen by the user. These three steps, in essence, describe how the sampling is done so that the summation of equation (18) can be accomplished and so that the posterior distribution can be found and thus allowing the optimisation of the MLP or SVM. The sets of optimisation variables are thus selected or rejected according to the three steps above and the numbers of samples that are retained are the number of sets of optimisation variables accepted. For each set of optimisation variables there is a corresponding MLP or SVM output. The prediction of the MLP or SVM is the average of all the predictions.

There are still several questions that may be unexplained thus far but one main question is why the Bayesian framework is useful. It does prevent over-fitting of the MLP's and SVM's but there is another main reason and that is the confidence levels that it provides.

D. The Confidence Levels Provided by Bayesian Inference

The sampling done by the Metropolis-Hastings Algorithm provides the predictions from the MLP's or SVM's with certain confidence levels. Each sample for the optimisation variables will produce a different MLP or SVM. For each sample generated, for the MLP sampling, there is a corresponding MLP that will provide a prediction and the same is true when sampling Metropolis-Hastings sampling is applied to SVM's. The predictions for both MLP's and SVM's will follow a normal distribution of enough samples are generated.

Let us first consider the Normal Distribution. A normal distribution is a bell shaped distribution that can be totally defined by its average and standard deviation (see Mirer [20]). The average value for the distribution is in the centre of the bell shape where the Peak occurs.

Let us just consider MLP's for now just for clarification. If we choose to have, say, 5000 samples in our Metropolis-Hastings algorithm for MLP's then there will be 5000 different predictions for each test input. If the predictions are to follow a normal distribution then we can say that the actual prediction is the average of all the predictions. We can then also find the standard deviation of the predictions. For a normal distribution, one standard deviation from the mean constitutes 68% of all possibilities for the predictions. In conclusion, what can be found is a prediction and the bounds that show a confidence level on the prediction. If one standard deviation is plotted above and below the average, then it is said that in this region we are 68% certain that the actual value lies between this band and therefore giving us more confidence in the predictions.

The same procedure is done for SVM's and we can see that if the bounds are smaller then the more confident we will be in our predictions as we will know the actual value to be within a smaller range. This is the added advantage of using Bayesian techniques for MLP's and SVM's. It must be stated, however, that the algorithm requires enough samples to be used so that the predictions given by the samples can actually follow a normal distribution.

VI. Implementation and Results

Data had to be obtained to test the procedures described by Sections VI and V. SAFEX [21] has a website with all its historical data. Data was obtained for all options for the period of January 2001 to December 2003. This resulted in there being too much data and it was decided to use a particular option. If MLP's or SVM's were trained using too much data it would result in the techniques having to learn too much information and could result in poor performance at the testing stage due to over-fitting. It was decided to use call options on the All Share Index (ALSI). This resulted still in much data but at least the data wasn't very different with strike prices as it would be if many different options were used. In conclusion it resulted in the MLP's and SVM's having to learn less information and make them less susceptible to forgetting. Each technique requires training and test data. The inputs used have been discussed in Section III. The output used will be the call price of the option. Since option prices are susceptible to market forces, there are high and low prices and the average of these was used as the actual call price.

Training data is used to minimize equation (9) and so that the algorithms learned from this data. The test data (consisting of inputs and outputs) is fed into the trained algorithm (only inputs are fed in) and then predictions are obtained. These output predictions are then measured against the actual values so that certain error analyses can be done.

For both MLP's and SVM's and for the Bayesian techniques applied to both, the data fed into the techniques has to be normalized. This is because some of the inputs will have higher values than others. For example, asset volatility was usually in the 20 to 30 range and some of the strike prices used was in the R8000 range. To MLP's and SVM's, this would mean that the training algorithms would see certain inputs as more important than others and so it was desirable to get all the inputs within a certain range.

There are several normalization techniques that can be used and many were tried. The one that resulted in the most accurate predictions is given by:

$$data = \frac{data - \mu}{\sigma} \tag{19}$$

where *data* is the data set to be normalized, μ is the mean of the data set and σ is the standard deviation of the data set. This is done for both test and training data because normalised data must be fed into the trained algorithms if predictions are to be obtained. Each input data set will have a mean and standard deviation associated with it and each output will have a mean and standard deviation associated with it.

This normalisation technique was done for all the training algorithms including Bayesian techniques applied to MLP' and SVM's. It was then just a matter of testing the actual algorithms and choosing the best parameters for each algorithm. For example it is not known how many hidden units to use for standard MLP's and so different amounts had to be used for this training algorithm. It must be stated the same test data (test data is known as unseen data as the techniques are not exposed to this data in the training and this is desired when trying to gain predictions and there were always 300 test points) was used for both techniques but different training data was used for each technique as some techniques required more training data to provide better predictions than others.

Both MLP's and SVM's were trained using a machine with a 1.8GHz processor and 256MB of RAM with no other programs running but the Bayesian techniques were run using a machine with a 2.8GHz processor and 512MB of RAM with no other programs running. This is included as times are given for the algorithms used The reason for stating the processing power used is because times for the algorithms trained are stated and so these times can be put into context.

A. The Multi-Layer Perceptron

To implement MLP's a toolbox was used, called the NETLAB toolbox which provided several functions and examples that could be edited so that the algorithms could be applied to particular problems [14]. This toolbox was constructed for

use with MATLAB® which is a mathematically intensive computer software package.

With MLP's there are several factors that can be experimented with so that the algorithm will perform at its best and provide predictions with the lowest average error. Firstly, the number of training cycles can be experimented with. If the algorithm is allowed to train for too many cycles then equation (9) will be at too low a value. This would mean that the MLP would be too well trained to its training data and would not predict well on data that it hasn't seen. The opposite is also true as what is not desired is for the error value of equation (9) to be too high as this would mean that the MLP trained would not have learnt enough information and this would result in poor predictions again. There is a fine balance between under training an MLP and over-training an MLP.

Secondly, there is an optimum number of hidden units (neurons) in the hidden layer. This number of units also influences the training of an MLP as if too few neurons are used then the networks' ability too learn information is reduced and thus its predictions will not be as accurate as they can be. Again, if too many neurons are used then the network would retain too much information about the training data and again the network would be over-trained resulting in the network producing very erroneous predictions again.

Thirdly, the number of training points used also had to be experimented with. If too many training points were used then it would mean that the MLP would have too much information to learn and this would make it very susceptible to forgetting information and produce poor predictions. Again, if too few training points are used then the MLP would not have learnt enough information and again it would produce poor predictions. There are different optimisation techniques that could also be used to minimize equation (9) and there is also a weight decay value and these could also be experimented with although it must be stated that the weight decay value didn't have as much an impact on the predictions as would be expected.

The parameters were experimented with at various different values and the best values for the parameters can be seen in Table 5.

Table 5: Best MLP Parameters

Parameter	Value or Approach
Number of Training Points	400
Number of Hidden Units	10
Number of Training Cycles	300
Optimisation Technique for Equation (10)	Scaled Conjugate Gradients (see Nabney [14] for details)
Weight Decay Value	0.05

The MLP was trained using these parameters and results were obtained. The results can be seen in Table 6 with a plot of the predictions and with the actual values for 100 of the test points can be seen in Figure 8.

Table 6: Best MLP Results

TP	Max Error (%)	Mean Error (%)	Error less than 10%	Error less than 5%
300	2585	52	139	76

TP = no. of test points used, Max Error = Maximum error between the actual output and that predicted by the network in the *TP* point test set used, Mean Error = average error of the size of the test set used (TP), Error less than 10% = amount of outputs given by the network that fell within an error of 10%, Error less than 5% = amount of outputs given by the network that fell within an error of 5%.

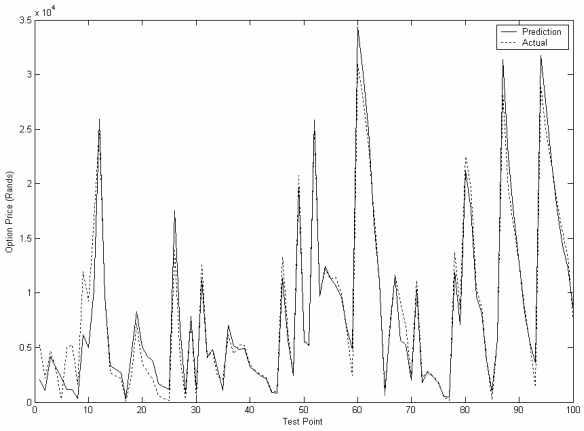

Figure 8: Plots of Actual and Prediction Values for the MLP Trained

What the results show is that the maximum error was 2582% but this translated to a price of R1409.00 when the price of the option should only have been R54.50 (the exchange rate is about R6/$1) but this is not necessarily reflective as this point of data could be considered an anomaly (the data point is an outlier meaning it wasn't in keeping with the training data) and these points were not removed from the data set. Since American option prices are susceptible to market forces, some options that are not in demand (for call options it would mean that the strike price is too low) would have low prices and it would be difficult to predict the price of these options and so very low option price can be considered to be anomalies. What is encouraging is that almost 50% of the test points used produced errors of 10% or less and just over a quarter of the test data produced an error pf 5% or less and the average was pushed up by the error produced by these anomalies.

B. Support Vector Machines

There was also a toolbox obtained for the testing of this training algorithm (see Gunn [18]) and it could also be used with MATLAB®.

As with MLP's, there are also several factors that influenced the predictions that SVM's produce. There is firstly the capacity value which tries and control over training of the algorithm (see Section VI.B). There is also ε-insensitivity (ε) which controls the deviation of certain points (see Section VI.B). There are also several kernel functions offered by the toolbox. Various were tried until the best was found. The number of training points is also a factor in this algorithm for the same reason as that for MLP's.

It was found that due to the computational power needed by the SVM network training, 400 points of data sufficed for the network training otherwise it took too long to train the network. At first a polynomial kernel of degree 2 was used with 1000 data points was used to train the data but this caused the network to take 1 hour and 40 minutes to train (on a Pentium 4 1.8GHz laptop with no other software running) and the results were poorer than with only 400 training points used (with the same values for capacity and ε being used with 400 and 1000 training points). At 500 training points, the time to train the network was double than at 400 points and the results were also poorer than at 400 points.

The best SVM parameters were found and the values for the parameters and the kernel function used can be seen in Table 7.

Table 7: Best SVM Parameters

Parameter	Value or Approach
Number of Training Points	400
Capacity	10
ε-insensitivity	0.005
Kernel Function	rbf with sigma at 1 (see Gunn [18] for details)

The results of the SVM algorithm with these parameters can be seen in Table 8 with the plot of 100 of the test points with actual values and prediction plotted in Figure 9.

Table 8: Best SVM Results

TP	Max Error (%)	Mean Error (%)	Error less than 10%	Error less than 5%
300	1152	34.4	138	94

TP = no. of test points used, Max Error = Maximum error between the actual output and that predicted by the network in the *TP* point test set used, Mean Error = average error of the size of the test set used (TP), Error less than 10% = amount of outputs given by the network that fell within an error of 10%, Error less than 5% = amount of outputs given by the network that fell within an error of 5%.

