A Neural Network Versus Black-Scholes: A Comparison of Pricing and Hedging Performances

Henrik Amilon*
Department of Economics
Lund University
P.O. Box 7082
SE-220 07 Lund
Sweden
Phone: +46(0)462227909
Fax: +46(0)462224118
E-mail: henrik.amilon@nek.lu.se

Abstract

The Black-Scholes formula is a well-known model for pricing and hedging derivative securities. It relies, however, on several highly questionable assumptions. This paper examines whether a neural network (MLP) can be used to find a call option pricing formula better corresponding to market prices and the properties of the underlying asset than the Black-Scholes formula. The neural network method is applied to the out-of-sample pricing and delta-hedging of daily Swedish stock index call options from 1997-1999. The relevance of a hedge-analysis is stressed further in this paper. As benchmarks, the Black-Scholes model with historical and implicit volatility estimates is used. Comparisons reveal that the neural network models outperform the benchmarks both in pricing and hedging performances. A moving block bootstrap procedure is used to test the statistical significance of the results. Although the neural networks are superiour, the results are often insignificant at the 5% level.

Keywords: options, hedging, bootstrap
JEL code: C45, C51, C52, G13

*The author is grateful to Ola Hössjer for most valuable help on bootstrapping and statistical inference.

1 Introduction

In the early seventies, there was a major breakthrough in financial asset modeling and option price theory, when Black and Scholes (1973) presented their famous formula for pricing derivatives. By modeling the price of the underlying asset, I_t, as a continuous-time diffusion

$$dI_t = \mu I_t dt + \sigma I_t dW_t, \tag{1}$$

with constant parameters, μ and σ, they show that a continuous rebalancing of a certain combination of the option and the underlying asset entails no risk. Hence, in the absence of arbitrage, this combination must yield the risk-free interest rate, which is also assumed to be constant. In this framework, a closed form solution of the option price is possible. Despite the large number of questionable assumptions, the most crucial of which are the log-normality of prices, implied by (1), and continuous trading, the Black-Scholes prices are quite close to those observed in the market.

A great deal of effort has been made to relax some of the assumptions in the original framework, of which the stochastic volatility approach is most notable. By also letting the volatility parameter (or some transform of σ) in (1) be described by a stochastic diffusion, prices are obtained with slightly better resemblance to market prices, see e.g. Scott (1987). Still, the price formulas hinge upon the continuous-time approximation, the normality of the disturbances, and the parametrization of the diffusion processes.

Hutchinson et al. (1994) take a different approach. Instead of specifying the stochastic properties of the underlying diffusion, they use an artificial neural network in a nonlinear regression of some input variables on the observed market prices. The use of a neural network makes this approach nonparametric. They do not specify how inputs affect option prices, but rather let the data determine the relationships. As inputs, they choose the price of the underlying asset normalized with the strike price of the option, and the time-to-maturity. The neural network model is tested against a benchmark, that is, the Black-Scholes formula with historical volatility estimates. The results reveal that the neural network model gives a better out-of-sample fit to observed market prices, and a lower hedge error than does the benchmark model.

The practical relevance of a hedge analysis is mentioned in Hutchinson et al. (1994) and its importance is further stressed here. There are two ways of determining the parameters in the parametric diffusion models, one of which is to calibrate the model prices to option market data, the other to estimate the parameters from time series of the underlying asset. The parameters, and hence the model option prices, obtained in these two ways may not coincide, and this is often the case[1]. An obvious explanation would be that the model is misspecified, a conclusion which might be too hasty, however. How do we really know if the market option prices are correct? Believing the observed option prices to be totally wrong may seem naive, but believing all option prices to always be correct is equally naive. If they were, there would hardly be any need for any "improved" option pricing model whatsoever.

It is not surprising that calibrated model prices are closer to observed market prices, since these are used in the estimation, but they say little about the connection to the actual dynamics of the underlying asset. Regardless of the pricing model, it is, after all, the value of the underlying asset at expiration that determines the option pay-off. The market prices act as a prediction of future price variability in the underlying asset. If, on average, these predictions are not realized, then the market prices cannot be correct, and measures-of-fit based on price accuracy are of little use. A hedge analysis will, however, reveal if the market predictions are correct, that is, if it is better to fit a model to option data, or to focus on the stochastic time series properties of the underlying asset.

A neural network, on the other hand, is estimated on option market data. The nonlinear regression approach does not assume that all prices are always correct, but rather that a "true" option pricing formula is embedded in the noisy market prices. The problem is rather the specification of the relevant input variables. Do we only want to include variables from the time series of the underlying asset, or do we want to use information from the market option data, very much like the calibration approach?

This paper proposes an extension of the nonparametric neural network approach in Hutchinson et al. (1994). I estimate, or train, neural networks on daily Swedish index call option data with more input variables, in order to better capture the relationships between the derivative and the underlying asset. I also model the spread, that is I use the neural network to find the mapping from the inputs to the bid and the ask prices of the options. The neural network pricing formulas are compared to two versions of a parametric benchmark model, the Black-Scholes formula with historical volatility estimates and implicit volatility estimates, respectively. These benchmarks are chosen for three reasons. First, they are often used as benchmarks in comparisons with more elaborate parametric models. Second, they are frequently used by practitioners, and third, they are "not bad". In spite of the questionable assumptions, they are not always outperformed by more general diffusion-based models, at least in price comparisons, see e.g. Chesney and Scott (1989). For the reasons above, all models are compared according to both their pricing and hedging performances. Hedge analyses are also made in Hutchinson et al.

[1]The two approaches can also be combined. Some parameters are estimated from time series data, others are calibrated to option data.

(1994) and in Chesney and Scott (1989), but the hedging scheme used here differs from theirs in some important ways. Furthermore, by using the bootstrap technique in Efron (1979), I try to do statistical inference on my results, an issue overlooked in the earlier literature. Much of the methodology in this paper is not restricted to neural network valuation. The same steps can be taken when comparing other option pricing formulas, parametric as well as non-parametric ones.

In section 2, I give a brief introduction to artificial neural networks in general and my choice of neural network model, the multilayer perceptron, in particular. Section 3 describes the data, and section 4 more closely presents the neural network regression setup. The empirical results are given in section 5. Summary and conclusions are found in section 6.

2 Artificial Neural Networks

The mammal brain is an astounding creation, in many cases superior to the most powerful computer. The brain is, for example, capable of processing large amounts of information in order to solve different problems, even though the information is incomplete and "noisy". It also has the desirable ability of learning from experience and adapting to changes in environmental factors. This is quite a contrast to conventional computers which, at all times, must receive detailed instructions. These robust features are possible, due to the parallel structure of the brain, where billions of neurons work simultaneously. Conventional computers, on the other hand, work sequentially and contain only a few processors.

The philosophy of the Artificial Neural Network (ANN) approach is to design an architecture and a learning algorithm that show more resemblance with the human brain. It abstracts some key ingredients from biology and, out of these, constructs simple mathematical models that exhibit many of the above-mentioned appealing features. The link to biology should not be exaggerated, however. The ANN framework has turned out to have some very powerful theoretical properties, which also makes it useful in more traditional financial areas, such as function approximation and time series prediction, see e.g. Cybenko (1989) and Weigend et al. (1990).

Depending on the application, different types of ANN models are used, such as the Boltzmann machine, the Hopfield model, and the Kohonen networks, see Hertz et al. (1991) and Peterson and Rögnvaldsson (1991) for further references. The most popular model by far, particularly in nonlinear regressions, is the Multi Layer Perceptron (MLP).

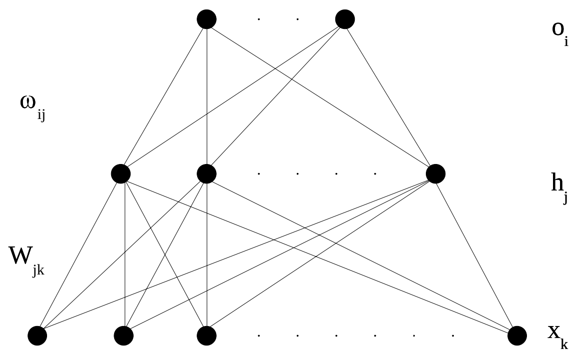

Figure 1: A single-hidden layer MLP architecture. Notations are explained in the text.