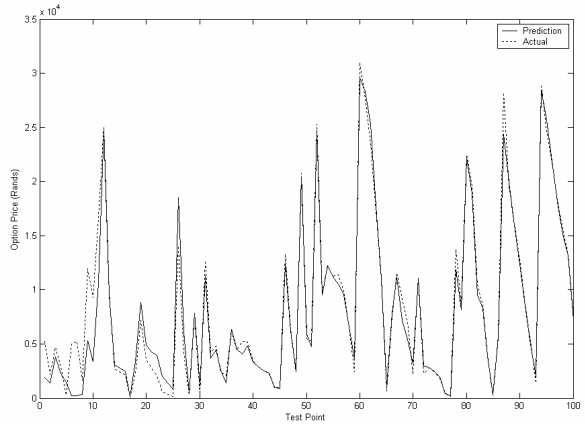

Figure 9: Plots of Actual and Prediction Values for the Best SVM Trained

We can see that the SVM algorithm produced a much lower average error and a much lower maximum error as well. The number of predictions with 5% or less error was also increased from a quarter to almost a third of the data with 94 of the test points now having an error of 5% or less.

In conclusion we can see that the SVM algorithm outperformed the MLP NN but this comes at a bit of a price. The SVM took about 6 minutes to train and the MLP took only 5 seconds to train. However, this is a small price to pay for obtaining a more accurate pricing model. In fact the algorithms only require training once and once they have been trained subsequent prices can then be produced. However, for future studies time does play a role as more studies can be attempted on algorithms that take less time to run.

C. Bayesian Multi-Layer Perceptrons

The NETLAB toolbox provides a function which implements Metropolis-Hastings sampling and this was the function used to sample for the weights and bias values for the MLP algorithm. There were example files that demonstrated this in the toolbox and one of these was edited and modified to fit the option pricing problem at hand.

In addition to the parameters for MLP's, there are also certain parameters that come with Bayesian techniques. The number of hidden units was also experimented with an optimal number of weights was found, however, the number of training cycles was not experimented with as Bayesian techniques offer the optimisation routine through their sampling and so no scaled conjugate gradient or other method is needed to optimise equation (9). This is done by the sampling algorithm used (in this case the Metropolis-Hastings algorithm). With Bayesian techniques there is the number of samples to experiment with as well. In reality, the more samples the better the predictions will be but too few would result in very erroneous predictions. There is also an amount of rejected samples. This also had to be experimented with and is the number of samples that are rejected and not considered by the algorithm to find predictions and bounds. This is usually required (especially if too few samples are used) because the average of the predictions is the actual prediction and since the sampling starts at random values for weights and biases then the average will be very far from the actual if the initial samples in the chain are considered. This is because the initial samples may take the average very far away from the actual outputs as they are initially randomly generated and there is no criteria for finding these values other than they must be close to zero (see Nabney [14]).

What was also experimented with, for the Metropolis-Hastings sampling method, is the step size. This is the size of the step taken to generate a new sample and so it is intuitive to note that if the step size is made too large then samples in between the current sample plus the step size, may be missed out and so it is desired not to have too large a step size. However, it is also not desirable to have too small a step size as then not enough of the search space is explored unless many thousands or even hundreds of thousands of samples are used but this would result in the sampling taking too long if too many samples are required. Therefore there is a balance between the step size and the number of samples.

The parameters used to gain results can be seen in Table 9.

Table 9: Bayesian MLP Parameters

Parameter	Value or Approach
Number of Training Points	300
Number of Hidden Units	40
Weight Decay Value	0.05
Number of Samples	1000
Number of Omitted Samples at Start of Chain	500
Step Size	0.0005

The results for the trained Bayesian MLP can be seen in Table 10 with the plots of 50 of the test points of actual values, predictions and bounds in Figure 10.

Table 10: Best Bayesian MLP Results

TP	Max Error (%)	Mean Error (%)	Error less than 10%	Error less than 5%	Ave. Dev.
300	3855	64	112	56	281

TP = no. of test points used, Max Error = Maximum error between the actual output and that predicted by the network in the *TP* point test set used, Mean Error = average error of the size of the test set used (TP), Error less than 10% = amount of outputs given by the network that fell within an error of 10%, Error less than 5% = amount of outputs given by the network that fell within an error of 5%, Ave. Dev. = the average deviation and thus the magnitude of the bounds above and below the prediction.

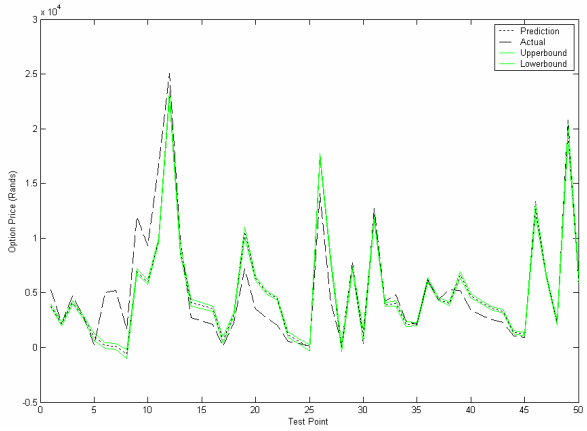

Figure 10: Bounds, Predictions and Actual Values for Bayesian MLP's

We can see that the bounds are quite small thus making the technique quite successful but the average and maximum errors were increased when compared to standard MLP's. This however is not too concerning as Bayesian techniques have been shown to converge to their standard counterparts if enough samples are used. What is attempted here is to show that the bounds are quite small. That is the true test of the Bayesian inference as now we can see whether a certain amount should be added or subtracted from the prediction to make the prediction more accurate. We will now see if the bounds for Bayesian SVM's were less broad than that for Bayesian MLP's.

D. Bayesian Support Vector Machines

In the SVM toolbox, there was no Metropolis-Hastings algorithm and so what had to be done was to integrate the SVM toolbox and the NETLAB toolbox [22]. For Bayesian MLP's the error function (optimisation function of equation (9)) was included in the toolbox and so the Metropolis-Hastings algorithm could be easily used to sample for weights. However the SVM toolbox provided no such error algorithm and so one had to be coded in MATLAB®.

For SVM's the parameters that were to be found were only the beta values. The bias values found from the training of standard SVM's, found that the bias values were always found to be zero and so there was no need to sample for the bias values. The bias values were just kept at zero.

Bayesian techniques were applied a bit differently to SVM's than to MLP's. With MLP's, random values were assigned to the optimisation variables (weights and biases) but with SVM's if random values were assigned then many more samples would have to be obtained for the optimisation variables. So what was decided was to train a standard SVM and then use the values for the optimisation variables obtained (beta values) as the first sample for the Metropolis-Hastings algorithm. What this meant was that there were no omitted samples needed as the samples were fairly close to the starting point provided by the initial SVM trained.

As with Bayesian MLP's, the number of samples also had to be experimented with especially since it can take very long to train SVM's and obtain an output from an SVM using its output function. The step size also had to be experimented with for the same reasons as that for Bayesian MLP's. The capacity value and the ε-insensitivity value used is that shown in Table 7. Otherwise the parameters used can be seen in Table 11.

Table 11: Best Bayesian SVM Parameters

Parameter	Value or Approach
Number of Training Points	50
Samples	10000
Step Size	0.05

It must be noted that only 50 training points were used as opposed to 400 from standard SVM's. This is the case because if 400 points were used then the Bayesian SVM algorithm would take 8 times as long as with 50 points and the results were only marginally better.

The results of the Bayesian SVM approach can be seen in Table 12 with a plot of 50 of the test points used for predictions, actual values and bounds in Figure 11.

Table 12: Best Bayesian SVM Results

TP	Max Error (%)	Mean Error (%)	Error less than 10%	Error less than 5%	Ave. Dev.
300	1091	53	82	42	238

TP = no. of test points used, Max Error = Maximum error between the actual output and that predicted by the network in the *TP* point test set used, Mean Error = average error of the size of the test set used (TP), Error less than 10% = amount of outputs given by the network that fell within an error of 10%, Error less than 5% = amount of outputs given by the network that fell within an error of 5%, Ave. Dev. = the average deviation and thus the magnitude of the bounds above and below the prediction.

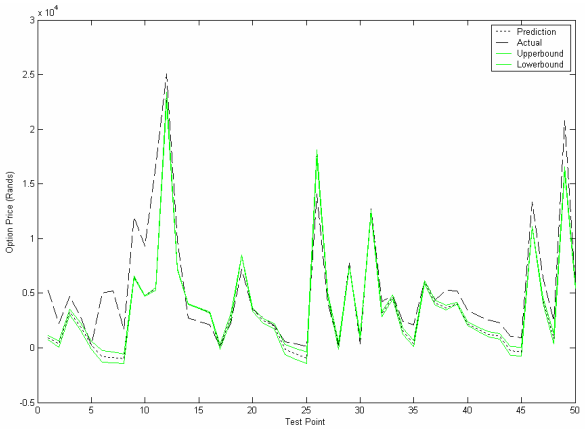

Figure 11: Bounds, Predictions and Actual Values for Bayesian SVM's

From the results for Bayesian SVM's we can see that the average error has been raised (when compared with standard SVM's) but again the usefulness of the Bayesian framework is for use of the bounds to add or subtract some value to the price and obtain a more accurate price. In fact if more than 10000 samples were used then the 34.4% average error mark (the level for standard SVM's) could be achieved.

Better error values were obtained with smaller step sizes but then the bounds were too small and weren't indicative of what they should actually be.

In conclusion, Bayesian SVM's out-performed Bayesian MLP's with regards to average error and due to the fact that the bounds were smaller for Bayesian SVM's than Bayesian MLP's. However, again, this came at a price because the Bayesian MLP trained took only 44 seconds to produce results where as the Bayesian SVM took 2.5 hours.

VII. RECOMMENDATIONS

From all the results obtained, it can be seen that the models cannot be used, yet, to accurately price American options but the new method of SVM's has provided a good grounding for further research so that a more accurate pricing model can be obtained. Here, some recommendations are outlined so that further work can be done using both MLP's and SVM's:

1. In the training algorithms for both SVM's and MLP's there are certain parameters that were found by trial and error (see Sections VI.A and VI.B), an optimisation technique (such as Particle Swarm Optimisation or Genetic Algorithms) could be used to better find these parameters by minimizing the error on the *unseen* data. This may provide a problem with SVM's because of the time it takes to train them but as computational speed increases, this will make the process faster and more probable.

2. For the Bayesian framework, add one standard deviation on top of the predictions and subtract one standard deviation form the predictions and then note if this has improved the errors found or if it hasn't. This should be done once more samples are generated for both Bayesian MLP's and Bayesian SVM's so that better initial predictions can be found (i.e. closer to the average errors found by the standard MLP's and SVM's).

3. Use an optimisation technique to find the best value for the step size (used for both Bayesian MLP's and Bayesian SVM's) according to some error function so that the most indicative bounds can be found and so that the average error can be minimized.