2.1 The Multi Layer Perceptron

Neurons constitute the basic building blocks in all ANN's. In the MLP, they are organized in different layers. The inputs (x_k) are fed to an input layer, the outputs of the network (o_i) are given in the output layer, and in between, there is an arbitrary number of hidden layers. The hidden neurons build up an internal representation of the data. The architecture is shown in Figure 1 for an MLP with one hidden layer.

The functionality of the individual neurons is simple. Each neuron sums up the signals leading to it, adds a bias term, and makes a nonlinear transformation. The transfer function is typically a smooth monotonically increasing function, a sigmoid, such as the hyperbolic tangens or the logistic function. It can also be a linear function, but for the network to be able to map nonlinear functions, some hidden layers must have nonlinear transfer functions. The transformed signal of the neuron is passed on to neurons in subsequent layers, and the procedure is then repeated. The connections between the neurons are represented by weights. When the MLP is presented to input vectors, these are fed forward through the network via the neurons. The network outputs are compared to a known target vector and an error function is computed. The error is propagated backwards through the network and the weights are adjusted to minimize the error function. The same procedure is repeated over and over until the network outputs match the targets with an acceptable accuracy. The MLP has now been trained, and is ready to use for data it has never "seen". The training algorithm in adjusting the weights is called back propagation and is originally derived (Rumelhart et al., 1986) by using gradient descent.

2.2 The Back Propagation Algorithm

For simplicity, the derivation is for an MLP with a single hidden layer – the generalization to several hidden layers is straightforward. The network architecture and some notation are shown in Figure 1. Inputs are denoted x_k $(k = 1, ..., N_{input})$, outputs from the hidden neurons are denoted h_j $(j = 1, ..., N_{hidden})$, and outputs from the neurons in the output layer, the network output, are denoted o_i $(i = 1, ..., N_{output})$. Inputs are organized in vectors, $\mathbf{x}_p = (x_{p1}, ..., x_{pN_{input}})$ and for each input vector, there is a known target vector, $\mathbf{t}_p = (t_{p1}, ..., t_{pN_{output}})$, where p denotes which of the input vectors is currently presented to the network. For example, in a time series analysis with 10 input vectors, $\mathbf{t}_p$ could be equal to x_t and $\mathbf{x}_p = (x_{t-1}, ..., x_{t-10})$. The next pair of input-target vectors would then be $\mathbf{t}_{p+1} = x_{t+1}$, and $\mathbf{x}_{p+1} = (x_t, ..., x_{t-9})$.

The training of the network can be done either in batch-mode, where the weights and biases are adjusted after all input vectors have been presented to the network, or on-line, where the adjustments take place after each input vector. In order to suppress the index p, the latter alternative is assumed.

The weight from x_k to the jth hidden neuron is denoted W_{jk} and the weight from the jth hidden neuron to the ith output is called ω_{ij}. At each layer, I use the notation a_i and a_j for the weighted input signals to the neurons:

$$\begin{aligned} a_j &= \sum_k W_{jk} x_k + \theta_j \\ a_i &= \sum_j \omega_{ij} h_j + \theta_i. \end{aligned}$$

The thresholds are also parameters and can be absorbed into the weight-formalism by letting $\theta_j = W_{j0} x_0$ and $\theta_i = \omega_{i0} h_0$, where x_0 and h_0 are neurons clamped to $+1$. With the same transfer function, $g(\cdot)$, for all lags, it follows that the output from each layer is

$$\begin{aligned} h_j &= g(a_j) = g(\sum_k W_{jk} x_k) \\ o_i &= g(\sum_j \omega_{ij} h_j) = g(\sum_j \omega_{ij} g(\sum_k W_{jk} x_k)). \end{aligned} \tag{2}$$

As an error function, the sum of squared errors is often used:

$$E = \frac{1}{2} \sum_i (t_i - o_i)^2 = \frac{1}{2} \sum_i (t_i - g(\sum_j \omega_{ij} g(\sum_k W_{jk} x_k)))^2. \tag{3}$$

Training the MLP is equivalent to adjusting the weights so as to minimize E, and can be done with gradient descent. For weights between the hidden and the output layers, we have

$$\Delta \omega_{ij} = -\eta \frac{\partial E}{\partial \omega_{ij}} = -\eta \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\omega_{ij}} = \eta (t_i - o_i) g'(a_i) h_j = \eta \delta_i h_j,$$

where η is a learning parameter and $\delta_i = (t_i - o_i)g'(a_i)$. In order to compute the updating rules for the hidden layer, the chain rule gives

$$\Delta W_{jk} = -\eta \frac{\partial E}{\partial W_{jk}} = -\eta \sum_i \frac{\partial E}{\partial o_i} \frac{\partial o_i}{h_j} \frac{\partial h_j}{\partial W_{jk}} = \eta \sum_i (t_i - o_i) g'(a_i) \omega_{ij} g'(a_j) x_k =$$
$$= \eta g'(a_j) x_k \sum_i \delta_i \omega_{ij}. \quad (4)$$

The weights are then updated, using vector notation, according to $\boldsymbol{\omega}_{p+1} = \boldsymbol{\omega}_p + \Delta\boldsymbol{\omega}$, until convergence or the desired performance level is achieved. The updating is proportional to the δ's in the subsequent layer. Starting from the output neurons, the error propagates backwards in the network via the δ's; hence the name of the algorithm.

Many applications do amazingly well with gradient descent updating, but for some problems, this optimization method is too slow. Faster convergence is sometimes possible if gradient descent is combined with a line search. This means that when moving along the gradient, $-\eta \bigtriangledown E$, the step length η is not fixed, but instead chosen so as to minimize E. After one step, the gradient is recomputed and the procedure repeated. Further improvements can be made if, at each step, the new search direction, $\mathbf{d}_{p+1}$, is chosen as a combination of the gradient and the earlier direction:

$$\begin{aligned} \boldsymbol{\omega}_{p+1} &= \boldsymbol{\omega}_p + \eta \mathbf{d}_{p+1} \\ \mathbf{d}_{p+1} &= -\bigtriangledown E(\boldsymbol{\omega}_p) + \beta \mathbf{d}_p. \end{aligned}$$

If β is chosen to spoil as little of the previous achievements as possible, that is, so that $\mathbf{d}_p \cdot \bigtriangledown E(\boldsymbol{\omega}_{p+1}) = 0$, we get an optimization method with often excellent properties regarding convergence and speed, see Hertz et al. (1991) and Fletcher (1990). Such optimization techniques are called conjugate gradient methods.

2.3 Network Architecture

The optimization methods above find the optimal, at least the locally optimal, weights for a given MLP architecture, but how to find the optimal architecture, that is the optimal number of hidden layers and neurons? The issue is very much problem dependent, but it is a general statement that no more than two hidden layers are needed to approximate an arbitrary function. If the function is continuous, then only one hidden layer of sigmoid neurons is needed, see Cybenko (1989) and Hornik (1989). This is not necessarily the most efficient number of layers. The use of more hidden layers may result in a smaller number of neurons in total, and thereby faster training.

Regarding the number of hidden neurons, one must often, to some extent, rely on experimentation. The MLP is a universal approximator (Hornik, 1989). The implication is that given enough hidden neurons, any function can be modeled arbitrarily well. This is a very strong result, but in the presence of noise and outliers, too many neurons are likely to result in overfitting, and we end up in a situation characterized by a good fit of input vectors, but with large fluctuations in between. The network has been trained to copy the input data instead of generalizing, so when the network is presented to unseen data, its performance is low. A popular approach to avoid overfitting is to split the data into three parts, see e.g. Peterson and Rögnvaldsson (1991). The training set is used for estimating the weights, the validation set is used for model selection, and the test set is used for out-of-sample evaluation. It is important to clarify that the performance of the test set must not influence the choice of architecture.