4. For SVM's, beta values are always between the capacity value and the negative of the capacity value (for the capacity at 10, beta values range between -10 and 10) and it was found by many of the standard SVM' trained that many of the beta values were either -10 or 10. In the Bayesian framework, when sampling it meant that these values were then deviated form 10 or -10 for many of the samples. It may be very interesting to note the results if only some of the beta values are sampled for and not all thus leaving the ones at the extremes at either -10 or 10 and only sampling for the other Beta values. This may produce SVM results with better error results and smaller values for the bounds as well.

5. Remove all data points to be considered anomalies from the data set. This would allow the techniques to provide better average error results and make the training algorithms learn the data much better.

6. Use more inputs to all training techniques. Extra inputs could have been the spot price and the risk-free rate of interest. Although it was chosen to exclude these, it may be interesting to see if these inputs actually would improve error performance. Another input could be the actual Black-Scholes price for the option. This may aid the techniques and aid in the predictions. Another example of an input could be liquidity of the option. Some options are more liquid than others (are traded more thus making them easier to sell) and these make the price of these options slightly higher. This liquidity factor could be added as an input to further improve the predictions.

All these recommendations can be attempted and it is believed that an accurate option pricing model can eventually be achieved and used in the market.

VIII. CONCLUSION

When considering everything that has been implemented, we can conclude that the new technique of SVM's has outperformed the more traditional predictive tool of the MLP. This was the case found in both the Bayesian framework and

external to the Bayesian framework. This did come at a price of computational time but it must be said that this is a small price to pay to obtain a more accurate pricing model. In fact this time factor only comes into play for future studies because once an accurate pricing model is obtained the technique is only needed to be trained once and then subsequent predictions can be obtained with very little computational cost. There are several recommendations made and these emphasize the usefulness of computational intelligence methods in this field and that an accurate model is attainable with further research.

IX. ACKNOWLEDGEMENTS

The authors would like to thank the National Research Fund (NRF) in South Africa for their financial support and making this research possible.

The authors would also like to thank the University of the Witwatersrand, Johannesburg for allowing the use of computer facilities and licenses for the software needed.

REFERENCES

[1] S. Ross, R. W. Westerfield, B. D. Jordan and C. Firer, *Fundamentals of Corporate Finance 2nd South African Edition*, McGraw-Hill Book Company, Sydney, Australia: 2001.

[2] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, vol. 81, 1973, pp. 637-659.

[3] R. A. Jarrow and S. M. Turnbull, *Derivative Securities 2nd Edition*, South-Western College Publishing, U.S.A.: 2000.

[4] B. D. Ripley, *Pattern Recognition and Neural Networks,* Cambridge University Press, 1996.

[5] D. L. Hudson and M. E. Cohen, *Neural Networks and Artificial Intelligence for Biomedical Engineering,* Wiley-IEEE Press, Sep. 1999.

[6] T. Marwala and M. Lagazio, "Modelling and Controlling Interstate Conflict", *IEEE International Joint Conference on Neural Networks,* July 25-29 2004, Budapest, Hungary, pp. 1233-1238.

[7] J. M. Hutchinson, A. W. Lo, and T. Poggio, "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks," *Journal of Finance*, Vol 9, No. 3, pp. 851-889, 1994.

[8] M. J. Morelli, G. Montagna, O. Nicrosini, M. Treccani, M. Farina, and P. Amato, "Pricing Financial Derivatives with Neural Networks," *Physica A: Statistical Mechanics and its Applications*, 1 July 2004, Vol. 338, Issues 1-2, pp. 160-165.

[9] D. L. Kelly, "Valuing and Hedging American Put Options Using Neural Networks," Working Paper, Carnegie Mellon University, PA, December 1994.

[10] T. B. Trafalis, and H. Ince, "Support Vector Machine for Regression and Applications to Financial Forecasting," IEEE-INNS-ENNS International Joint Conference on Neural Networks, Como, Italy, July 24-27, 2000.

[11] K. Chan, T. Lee, P. A. Sample, M. H. Goldbaum, R. N. Weinreb, and T. J. Sejnowski, "Comparison of Machine Learning and Traditional Classifiers in Glaucoma Diagnosis", *IEEE Transaction on Biomedical Engineering*, Vol 49, No. 9, pp. 963-974, Sep. 2002.

[12] W. Chu, S. S. Keerthi and C. J. Ong, "Bayesian Support Vector Regression Using a Unified Loss Function", *IEEE Transactions on Neural Networks*, vol. 15, no. 1, 2004, pp. 29-44.

[13] J. C. Hull, *Options, Futures and Other Derivatives*, Prentice Hall, Upper Saddle River, New Jersey, U.S.A.: 2003.

[14] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition*, Springer-Verlag, London, Great Britain: 2003.

[15] K. M. Hornick, M. Sticnhcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol. 2, No. 5, 1999, pp. 359-366.

[16] M. M. Polycarpou, "Online Approximators for Nonlinear System Identification: A Unified Approach", in C. Leondes, *Neural Network Systems, Techniques and Applications, Volume 7: Control and Dynamic Systems*, Academic Press, California, 1998.

[17] J. Joachims, "Making large-scale SVM learning practical" In B. Scholkopf, C.J.C. Burges and A.J. Smola, editors, *Advances in Kernel Methods-Support Vector Learning,* pp. 169-184, Cambridge, MA, 1999, MIT Press.

[18] S. R. Gunn, *Support Vector Machines for Classification and Regression.* Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, University of Southampton, U.K. 1998.

[19] C. J. C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition*, Kluwer Academic Publishers, Boston, pp 1-43.

[20] T. H. Mirer, *Economic Statistics and Econometric, Third Edition.* U.S.A: Prentice Hall, Inc., 1995, pp. 209-218.

[21] *South African Futures Exchange,* Available: http://www.safex.co.za.

[22] M. M. Pires, Masters Thesis, School of Electrical and Information Engineering, University of the Witwatersrand, 2005, Appendix C: Software Code, Toolboxes, Data and Output Files.

Chapter 5: Conclusion and Further Work

In conclusion, it can be said that Support Vector Machines (SVM's) out-performed the Multi-Layer Perceptron (MLP) in terms of error analysis. This was found both using the maximum-likelihood framework and the Bayesian framework. Without the Bayesian framework, SVM's had an almost 20% better accuracy when pricing the call options even though they took much longer to train. Within the Bayesian framework, SVM's still produced better accuracy than Bayesian MLP's even though the time to train was much longer. Time to train certain algorithms may not be seen as a problem because the algorithms only need to be trained once to produce subsequent prices and so accuracy is much more important than the time needed to train the different algorithms.

What can also be said is that Computational Intelligence is a field that needs to be explored much more in the options pricing field. Even though an accuracy close to 70% with SVM's was achieved, the accuracy level needs to be somewhere near the 95% mark for the model to be used by the South African Futures Exchange (SAFEX). There are several steps that can be taken next so that the model can be further improved. A list of suggestions are as follows:

1. Many of the MLP's and SVM's trained were done by choosing certain parameters (such as capacity and number of training points for SVM and number of hidden units and number of training points for MLP's) by trial and error. What would be better would be to use some optimisation technique to do this and set up an error function so that the optimal parameters could be found. It is suggested that an optimisation techniques such as Particle Swarm Optimisation (PSO) or Genetic Algorithms (GA) be used.

2. Within the Bayesian framework, what can be done is subtract or add one standard deviation to the actual predicted price and then do some error analysis to see if the predictions have been improved.

3. Within the Bayesian framework, generate more samples for both Bayesian MLP's and Bayesian SVM's so that the predictions that are found can have average error values close to that found by standard MLP's and SVM's. This would make the Bayesian framework more useful. This may take much time but can be attempted by setting up the algorithms to run for a long period of time.

4. Within the Bayesian framework, an optimisation technique can be used again to find the best value for the step size. Again PSO or GA can be used by setting up an appropriate error function.

5. For SVM's, beta values are always between the capacity value and the negative of the capacity value (for the capacity at 10, beta values range between -10 and 10) and it was found by many of the standard SVM's trained that many of the beta values were either at the extremes of -10 or 10. In the Bayesian framework, when samples were obtained, it meant that these values then changed from these extremes and were thus not at their optimal values. It may be very interesting to note the results if only some of the beta values are sampled for thus leaving the ones at the extremes at either -10 or 10 and only sampling for the certain Beta values. This may produce SVM results with better error results and smaller values for the bounds as well.

6. The data could be analysed closer so that anomalies can be removed and so that only relevant data can then be used in the training and testing of the algorithms. Anomalies are those data points that are different to normality and thus cause the networks to not be able to learn these points very easily and thus cause large errors in the predictions. Some work on the detection of outliers (anomalies) has been done [1].

7. Use more inputs to all training techniques. Extra inputs could have been the spot price and the risk-free rate of interest. Although it was chosen to exclude these, it may be interesting to see if these inputs actually would improve error performance. Another input could be the actual Black-Scholes price for the option. This may aid the techniques and aid in the predictions. Another example of an input could be liquidity of the option. Some options are more liquid than others (are traded more thus making them easier to sell) and these make the price of these options slightly higher. This liquidity factor could be added as an input to further improve the predictions.

8. A different stopping criteria for SVM's may make the SVM's train faster (as the number of support vector required is reduced) and thus provide more scope for work with SVM's in this field. A different stopping criterion has been used before [2] with a classification problem but what can be done is to see whether the same criterion can be applied to this regression problem.

Therefore there are many suggestions as to future work and how the models can be improved to gain a more accurate pricing model. Overall, the research was successful in determining which methods should be used over others and an idea of the range of parameters that should be used when training the different algorithms. This work shows insights into pricing options and what affects the prices of American options and also shows insights into which Computational Intelligence methods perform better than others.

REFERENCES

[1] J. R. Greene, "A Simple Method for Visualising Labelled and Unlabelled Data in High-Dimensional Spaces", *Proceedings of the Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa,* November 25-26 2004, Cape Town, South Africa, pp. 45-49.

[2] N. Muller, "Fast Stopping in Support Vector Machine Classifiers", *Proceedings of the Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa,* November 25-26 2004, Cape Town, South Africa, pp. 41-43.

Appendix A:
Hedging with Options and Other Contracts

Abstract

Companies around the world are exposed to certain kinds of risk and they are always looking for ways to protect themselves against this risk (hedging). For example a gold mining company in South Africa is exposed to changing gold prices and usually in South Africa, most of the gold is exported and so gold mining companies are exposed to risk due to the Rand exchange rate with the Dollar. Hedging with forwards, futures and options is presented with emphasis on the use of options for hedging. This Appendix shows how useful options can be and indicates how important it is to obtain an accurate price for an option.

I. INTRODUCTION

In recent times, the world has seen a tremendous increase in the volatility of interest rates, exchange rates and commodity prices. Commodity prices were stable up until the 1960's but since then there has been a rapid increase in commodity prices [1]. Exchange rates in South Africa have been even more volatile. The Rand was only allowed to float against the U.S Dollar after 1979 and the 1990's and early 2000's has seen a very volatile Rand/Dollar exchange rate. For example, in late 2001 and early 2002, the Rand was as weak as R12/$1 and in 2004 it hovered around the R7/$1 mark, an almost doubling in strength against the Dollar. Interest rates have also been very volatile in South Africa. For example, 1998 saw an increase of over 250 basis points in the interest rate and 1999 saw a decrease of 150 basis points (1 basis point equals 1 hundredth of 1%).