A normalization of the input data to the same order of magnitude is often an effective way of decreasing the complexity of the network. The reason is seen in Equation (4), where weight adjustments are proportional to inputs, x_k. If the learning parameter is the same for all network neurons (although it may change from time to time by the line search), its size will be dictated by the largest input. To keep stability, η must be low and the convergence for the other weights will be more time consuming.

3 The Data

The data used in the empirical analysis are daily closing quotes of the Swedish OMX index and daily closing bid and ask prices of European OMX index call options, during the time periods June 97-March 98, and June 98-March 99. The OMX index is a value weighted combination of the 30 most traded securities at the Stockholm Stock Exchange. The composition of the index is updated every sixth month, on December 30 and June 30. At any day, OMX index options have at least three unique expiration dates; the current month, the next month, and two months ahead. On the fourth Friday each month, some contracts expire, and new ones, with a time-to-maturity of three months but different strike prices, are introduced. The new contracts have at least three, but often five, strike prices centered around the current OMX index value. If the index moves outside the current strike price range, another strike price is added for all expiration dates to bracket that index value. Thus, the strike prices reflect the path of the OMX index during the time-to-maturity.

The reason for the ad hoc split into two periods is that Swedish stocks pay dividend in two months only, April and May. Since the options are of European style, options traded in April or May or expiring in these months are excluded, to avoid adjustments of the OMX index for dividend pay-offs[2]. After these exclusions, 514 different option contracts are traded, yielding a total of 9416 pairs of bid and ask prices.

I also use an approximation of a risk-free interest rate, which is chosen as the continuously compounded return on a 90 day treasury bill (*Statsskuldväxel*)[3].

4 The Neural Network Model

The goal is to model a mapping of some input variables onto the observed option prices. The use of a neural network makes this approach nonparametric. I do not specify how inputs are supposed to affect option prices, but rather let the relationships be determined by the data.

The functional I try to model is specified to be of the form

$$(c_t^b, c_t^a) = \mathbf{f}(I_t, K, T-t, T^{cal}-t, r_t, \mathbf{a}),$$

where c_t^b is the bid price of the call option at time t, c_t^a is the ask price, I_t is the OMX index value, K is the strike price, $T-t$ is the time-to-maturity in trading days, $T^{cal}-t$ is the time-to-maturity in calendar days, and r_t is the risk-free interest rate. The vector $\mathbf{a}$ contains additional inputs. I defer the specification of $\mathbf{a}$ until the next section, since $\mathbf{a}$ differs depending on which of the benchmark models is used. Here, I only consider issues similar to both network models.

In order to reduce the number of inputs, I make two simplifications. First, I follow Hutchinson et al. (1994) and assume that $\mathbf{f}$ is homogenous of degree one in I_t and K, that is $\mathbf{f}(I_t, K, ...)/K = \mathbf{f}(I_t/K, 1, ...)$. This is certainly true for the Black-Scholes formula and it is quite a natural assumption. Otherwise, a split of the underlying asset would affect the option price. In a recent paper, Garcia and Gençay (2000) exploit the homogeneity property in terms of functional shape. Instead of using a neural network to map I_t/K and $T-t$ directly onto the option price divided by K, they break down the pricing function into two parts. One part is controlled by I_t/K and the other by a function of $T-t$, similar to the functional shape of the Black-Scholes formula and other models with the same homogeneity property. They model each part with an MLP network with I_t/K and $T-t$ as the only inputs, and show that such a restrictive model, although more time consuming to estimate, has a better pricing accuracy than the unrestrictive model by Hutchinson et al. This idea is not explored further here.

The second simplification concerns the fact that the time-to-maturity of a treasury bill is measured in calendar days. If r_t is only assumed to enter in the pricing function as a discount

[2]The sample largely consists of artificial prices, introduced by the option broker firm (OM) and used for "safety calculations". These are also excluded. The data contains no obvious outliers.

[3]The 90 day treasury bill is preferred to the less liquid 30 and 60 day treasury bills.

factor (as in the Black-Scholes formula), the relevant input argument is $r_t(T^{cal} - t)$. Thus, the specification becomes

$$(c_t^b, c_t^a)/K = \mathbf{f}(I_t/K, T - t, r_t(T^{cal} - t), \mathbf{a}).$$

The neural network model used is an MLP with one hidden layer of neurons with transfer functions $g(\cdot) = \tanh(\cdot)$. Since I model both the bid and the ask prices of the call options, there are two output neurons which are compared to the target vectors $(c_t^b, c_t^a)/K$. I choose the logistic transfer function for the output neurons, that is $g(\cdot) = \frac{1+\tanh(\cdot)}{2}$, so that the output range $\in [0, 1]$. Usually the output neurons are linear in function approximation, but since the targets are positive, using the logistic functions makes more sense.

In the earlier section, I stressed the importance of evaluating the network by the use of a training set, a validation set, and a test set. In addition, there is the (usual) trade-off between long enough estimation periods to ensure a good fit, and the risk of invoking nonstationarities, that is, changes in the relationship between input and output variables. The original periods (periods 1 and 2) are therefore split into subperiods in the following way: I train MLP networks with different numbers of hidden neurons on the first four months of period 1 (Jun. 97-Sep. 97), validate them on the consecutive two months (Oct. 97-Nov. 97), choose the one with the lowest validation error, and evaluate it on the following month (Dec. 97). I then repeat the procedure, but now with the first five months (Jun. 97-Oct. 97) in the training set, the two consecutive months (Nov. 97-Dec. 97) in the validation set, and test the best network the following month (Jan. 98). Since period 1 ends in March 98, this can be done another two times, thereby giving four non-overlapping test sets. Applying exactly the same procedure to period 2, we end up with a total of eight test sets (Dec. 97-Mar. 98 and Dec. 98-Mar. 99) for out-of-sample evaluation.

It is important to note that the networks can be sensitive to extrapolation. If they are presented to an input vector outside the range of the training data, their performance may be low. It is therefore of importance to assure that the validation set is "embedded" in the training set. For example, suppose that the index drops four months and then rises the following two months. The training data, as I have chosen it, would then contain a larger amount of out-of the money options, while the validation set is overrepresented by in-the-money options. In such a case, it is wiser to choose, say, the first, the third, the fourth, and the sixth month as training data, and use the second and the fifth month for validation, a procedure not required for the data used here.

Designing the test data in a similar way would not be correct. Instead, I monitor the largest error in the training and the validation sets. If the network gives a larger out-of-sample prediction error, that prediction is considered unacceptable and is not used.

The MLP networks are trained in batch mode, using a conjugate gradient algorithm for adjusting weights and biases[4]. For each period, 20 networks are trained with three different numbers of hidden neurons. In all cases, the sum of squared errors, Equation (3), is minimized for the validation set. All inputs are normalized by extracting the mean and dividing by the standard deviation of the training set.

5 Empirical Results

In this section, I explore how well the MLP networks do compared to the two benchmark models: the Black-Scholes formula with historically estimated volatility, and with implicitly estimated volatility. The implicit volatility procedure means that the volatility parameter for a certain time-to-maturity is calibrated each day to get a perfect fit for an at-the-money option. The volatility estimate is then used to price other options with the same time-to-maturity.

I evaluate the competitive models by using both a price and a hedge analysis. The very essence of all parametric continuous-time pricing formulas is the ability to replicate an option through a dynamic hedging strategy. To be of any practical relevance, the neural network models must also be able to hedge an option position. Fortunately, Hornik et al. (1990), and Gallant and White (1992), show that MLP networks with a sufficient number of hidden neurons can approximate the derivative of an arbitrary nonlinear function arbitrarily well. Specifically, delta-hedging is achieved by derivating (2) on the first (unnormalized) input variable. A delta-hedge analysis of neural network based option pricing models is also performed in Hutchinson et al. (1994) and in Chesney and Scott (1989). However, the hedging scheme in this paper differs substantially from theirs.

Statistical tests of the results are difficult to design properly, because of the statistical dependences of the option-price paths as noted in Scott (1987) and Hutchinson et al. (1994). Here, I try to design such tests by using the bootstrap technique originating from Efron (1979).