Therefore companies can be exposed to these volatilities. For example, a gold mining company is exposed to changes in the gold price (if the gold price increases then the company will make more money and if it decreases then it will make less money). Most gold mining companies in South Africa export the ounces mined. In this case then they would prefer a weaker Rand because the gold price is usually (if the Rand is weak against the dollar then the gold mining company will receive more Rands per ounce of gold sold). Also if the company has substantial loans from banks in South Africa then they will also prefer a lower interest rate as then the company will pay less interest over to the lender. From this example we can see that the gold mining company is thus a seller of the commodity and a buyer of particular exchange and interest rates. We can now construct risk profiles for the gold mining company. The risk profiles show how the company is exposed to risk from these three volatile measures. There is a separate risk profile for each of these measures. The profiles for the gold mining company can be seen in Figure 12 and Figure 13 below:

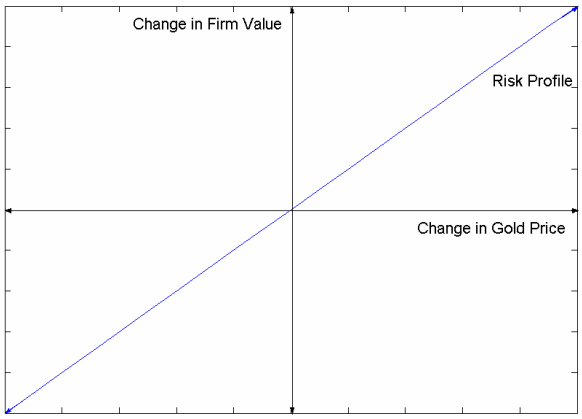

Figure 12: Risk Profile for Commodity Price Change

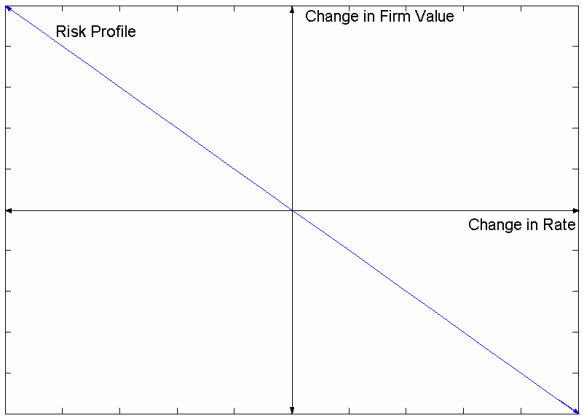

Figure 13: Risk Profile for Rate Change

Figure 13 is the risk profile for the gold mining company for both exchange rate and interest rate risk. Therefore a gold mining company in South Africa with some long term debt will have three different risk profiles, one as in Figure 12 and two as in Figure 13. Note that a buyer of a commodity will have a risk profile as in Figure 13 and a seller of rates will have a risk profile as in Figure 12 because buying and selling risk profiles are the same no matter the asset that is being bought or sold.

In essence what these profiles tell us is that an increase in commodity price (in the case of our example, gold is the commodity) will result in the company making money and thus a positive increase in value and a decrease in either rate (either the Rand weakens against the Dollar or the interest rate drops) will result in the firm making money. Of course the problem is not when the firm makes more money the problem is when the firm loses money and thus decreases in value. From the profiles we can see that this happens when the commodity price drops or the Rand strengthens against the Dollar or the interest rate rises. Companies hedge (protect) against this risk through the use of forwards, futures or options.

II. FORWARDS, FUTURES AND OPTIONS

A. Forwards and Futures

A forward contract is a legally binding contract between two parties calling for the sale of an asset or product in the future at a price agreed upon today [1]. In terms of the gold mining company, what this means is that they could agree to sell gold at a particular price to their customers (a price low enough to be attractive to the customer but not too low so that they can still make a profit on the gold sold). They could also then agree with the bank providing the loan and the exchange rate to fix the rates slightly higher than currently offered (again, just high enough to seem attractive to the bank but not too high otherwise the gold mining company may be put out of business). These contracts would be desirable to the gold mining company if it foresaw an increase in rates and a decrease in gold price. There is a downside to having these contracts though. If say commodity prices rise and rates fall (thus favourable situations occur in the market) and the contracts have already been agreed upon, then the gold mining company will not benefit from this and as a result the net gain of the company in taking forward contracts is always zero. Therefore the firm's value, with the contracts, will remain the same whether a favourable situation or unfavourable situation occurs in the market. The payoff that the company will experience is shown by payoff profiles. The payoff profiles for these forward contracts and the hedged profiles (the sum of the risk and payoff profiles thus showing net company gains) can be seen in Figure 14 and Figure 15 below:

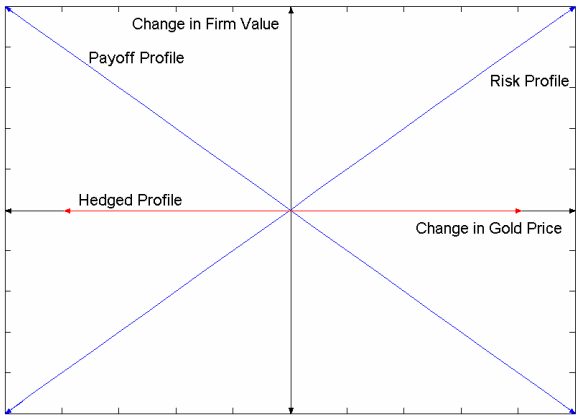

Figure 14: Hedged and Payoff Profile for Commodity Risk

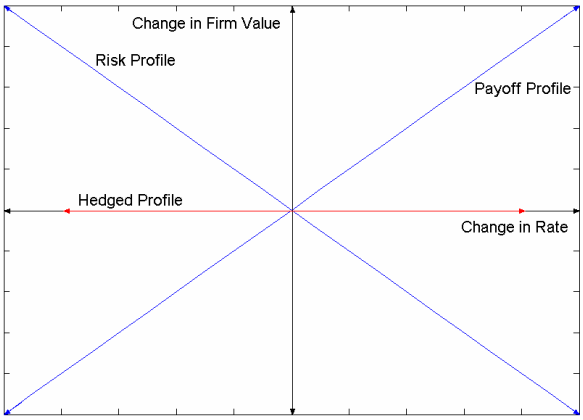

Figure 15: Hedged and Payoff Profile for Rate Risk

Futures are very similar to forward contracts. The difference with Futures is that the gains and losses of the contracts are realised on a daily basis and with forwards they are only realised on a certain later date.

B. Options

Options are slightly different contracts. Options give the owner the right, not the obligation, to buy (a call option) or sell (a put option) an underlying asset for a fixed price (the strike price) for a certain period of time [1]. What this means, in terns of forwards and futures, is that companies can still hedge against financial risk as with forwards and futures but can also benefit from favourable market situations (if the gold price increases and rates drop for our gold mining company example) by not exercising the right that they have. This makes options valuable assets to companies and companies have to pay to have this right. This price that they pay is known as the option price.

There are American and European options. European options only allow the exercise of the option on a later date (known as the maturity date) and American options allow the exercise of the option on any date leading up to the maturity date. A European option pricing model has been developed by Fischer Black, Myron Scholes and Robert Merton and they developed what is known as the Black-Scholes model [2]. Their equations for call and put options are seen below [3]:

$$c = S_0 N(d_1) - Ke^{-rT} N(d_2) \tag{1}$$

and

$$p = Ke^{-rT} N(-d_2) - S_0 N(-d_1) \tag{2}$$

where

$$d_1 = \frac{\ln(S_0 / K) + (r + \sigma^2 / 2)T}{\sigma\sqrt{T}} \tag{3}$$

and

$$d_2 = \frac{\ln(S_0 / K) + (r - \sigma^2 / 2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T} \tag{4}$$

N() is the cumulative probability distribution function, S_0 is the spot price (current price today) of the underlying asset, K is the strike price agreed upon, r is the risk-free rate of interest (this interest rate is usually that denoted by government bonds in a politically stable country [1]), T is the time to maturity in number of years and σ is the volatility of the underlying asset (volatility implies how much the underlying asset has deviated in the past).

The problem is that American options cannot be priced with these formulas and American options are used in South Africa. American options introduce a second random process as it is not known when it will be optimal for the owner of the contract to exercise it. Intuitively we can say that American option prices would be a function of the same factors that are used for European options. American options are more valuable than European options [4] as they give the owner more flexibility with regards to when to exercise. Companies all over the world use options to hedge against financial risk. Since options give the owner the right, not the obligation, to exercise the contract, it makes options very valuable and they will have different payoff and hedged profiles from forwards and futures. In the case of our example, a gold mining company would purchase a commodity put option (allowing the company to sell gold for a fixed price), purchase an exchange rate call option (allowing the company to purchase a particular exchange rate) and purchase an interest rate call option (so that they can purchase a particular interest rate). The payoff and hedged profiles for the gold mining company example for options can be seen in Figure 16 (for gold price fluctuation risk) and Figure 17 (for exchange rate and interest rate risk):

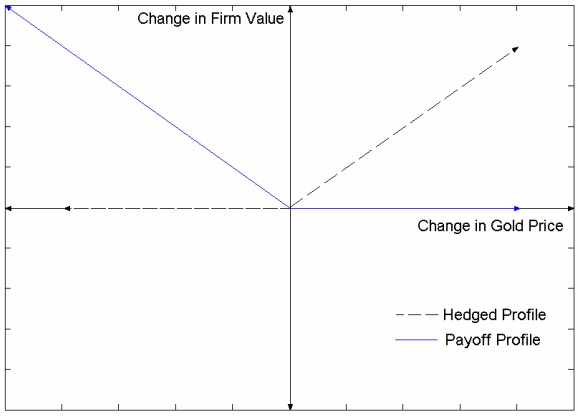

Figure 16: Hedged and Payoff Profiles for Options for Commodity Price Risk

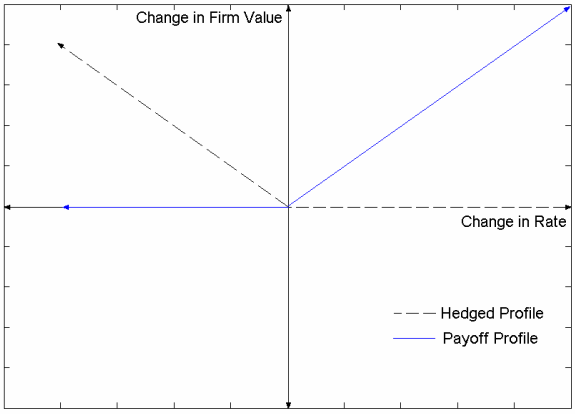

Figure 17: Hedged and Payoff Profiles for Options for Rate Risk

What is not taken into account in Figure 16 and Figure 17 is the option premium (price). This is lost to the owner of the option contract whether or not the option is exercised. Therefore the options will only be exercised if the gains that they provide compensate for this loss and more In fact this will be the only time that the options will be exercised. Therefore pricing of these options is very important to companies all over the world. A question still looms though, if options are not exercised then it means that someone will lose out and lose money because it means that a favourable situation has occurred in the market. For our gold mining example, if the gold price put option is not exercised then it means that the gold mining company is selling gold for more than that stipulated by the option contract and it means that the customer is losing out on a lower gold price stipulated by the option contract. The same thought process is used for the two rate options. If rates decrease to levels below that stipulated by the call options then the company will not exercise the options and it would mean that the bank is losing out on the higher rates stipulated by the call options. What the losing parties can do is buy options themselves. This can be done for interest rates with caps and floors [1].

C. Caps and Floors

For the bank not to lose out on a higher interest rate stipulated by the call option (for our gold mining example) then what is done by the mining company is usually to buy a cap and sell a floor. A cap is an interest rate call option and a floor is an interest rate put option. What this effectively does is give the mining company the right to lower interest rate but it also gives the

bank the right to enforce an interest rate of at least a certain level. This means that the interest rate paid by the mining company will always be between the floor and cap price and never lower or higher. The same can be done for exchange rates by buying an exchange rate call option and selling an exchange rate put option. The same can also be done with commodities. The gold mining company would buy a gold put option and sell a gold call option thus allowing their customers to not pay above a certain price and allowing the gold mining company to never sell below a certain price. This effectively makes the rates and prices trade between a certain ranges.

III. CONCLUSION

What has been outlined is the usefulness of options and how they are used all over the world to hedge against financial risk. The important issue to note is that option prices are very important to companies as they will not want to pay too much for an option and the seller of the option will not want to receive too little for the option. American options are the ones used in South Africa and since there is not standard to pricing them, their prices are a function of market forces as well. Therefore the option pricing problem is one that has brought much attention and work in the past and is a field that still requires much more work.

REFERENCES

[1] S. Ross, R. W. Westerfield, B. D. Jordan, and C. Firer, *Fundamentals of Corporate Finance 2nd South African Edition.* McGraw-Hill Book Company, Sydney, Australia, 2001.

[2] F. Black, and M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, Vol 81, pp. 637-659, 1973.

[3] J. C. Hull, *Options, Futures and Other Derivatives, 5th Edition.* Prentice Hall, Upper Saddle River, New Jersey, U.S.A., 2003.

[4] R. A. Jarrow, and S. M. Turnbull, *Derivative Securities, 2nd Edition.* South-Western College Publishing, U.S.A, 2000.

Appendix B:
Paper for *Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa*

Option Pricing Using Bayesian Neural Networks

M. M. PIRES
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg, South Africa
m.pires@ee.wits.ac.za

Abstract – **Options have provided a field of much study because of the complexity involved in pricing them. The Black-Scholes equations were developed to price options but they are only valid for European styled options. There is added complexity when trying to price American styled options and this is why the use of neural networks has been proposed. Neural Networks are able to predict outcomes based on past data. The inputs to the networks here are stock volatility, strike price and time to maturity with the output of the network being the call option price. There are two techniques for Bayesian neural networks used. One is Automatic Relevance Determination (for Gaussian Approximation) and one is a Hybrid Monte Carlo method, both used with Multi-Layer Perceptrons.**

I. INTRODUCTION

This document deals with the use of two kinds of Bayesian neural networks applied to the American options pricing problem. Both Bayesian techniques used were used with Mult-Layer Perceptron (MLP) networks. The techniques can also be used with Radial Basis Function (RBF) networks [1] but they were only used with MLP networks here. The two Bayesian techniques used are Automatic Relevance Determination (ARD) (for Gaussian Approximation) and the Hybrid Monte Carlo method (HMC) which will be discussed.

Firstly we need to introduce the notion of an option. An option is the right (not the obligation) to buy or sell some underlying asset at a later date but by fixing the price of the asset now [2]. For someone to have this option, he/she has to pay a fee known as the option price. There are two kinds of options, namely a call and a put option. A call option gives the person the right to buy the underlying asset and a put option gives the person the right to sell the underlying asset [2]. The pricing of either call or put options is equally difficult and something that has brought much research interest.

Black et al. [3] provided equations in 1973 that provided a pricing formula for call and put options. To obtain these equations, several assumptions had to be made. The most important assumption made is that the formulas only held for European styled options [4]. European styled options only allow the exercise of the option on the maturity date (which is the later date that the person is allowed to buy or sell the underlying asset) [5]. What are used extensively worldwide, though, are American styled options where the person is allowed to buy or sell the underlying asset at any date leading up to the maturity date. This introduces another random process into the pricing of the option (because it cannot be predicted when the exercise of the option will occur) and so the pricing of these kind of options is much more complex than European styled options [6].

Neural Networks (NN's) are a form of prediction based on trends that have occurred in the past. The outputs of the network are that which are to be predicted and the inputs are chosen as variables that affect the outputs in the real world and whose trends can be used to predict the output variables. MLP and Support Vector Machines (SVM's) have been used to price American options [7] and here what will be tested is the effectiveness of Bayesian Neural Networks.

II. BAYESIAN NEURAL NETWORKS

A. Bayesian Techniques

With NN's there is always an error in the predictions made and we thus have

$$y = f(x; w) + \varepsilon \tag{1}$$

where y is the actual output desired, f is the output predicted by the network, ε is the error, w are the weights [1] and x is a vector of inputs. Even if we are given ε and the same network is run twice with the same parameters, we will obtain different weights w both times and thus there is an uncertainty in the training of the networks [1] and this can be attributed to the randomness in the assignment of weights. Generally some complex models try to fit the noise into the predictions which cause problems when trying to predict with unseen inputs (the problem of over training) and thus cause there to be even more error in the predictions [1].

$p(.)$ wherever used from now on is used to denote the probability function from statistics. In the Bayesian approach, the uncertainty in the parameters estimated when training a network is assumed to follow a particular distribution. We first start with a *prior* distribution $p(w)$ which gives us an idea of the parameters before the data is used [1] but this only give us a vague idea as the distribution is quite broad. The prior distribution can be of any kind for example Poisson or Geometric. In this case we will only use a Geometric distribution. We then wish to narrow this distribution down by finding the posterior probability density of the parameters w given a particular dataset D, $p(w|D)$ where

$$p(w \mid D) = \frac{p(D \mid w)p(w)}{p(D)} \quad (2)$$

and $p(D|w)$ is the dataset likelihood and $p(D)$ is the evidence and ensures that the posterior integrates to 1 and is calculated by an integral over the parameter space. Once the posterior

$$p(D) = \int p(D \mid w')p(w')dw' \quad (3)$$

is calculated we can then make a prediction at a new input by first calculating the prediction distribution

$$p(y \mid x^*, D) = \int p(y \mid x^*, w)p(w \mid D)dw \quad (4)$$

where y is the predicted values and then the actual prediction is found by

$$E(y \mid x^*, D) = \int yp(y \mid x^*, w)p(w \mid D)dw \quad (5).$$

$E(.)$ is the expected value in statistical terms. As can be seen from equations (3) and (5), there is an integral involved and the dimensionality of the integral is given by the number of network parameters (weights) and this is not analytically possible and simple numerical algorithms break down [1]. Therefore approximations to the posterior are made (the toolbox used to train Bayesian Neural Networks is the NETLAB toolbox used with MATLAB®) and this is known as the evidence function in NETLAB and is used together with a Gaussian Approximation and ARD (see Section IIB). What can also be used is Hybrid Monte Carlo (HMC) methods combined with Monte Carlo sampling used for integral approximation [1] (see Section IIC).

The main reason for the use of Bayesian techniques is simply to reduce the uncertainty in the weights and thus try to reduce the problem of over fitting (i.e. over fitting occurs when a network predicts badly because it is trained too much to its training data and predicts badly with unseen inputs [1]). Bayesian techniques do reduce the problem of over fitting as has been proved by Nabney [1]. In NN's there is a need to optimise the network and thus reduce the error function [8]. In Bayesian techniques this is done by obtaining a posterior distribution for the weights so that they can only be found within a particular distribution thus narrowing the search for the optimal weight values [1]. Bayes' theorem helps us do this but there are large integrals and there are several ways of evaluating these integrals. There are Gaussian Approximations and HMC.

B. Automatic Relevance Determination

The prior distribution is chosen to be Gaussian [1] and thus is of the form

$$p(w) = \frac{1}{Z_W(\alpha)} e^{-\frac{\alpha}{2}\sum_{i=1}^{W} w_i^2} \quad (6)$$

where the normalization constant $Z_W(\alpha)$ is

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{W/2} \quad (7).$$

α is known as the hyperparameter because it is a parameter for the distribution of other parameters. It is then helpful to have different hyperparamaters, one for each set of the weight sets $W_1 \ldots, W_g$. The way to choose these different hyperparameters is to have values for them associated to how important each input variable is. This is known as Automatic Relevance Determination (ARD).

ARD is used because there is often the need to find the relevance of certain input variables. This is not easily done if there are hundreds of input variables. In Bayesian NN's we associate each hyperparameter with an input variable. Each hyperparameter represents the inverse variance of the weights and so the lower the value for a hyperparameter associated with a particular input, the more important that input is in the prediction process because it means that large weights are allowed [1].

C. Hybrid Monte Carlo Method

As stated before, Monte Carlo methods can be used to approximate the integrals involved in Bayesian techniques rather than using a Gaussian approximation with ARD and an evidence procedure [1].

Since there is an uncertainty in the process, we need to find the predictive distribution, i.e. the distribution that represents the possible outcomes of the network due to the uncertainty in the weights [1]. This distribution is an integral but in Monte Carlo methods it is approximated to a sum

$$p(y \mid x, D) = \frac{1}{N}\sum_{n=1}^{N} p(y \mid x, w_n) \quad (8).$$

where N is the number of samples chosen by the trainer of the network and w_n is the sample of weight vectors. These samples of weights can be chosen through different methods. A Metropolis-Hastings algorithm can be used to sample these weights but has proved to be very slow. This is because the method makes no use of gradient information and for NN's the method of error back-propagation provides an algorithm for evaluating the derivative of an error function and thus optimizing the network more computationally efficiently [1]. Another method that can be used is the Hybrid Monte Carlo (HMC) algorithm for sampling which is the one that is used in this application and makes use of the gradient information.

The HMC algorithm is a sampling algorithm that takes into consideration certain gradient information. The algorithm follows the following sequence of steps once a step size ε and the number of iterations L has been decided upon:

1. *Randomly Choose a Direction λ:* λ can be either -1 or +1 with the probability of either being chosen being equal.
2. *Carry Out the Iterations:* Starting with the current state $(w, p) = (\hat{w}(0), \hat{p}(0))$ randomly selected, where p is a momentum term which is evaluated at each step, we then perform L steps with a step size of $\lambda\varepsilon$ resulting in the candidate state $(\hat{w}(\lambda\varepsilon L), \hat{p}(\lambda\varepsilon L)) = (w^*, p^*)$.

3. The candidate state is accepted with probability $\min(1, e^{(-H(w^*,p^*)-H(w,p))})$ where *H(.)* is the Hessian matrix. If the candidate state is rejected then the new state will be the old state.

These three steps, in essence, describe how the sampling is done so that the summation of equation (8) can be accomplished and so that the posterior distribution can be found and thus allowing the optimisation of the NN. The momentum term *p* can be randomly generated or it can be changed dynamically at each step and there are different ways of doing this [9]. The sets of weights are thus selected or rejected according to the three steps above and the number of samples that are wished to be retained are the number of weights retained. For each set of weights there is a corresponding NN output. The prediction of the network is the average of the outputs.

The usefulness of the Bayesian approach comes into the fact that the prediction comes with certain confidence levels. In fact the prediction mathematically is the same as that of the standard MLP. If we plot the prediction and upper and lower bounds (where the upper bound is the prediction plus the standard deviation of the outputs and the lower bound is the prediction minus the standard deviation of the outputs of the network) then we say that the prediction is known to within a certainty of 68% (because in the normal distribution 1 standard deviation from the mean constitutes 68% of the possible outcomes [10]). This is done for the Gaussian and HMC approaches.

III. Results of Bayesian Neural Networks

A. Automatic Relevance Determination Approach

Data was obtained from the JSE Securities Exchange of South Africa. It was obtained for a particular stock option for the period January 2001 to December 2003. This resulted in there being 3051 points of data that could be used for training and testing of the networks. The inputs to the network were stock volatility, strike price and time to maturity (in days). The output of the network would simply be the call price of an option. Call prices were obtained for different options with there being both high and low prices. What was decided was to use the average of the high and low prices as the actual call price and these are the values used to train and test the network.

There are demos available in the NETLAB toolbox that show the procedure of training Bayesian NN's with the Gaussian Approximation and ARD, and HMC. These demos were edited so that the procedures could be experimented with on the options pricing problem. In the Gaussian Approximation with ARD, it was found that 500 training cycles showed the best results with 1000 data points being used to train the network. The network was tested with 300 data points so that the plots could be easily seen when viewing the error bars. The evidence procedure utilized in the toolbox has a certain amount of cycles associated with it as well and it was found that 10 cycles for this sufficed for the training of the Bayesian NN. The parameters changed were the number of hidden units, the number of loops used to find better hyperparameter values and the value for β that is associated with MLP NN's and is the coefficient of data error associated with the MLP. The results of the Gaussian Approximation approach with ARD can be seen in Table 13.

Table 13: ARD NN Results

β	Hid. Units	Mean Error (%)	Time (s)	n	σ	Alphas
1	25	64.7	52	1	1516	[1.2177 1.2036 0.6417]
10	25	53.16	52	1	1512	[0.8366 0.9051 0.5723]
100	25	61.94	49	1	1354	[1.0101 0.9760 0.4525]
1	50	57.44	104	1	1541	[1.2231 0.9342 1.1528]
10	50	52.72	103	1	1505	[1.5385 0.8931 0.8203]
100	50	61.16	102	1	1485	[0.7763 1.2248 0.9373]
1	25	62.1	105	2	1390	[1.7646 0.9891 1.1858]
10	25	70.49	97	2	1520	[1.8534 0.7471 0.9004]
100	25	58.95	102	2	1433	[1.1214 1.5703 0.4662]
1	50	78.49	191	2	1521	[2.0210 1.5175 0.7859]
10	50	76.56	177	2	1409	[1.7518 1.2180 0.6775]
100	50	61.85	175	2	1456	[1.9264 1.3430 0.7552]

β = coefficient of data error for the MLP, Hid. Units = number of hidden units used in the training of the MLP, Mean Error = average error found by subtracting each prediction from the actual value and multiplying by 100 over the size of the test set used (300), Time = time taken to train the network, n = number of loops used to find the best hyperparameter values, σ = the average size of the bounds for all the outputs (average of standard deviations of output samples), Alphas = hyperparameter values found for the corresponding input to hidden unit weights thus showing the importance of the different inputs.

There was a problem when trying to find the standard deviations of the outputs for the Bayesian NN's using the ARD approach. The function that provides the standard deviations, at times, produced some imaginary numbers so what was done was to search through the standard deviations and replace the imaginary numbers with the first standard deviation value in the array. This got rid of the errors in MATLAB® but showed that the ARD approach does have some bugs. In fact it is said that the Gaussian approximation is the same as the HMC under certain conditions but these conditions are not known and in fact the only reason that Gaussian approximations are used in Bayesian techniques is because they are more mathematically neat than other Bayesian approaches.

As can be seen from Table 13, the network performed the best with the coefficient of data error at 10, with 50 hidden units and the number of loops to find different hyperparameter values only set to 1. The values found for the different hyperparameters show that each input was important in the determination of call prices because each hyperparameter was in the same order of magnitude and there isn't one that is significantly smaller or larger than the others. The time column indicates that the networks didn't take too long to train and that if the number of hidden units was doubled so the time to produce a result also doubled (give or take a few seconds). Other values were tried for hidden units and what was also tried was to use more training data to improve the accuracy of the pricing model. It was found that with 1500 training points and 100 hidden units the mean error was much higher than the values found in Table 13 and also took up to 30 minutes to train. Note that to obtain these results the algorithm had to be run several times with the same parameters so that the best results for these parameters could be obtained, this is due to

the random nature inherent in the algorithm for training the NN as was found with standard MLP's [7]. The standard deviations found for each network trained are quite large and thus the predictions found by the network are known to be within a range of about R3000 with a certainty of only 68%. Therefore we can only say that we know the price to be within quite a large range (of R3000) and only with a confidence of 68%. The outputs for 50 of the 300 test points used and with the corresponding confidence levels for the 2nd network in Table 13 can be seen in Figure 18.

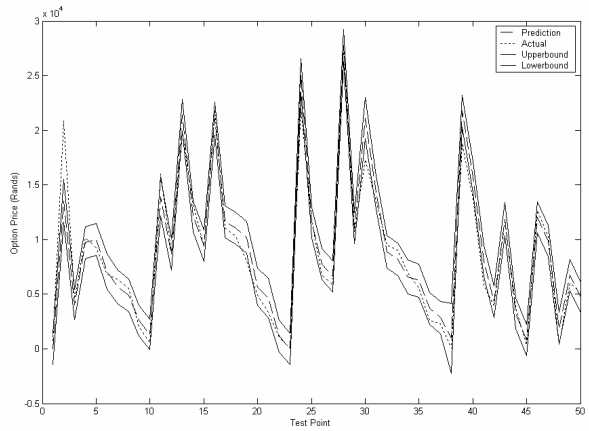

Figure 18: Bayesian NN with ARD results.

* Note the upper and lower bounds are solid and notice they are quite broad.

B. Monte Carlo Approach

The data used to train and test the HMC Bayesian NN was the same data as that used for the ARD approach. Here the coefficient of data error value was not experimented with and was rather kept at a value of 10. The number of hidden units was experimented with as well as the number of initial samples rejected and the number of samples in the HMC procedure. The step size was kept constant at 0.002 because it was found that if it was changed to other values bigger or smaller then the threshold (probability used in the rejection criteria) was not a number (NaN in MATLAB®) and so the procedure didn't work very well in these cases. The number of training points used was also 1000 and the number of points used to test the network was 300. The results for the HMC Bayesian NN approach can be seen in Table 14.

Table 14: HMC NN Results

Rej.	Max Error (%)	Mean Error (%)	Samp.	Time (s)	σ	Hidden Units
100	5241	76.07	100	259	445.95	10
100	5990	95.68	100	444	502.72	20
100	4372	82.76	100	816	699.36	40
100	4378	98.31	400	648	468.71	10
100	5212	77.92	400	1114	575.67	20
100	6719	98.26	400	2104	814.61	40
200	5662	79.21	100	390	401.42	10
200	7618	103.42	100	665	684.75	20
200	4021	91.80	100	1227	680.83	40
200	3849	92.04	400	777	472.08	10
200	4093	78.29	400	1322	591.20	20
200	5836	78.53	400	2451	722.30	40

Rej. = number of samples to be rejected initially (at the start of the Markov chain), Max Error = Maximum error between the actual output and that predicted by the network in the 300 point test set used, Mean Error = average error of the size of the test set used (300), Samp. = number of samples in the HMC method, Time = time taken to train the network, σ = the average size of the bounds for all the outputs (average of standard deviations of output samples), Hidden Units = number of hidden units used in the MLP.

As can be seen from Table 14, the networks took quite sometime to train with 1000 training points. It was attempted to try fewer points for training but just reduced the performance of the network significantly. What was also attempted was to use more hidden units to train the network but this proved to increase the amount of time required to train the network with no improvement in the error analysis. Note that the algorithm for each result in Table 14 was found by training the same network only once. It didn't have to be run several times. The process of training networks in this was still random but the seed used for the random number generator was the same every time and so there was no difference between the results of two networks that were trained with the same parameters. The standard deviations found by each network trained are significantly smaller than that found by the Gaussian approach with ARD. Therefore the predictions of the network are known with a confidence of also 68% to be within a certain range but the range is much smaller and at best the range was R802.84. The outputs for 50 of the 300 test points used and with the corresponding confidence levels for the 1st Network in Table 14 can be seen in Figure 19.

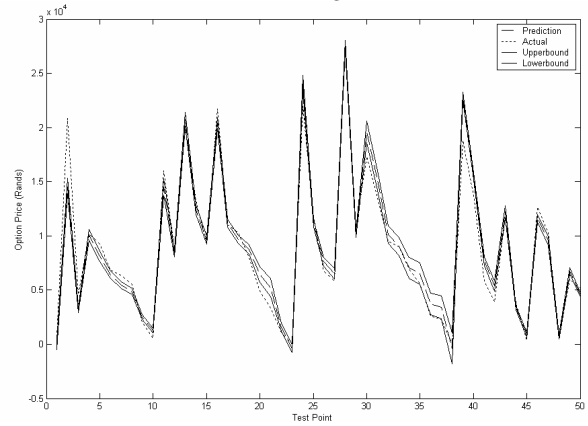

Figure 19: Bayesian NN with HMC results.

* Note the upper and lower bounds are solid and notice that they are much less broad than the bounds for the ARD approach.

IV. Comparison of Bayesian Techniques with Standard Multi-Layer Perceptrons and Support Vector Machines

From the results obtained for the standard MLP and SVM [7], it must be said that the Bayesian techniques applied to NN's didn't provide any improvements. In fact mathematically they are said to be the same as standard NN's but the advantage they bring is the actual confidence levels. With regards to the ARD approach, the best level of mean error was found to be 53% which is very close to the 51% found by the standard MLP trained before. The amount of time taken to train the network was much more than that found by the standard MLP as was to be expected due to the extra functions being utilized in the Bayesian approach due to the approximations inherent in the technique. Compared to SVM it was faster than the 7 minutes taken to train an SVM network but the results were significantly poorer because the average error found by the SVM network at best was 34.4%.

With regards to the HMC approach the best value found for average error over the test set was found to be 76.07%. HMC is mathematically supposed to provide the same results as standard MLP's but it didn't in this case. This is probably because not enough samples were taken when obtaining a prediction. With there being 400 samples the network took up to 40 minutes to train and so for the purposes of this study what was considered to be more interesting is the fact that HMC provided a much narrower band of confidence than that found by the Gaussian approach with ARD. The band produced by the HMC approach was R804.84 which is significantly better than the R3000 found by the ARD approach. Therefore even though the error found by the HMC approach was found to have at best an average of 76.07% we know that the price given by the network is known to be within a band of R804.84 with a confidence of 68%. A drawback is of course the time taken to train the network using HMC. It takes very long but is still more useful than standard MLP's and MLP's with the ARD approach.

In conclusion the best NN method was found to be the SVM method because it produced the best error analysis results and even though it took 7 minutes to train it is worth using in the future. But it must be said that Bayesian NN's do produce confidence levels for the outputs which is still a serious advantage over standard NN's when pricing options. This is because what can be done is to say that a price is provided with this degree of confidence and thus we can then see the implications of adding a bit to the price because we know the confidence or subtracting from the price. Based on this we can see that optimally a Bayesian SVM approach would be favourable and this could be further researched.

V. Conclusion

The algorithm that worked the best for the option pricing problem is the SVM algorithm. It produced the best error analysis results even though it takes a bit longer to train than standard MLP NN's and Bayesian MLP NN's with ARD. What can be attempted in the future is to use some optimisation approach (such as Particle Swarm Optimisation or Genetic Algorithms) to obtain the optimum number of weights and values for other parameters so that the best Bayesian NN can be found. This may prove to be very computationally intensive and may take a very long time especially with the HMC approach with Bayesian NN's. Bayesian techniques can be very powerful and should be experimented with further so that the best parameters for them can be found but at first hand it has been found that the best performing NN is the SVM. The HMC Bayesian approach provides the best confidence levels and maybe a combination of these confidence levels with SVM can be attempted in some manner.

References

[1] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition.* London, Great Britain: Springer-Verlag, 2003, pp. 325-365.

[2] J. C. Hull, *Options, Futures and Other Derivatives, 5th Edition.* Upper Saddle River, New Jersey, U.S.A.: Prentice Hall, 2003, pp. 6-10.

[3] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal Political Economy*, vol. 81, pp. 637–659, 1973.

[4] J. C. Hull, *Options, Futures and Other Derivatives, 5th Edition.* Upper Saddle River, New Jersey, U.S.A.: Prentice Hall, 2003, pp. 234-257.

[5] R. A. Jarrow, and S. M. Turnbull, *Derivative Securities, 2nd Edition.* U.S.A.: South-Western College Publishing, 2000, pp. 15-20.

[6] R. A. Jarrow, and S. M. Turnbull, *Derivative Securities, 2nd Edition.* U.S.A.: South-Western College Publishing, 2000, pp. 175-202.

[7] M. M. Pires and T. Marwala, "American Option Pricing Using Multi-Layer Perceptron and Support Vector Machine", in *Proc. IEEE Conference on Systems, Man and Cybernetics*, The Hague, October 10-13 2004, pp. 1279-1285.

[8] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition.* London, Great Britain: Springer-Verlag, 2003, pp. 156-157.

[9] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition.* London, Great Britain: Springer-Verlag, 2003, pp. 300-307.

[10] T. H. Mirer, *Economic Statistics and Econometric, Third Edition.* U.S.A: Prentice Hall, Inc., 1995, pp. 209-218.

Appendix C:
Software Code, Toolboxes, Data and Output Files

Abstract

This document deals with the software written for the implementation of the various computational intelligence methods used throughout the research period undertaken. It details the steps taken as well as the format used for the output files for each kind of techniques used. What was also done was to write a piece of code to organise the data that was needed and remove what was not needed from the raw data files.

I. SOFTWARE CODE, TOOLBOXES AND DATA

A. Arranging the Raw Data

The data needed was American option data and it was decided to pick a particular underlying asset and then use the options on this asset as the data to be used for training and testing of the different methods. There was data available for all options for the period January 2001 to December 2003. The raw data used was obtained from the South African Futures Exchange (SAFEX) website [1] and it was decided to use the All Share Index options as the data to be used. It was then decided to use the call option data only. It was felt that if call options could be priced then the techniques used could be used in a similar way to price put options.

Neural Networks (NN's) and Support Vector Machines (SVM's) predict outputs based on inputs and thus these inputs need to be chosen according to what would affect the option price [2]. What was then needed to be decided was what to use as inputs and outputs to each of the techniques. From the Black-Scholes equations it can be seen that there are five factors that affect European options [3]. It was then decided that each of these factors would also influence American option prices. Spot price was very close, usually, to the strike price and thus it wouldn't aid the NN's or SVM's trained. Thus spot price was excluded as an input to each of the techniques used. Another factor excluded was the risk-free rate of interest. This interest rate is very constant over a certain period (especially over the last 3 years) and so if it was used as an input the NN's or SVM's would not be able to see its significance (as it wouldn't vary for any of the different option data) and thus if it was included, it would not improve the performance of either of the techniques. The other three factors were decided to be used as inputs, namely the asset volatility, the strike price and the time to maturity (in days).

The output of the network then needed to be decided upon. It can be seen from the data that there is high and low prices. This is due to the fact that there is no standard pricing method for American options and so their prices are a factor of certain market forces [3]. Several brokers use different methods for pricing these options resulting in potential owners of options being able to gain different prices for American options from different brokers. What was decided was to use the average of the high and low prices as the actual option price and this was used as the output of the different techniques used.

The raw data downloaded form the SAFEX website was downloadable in .CSV format to be used with Microsoft® Excel. However it was necessary to transform these files to .XLS files so that the files could be exported to MATLAB®. There are files for every trading day between January 2001 and December 2003. There were no headings for the data entries in the files but the headings could be obtained off the SAFEX website. The Headings can be seen in the correct order in Table 15.

Table 15: Headings for SAFEX Option Data files

Contract
Expiry Date
Strike Price
Marked-to-Market Price
Volatility
Spot Price
High Price
Low Price
Last Traded Price
Volume Traded
Open Interest (No. of Contracts still in Existence)

A program was written in Microsoft® Visual Studio .NET (using Visual C# .NET) so that the relevant information could be extracted from the downloaded files. There are various steps in getting the data into the format required. The screen for the program written can be seen in Figure 20.

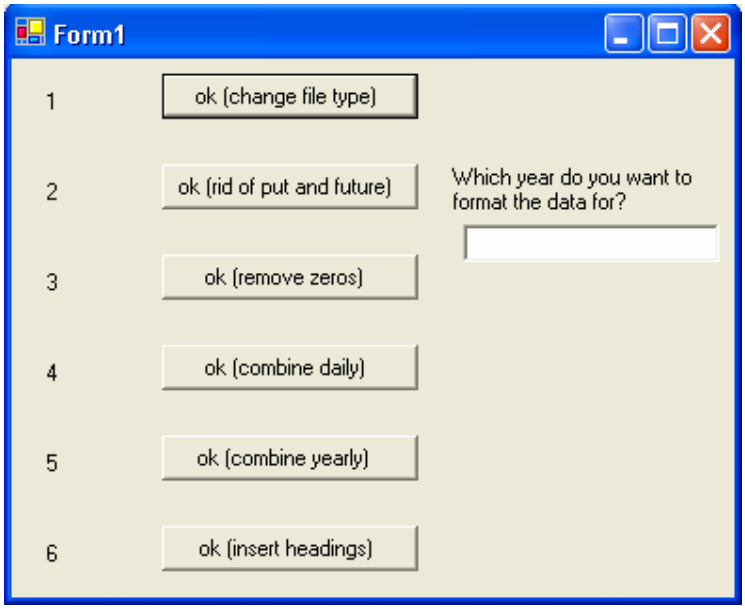

Figure 20: Program Written to Sort the Raw Data

Firstly it must be noted that the data files must be stored in the correct directories for the program to work. On the attached CD, there are three data folders called: "2001_data", "2002_data" and "2003_data". These files contain the daily data for the respective years. All three folders must be copied to the machine's "C:\\" directory for the program to work. As can be seen from Figure 20, there are six steps that must be followed to get the data into the right format so that it can be viewed and then edited in Microsoft® Excel. Step one involves changing each file to a .XLS file (the folders on the CD already contain .XLS files and so this step is not necessary but it can still be done). The user (of the program) must first enter "2001" in the textbox provided and then run through steps 1-4. The user then does the same for "2002" and "2003". The user then doesn't need anything entered in the textbox and he/she then runs steps 5 and 6. The program will create a directory under "C:\\data" and in this directory the file of interest is "data.xls". When this file is opened it will not be in a useful format. The user then selects the file and Microsoft® Excel will open. The "data" tab in Microsoft® Excel must then be clicked and then the "Text to Columns…" option selected. The user must then follow the steps and the data will now be in the columns that he/she needs them in (note that the last column will contain the date at which the data was quoted and so time to maturity can be found by subtracting this date form the expiry date).

This was the procedure followed. The columns not needed were then removed and columns for time to maturity and the average of the high and low prices were inserted (the average of the high and low prices was done by a simple formula in Microsoft® Excel). Time to maturity was found by converting the two date columns (expiry date and the date the data was quoted) into numbers (this can be done in Microsoft® Excel by changing the cell properties) in Microsoft® Excel and then creating a new column and subtracting these two date numbers. The data could then be imported into MATLAB® by selecting the "File" menu and then its "Import Data…" feature. This was done and the data in MATLAB® format is also stored on the CD as "proj1_data.mat". This MATLAB® data file must be copied into MATLAB®'s "work" directory found where MATLAB® is installed for any of the training algorithms programmed to work (the training algorithms are explained in Sections I.B, I.C, I.D and I.E).

The program of Figure 20 is stored on the CD as well. The source code is stored as the "change_file_type" directory. In the directory the project file is included so that the code can be edited, if need be, by future users. If this is not desired then the executable file is also included on the CD as "change_file_type.exe" and can be used in the procedure explained above. Once the data was placed in the format desired then the various techniques could be implemented in MATLAB® using the different toolboxes and functions. There was code written for the four different techniques used, namely, the Multi-Layer Perceptron, the Support Vector Machine and Bayesian techniques applied to both of these and the code written was done in MATLAB®.

B. The Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a Neural Network (NN) architecture. There is a toolbox available by Ian Nabney [2] and the toolbox is for use with MATLAB® and is called the NETLAB toolbox. The NETLAB toolbox is included on the attached CD and is named as the "NETLAB" folder. It must be copied to MATLAB®'s "toolbox" directory and then it must be added to the path by selecting the "File" menu in MATLAB® and then the "Set Path…" feature and selecting the "NETLAB" folder copied. The MLP architecture in this toolbox is a two-layer architecture. Two layers with sigmoidal activation functions in the hidden layer and linear activation functions in the output layer have been found to be universal approximators [4] provided that there are a sufficient amount of neurons in the hidden layer [5]. This meant that the toolbox could be used and more hidden

layers would not improve the predictions. That is why it was decided to use the NETLAB and in addition a sigmoidal function is used in the hidden layer (a hyperbolic tangent function) and a linear function is used in the output layer [1].

In the NETLAB toolbox there are examples that show the training and implementing of the architecture using the various functions. One of these examples was edited and used for the American option pricing problem.

The data arranged produced 3051 points of data and various amounts of training points could be used. This was done by introducing a "train_up" and "train_low" variable which shows the section of data that is to be used as the training data (so if the first 50 data points are to be used as training data then "train_low" is set to 1 and "train_up" is set to 50). The same is done for the test data and the same test data was tried to be used throughout all the techniques used so that relevant comparisons between the different techniques could be made.

There are also various normalisation techniques that can be used for normalising the data. Several were tried and it was decided to use the following normalisation technique:

$$data = \frac{data - \mu}{\sigma} \tag{1}$$

where *data* is the data point to be normalised, μ is the average of the data set being normalised (and so an average is needed for every input and each output) and σ is the standard deviation of the data set being normalised (and so the standard deviation for each input and each output is required). This normalising procedure was thus the procedure used for all the techniques attempted including Support Vector Machines and Bayesian techniques applied to NN's and Support Vector Machines.

The file to be run must be copied to MATLAB®'s "work directory. The file can be found on the CD in the MLP directory called "nettrainmlp.m". This can be run in MATLAB® at the prompt by just typing "nettrainmlp" once the file has copied to the "work" directory.

C. Support Vector Machines

To implement Support Vector Machines (SVM's) a toolbox was also obtained. It was created at the University of Southampton in the U.K [6]. The SVM toolbox is included on the attached CD and is named as the "SVM_toolbox" folder. It must be copied to MATLAB®'s "toolbox" directory and then it must be added to the path by selecting the "File" menu in MATLAB® and then the "Set Path…" feature and selecting the "SVM_toolbox" folder copied. It is a toolbox to be used with MATLAB®. There are several functions that are contained in the toolbox for support vector regression and classification. Support vector regression was used with the same normalisation algorithm as that for MLP's (see Section I.B). There were also variables created for "train_up" and "train_low" so that a particular number of points of data can be used for the training. It must be noted that there are two variables that needed to be created as global variables in MATLAB® ("p1"and "p2") and these are two of the variables used for the training of the SVM's. The file created in MATLAB® is called "trainSVM.m" and is found in the "SVM" directory on the attached CD. This file must be copied to MATLAB®'s "work" directory so that it can be run from the MATLAB® prompt.

D. Bayesian Multi-Layer Perceptrons

The NETLAB toolbox also contained examples for all Bayesian techniques. There were three techniques used with MLP's, namely the Automatic Relevance Determination (ARD) approach (used with Gaussian sampling), the Hybrid Monte Carlo (HMC) sampling method and the Metropolis-Hastings sampling method. Only the Metropolis-Hastings Algorithm is presented in the main body of this dissertation but the other two approaches are presented in a paper for a conference [6]. For Bayesian inference, there is an error function that must be created. The error function for NN's is the error found by the NN trained and it can be found from:

$$E = \sum_{k=1}^{1} \sum_{g=1}^{N} (y_{kg} - y_{k,t\arg et})^2 \tag{2}$$

where E is the error to be minimized and N is the number of sets of training data given. The sets of outputs y_{kg} are the outputs predicted by the network given the training inputs and $y_{k,target}$ are the training outputs. This error can be found by using the "neterr" function provided by the NETLAB toolbox.

For the ARD approach, an example in the NETLAB toolbox was edited and applied to the option pricing problem. The same normalisation technique was used as that for the MLP and SVM approaches. Upper and lower bounds are also found for this approach (as this is the usefulness of the Bayesian framework [7]) and this is done using the NETLAB toolbox and its "netevfwd" function. The file written to implement Bayesian MLP's using the ARD approach is on the CD in the "Bayesian MLP" folder. The name of the file is "bayes_ard.m" and must be copied to MATLAB®'s "work" directory and it can then be

run from the MATLAB® prompt. For this approach there is no "train_up" and "train_low" variables and what is rather used is a "train_amount" variable and so the first number of these points (set by the "train_amount" variable") is used for testing and the subsequent number of points (the number assigned to "test_amount") is used in the testing of the data. This is also done for the HMC approach but for every other approach there is the "train_up" and "train_low" variables used to select the training data (there is also "test_up" and "test_low" to select the test data).

For the HMC sampling method, an example in MATLAB® was also edited and then implemented for the option pricing problem. The normalisation function was the same as that used for all the other approaches. The bounds had to be found intuitively as there was no function in the NETLAB toolbox to do this. With Bayesian techniques what happens is that there are several MLP's (there are as many MLP's as there are number of samples) and therefore each output will have the sample number of predictions. The average of each output for all the networks is the prediction provided. The standard deviation could then be found for each output prediction and the bounds are found by adding the deviation to the prediction (for the upper bound) and subtracting the deviation from the prediction (for the lower bound). With HMC there is also the gradient of the error function required but there is a function in the NETLAB toolbox for this as well and it is called "netgrad". The training was done and the MATLAB® file for this is "bayes_hmc.m". This file can be found on the CD in the "Bayesian MLP" folder and can be run in the MATLAB® prompt by copying the m-file to MATLAB®'s "work" directory.

For the Metropolis-Hastings sampling method, the same file as for the HMC approach can be used. The only difference is that instead of using the 'hmc" function form the toolbox, the "metrop" function must be used and this resulted in only one line of code needing to be changed. In this "metrop" function no gradient function is needed and so instead of passing "netgrad" to the function, this can be replaced with "[ ]". The file of interest for this approach is "bayes_met.m" and can be found on the CD in the "Bayesian MLP" folder. It must also be copied to MATLAB®'s "work" directory and it can then be run from the prompt in MATLAB®.

E. Bayesian Support Vector Machines

To implement Bayesian SVM's was more difficult than with MLP's. The SVM toolbox was separate from the NETLAB toolbox and so some code had to be written. With SVM's the gradient of the error function cannot be found explicitly because the loss function used is the ε-insensitive loss function which is not differentiable [8]. Therefore the only Bayesian technique used was the Metropolis-Hastings sampling method and what needed to be coded was the error function. Two files were written namely "SVMerr.m" and "macherr.m". The "metrop" function form the NETLAB toolbox calls the "macherr" function written and the "macherr" function calls the "SVMerr" function written. The "SVMerr" function computes the error as in equation (2) but for SVM's. These two files written are included on the CD in the "Bayesian SVM" folder along with the function used to train the Bayesian SVM's called "bayes_metSVM.m". All three files must be copied to MATLAB®'s "work" directory. The "bayes_metSVM.m" file is written as a function and can be run from the MATLAB® prompt as follows:

```
[mean_err,max_err,min_err,count10,count5,time,ave_dev,upperbound,lowerbound,test_outputs,test_outputs_actual] = bayes_metSVM(step,samples,rejected)
```

The user can then select the step size, the number of samples and the number of rejected samples at the start of the chain.

All the files that were run created workspaces and this meant that the outputs of the testing could be saved in MATLAB®. The format for these files is now explained so that future researchers can use the results found for future research.

II. TEST OUTPUT FILES

A. The Multi-Layer Perceptron

The results of the MLP's trained can be seen in Table 1 and the first entry in the table has an output file called "mlpex1.mat" found on the CD in the MLP folder. This workspace can be opened ion MATLAB®. To plot the outputs against the actual outputs the variables of interest are "test_outputs" and "test_outputs_actual" respectively.

B. Support Vector Machines

The results of SVM's trained can be seen in Table 2 and the first entry in the table has a workspace associated with it called "SVMex1.mat" and can be opened in MATLAB®. All the entries have a workspace associated with them and can be found on the CD in the "SVM" directory. Graphs of actual and predicted outputs can also be done in MATLAB® and the variables to use in each workspace are "Y_test" and "Y_test_actual" for predicted and actual outputs respectively.

C. Bayesian Multi-Layer Perceptrons

The results for Bayesian MLP's with the ARD approach can be seen in Table 13. Each entry has a workspace associated with it and stored on the CD in the "Bayesian MLP" folder. For example, the first entry in the table is labelled as "ardex1.mat" in the directory on the CD. To plot the predicted outputs, actual outputs, upper bounds and lower bounds, the variables of interest are

"test_outputs", "test_outputs_actual", "upperbound" and "lowerbound" respectively. Each workspace can be opened in MATLAB®. The same variables are of interest for the HMC and Metropolis-Hastings sampling methods. There is a workspace saved for each entry in Table 14 where the first is labelled "hmcex1.mat" in the "Bayesian MLP" directory on the CD. There is also a workspace saved for each entry in Table 3 where the first is labelled "metex1.mat" in the "Bayesian MLP" directory on the CD.

D. Bayesian Support Vector Machines

The results of the Bayesian SVM's can be seen in Table 4. Every entry in the table has a workspace associated with it and can be accessed on the CD. All the workspaces are in the "Bayesian SVM" directory. For example, the first entry in the table has a workspace associated with it called "bayes_SVM_ex1.mat" stored on the CD and can be opened in MATLAB®. The variables for predicted outputs, actual outputs, upper bounds and lower bounds are "test_outputs", "test_outputs_actual", "upperbound" and "lowerbound" respectively. These can then be plotted once the respective workspace is opened.

III. CONCLUSION

All the software written and how it can be used has been shown in detail. Also the findings for each of the methods have been provided and how to replicate the graphs has also been explained in detail. All the software and output files are provided on the attached CD. For more details on each m-file written the comments in the m-files can be consulted. The necessary toolboxes required have also been included on the CD. Included on the CD is a copy of this thesis in both .doc (to be viewed with Microsoft® Word) and .pdf (to be viewed with Adobe Acrobat Reader) formats. These files can be found on the CD in the "dissertation" folder and are saved as "MPires_dissertation.doc" and "MPires_dissertation.pdf" respectively.

REFERENCES

[1] *South African Futures Exchange,* Available: http://www.safex.co.za.

[2] I. T. Nabney, *NETLAB: Algorithms for Pattern Recognition,* Springer-Verlag, London, Great Britain, 2003.

[3] M. M. Pires, Masters Thesis, School of Electrical and Information Engineering, University of the Witwatersrand, 2005, Appendix A: Hedging with Options and Other Contracts.

[4] K. M. Hornick, M. Stinchcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, vol.2, no. 5, 1989, pp. 359-366.

[5] M. M. Polycarpou, "On-Line Approximations for Nonlinear System Identification: A Unified Approach" In C. Leondes, editor, *Neural Network Systems, Techniques and Applications,* vol. 7: Control and Dynamic Systems, California, 1998, Academic Press.

[6] S. R. Gunn, *Support Vector Machines for Classification and Regression.* Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, 1998.

[7] M. M. Pires, Masters Thesis, School of Electrical and Information Engineering, University of the Witwatersrand, 2005, Appendix B: Paper for *Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa.*

[8] W. Chu, S. S. Keerthi and C. J. Ong, "Bayesian Support Vector Regression Using a Unified Loss Function", *IEEE Transactions on Neural Networks*, vol. 15, no. 1, 2004, pp. 29-44.