5.1 An MLP Versus Black-Scholes with Historical Volatility

For each of the eight test periods, I train 20 single-layer MLP networks with 10, 12, and 14 hidden neurons, and choose the one with the best validation performance. The additional input is extracted from the underlying OMX index only. I use a total of nine input variables,

$$\mathbf{x}_t = \Big(\left(I_t, I_{t-1}, I_{t-2}, I_{t-3}, I_{t-4} \right) / K, T - t, r_t(T^{cal} - t), \hat{\sigma}_{30}, \hat{\sigma}_{10} \Big) = (x_{t1}, ..., x_{t9}),$$

[4]All computations were run in MATLAB version 5.2.

Table 1: Sensitivity dependences, Equation (5), in the MLP regressions for each testperiod, and for all eight testperiods.

Periods	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
1	1.26	0.06	0.08	0.09	0.07	0.36	0.32	0.07	0.04
2	1.25	0.04	0.08	0.04	0.04	0.53	0.27	0.08	0.02
3	2.08	0.14	0.09	0.05	0.03	0.31	0.53	0.18	0.22
4	2.25	0.06	0.13	0.11	0.08	0.31	0.62	0.07	0.12
5	2.16	0.10	0.11	0.06	0.10	0.69	0.39	0.25	0.03
6	2.29	0.50	0.05	0.12	0.11	0.59	0.53	0.30	0.22
7	2.41	0.08	0.05	0.06	0.24	0.66	0.45	0.85	0.09
8	1.76	0.05	0.01	0.06	0.20	0.66	0.23	0.88	0.02
1-8	1.93	0.13	0.08	0.07	0.11	0.52	0.42	0.34	0.09

which are mapped against the target vectors $\mathbf{t}_t = (c_t^b/K, c_t^a/K)$, yielding the network estimates $\mathbf{o}_t = (\hat{c}_t^b/K, \hat{c}_t^a/K)$. The volatility of the underlying asset might be suspected to be important for option pricing. The purpose of including lagged index values is to allow the neural network to build a volatility structure from the data. Any number of lags can be included, but in order to economize on the input variables, only the four last lags are chosen. Previous information embedded in the time series is included by the standard deviation of the 10 and 30 most recent continuously compounded daily returns of the OMX index, $\hat{\sigma}_{10}$ and $\hat{\sigma}_{30}$. Thus, the networks are presented with two slowly varying volatility trends and may respond more quickly to departures from these trends through the lagged index values[5].

The dependence of the output upon the input variables is analyzed by inspecting derivatives after a completed regression. For the first output, this is done by calculating

$$S_k(t) = \frac{\partial \hat{c}_t^b \left(x_{t1}, ..., x_{t9}\right)/K}{\partial x_{tk}}, \qquad k = 1, ..., 9$$

for $t = 1, ..., n$, where n is the number of test vectors, and then compute

$$S_k = \frac{1}{n} \sum_t |S_k(t)| \,. \tag{5}$$

All input variables are normalized to the same order of magnitude, so that the S_k's are comparable in size.

The measure in (5) tells us which variables are most important. Table 1 shows the sensitivity dependences for the eight test periods, and for the average of the test periods. The results are valid for the first output variable, but the dependences are similar for the second output variable.

[5]The parameter estimates used in the Black-Scholes formula are r_t, $T^{cal} - t$, $\hat{\sigma}_{30}$, and $T - t$.

The magnitudes of S_k sometimes differ substantially between the test periods. Averaged over all test periods, there seems to be no doubt of the most explanatory variable being I_t/K followed by $T-t$, $r_t(T^{cal}-t)$ and $\hat{\sigma}_{30}$. The influences of the lagged indexes and the shorter moving average standard deviation are not considered to be very important. This suggests that the link between the volatility of the underlying asset, as modeled here, and the option prices are not as clear as might be suspected.

In the following section, I separate between price and hedge analysis.

5.1.1 Pricing

The results of the out-of-sample price comparisons averaged over all test periods are shown in Table 2. To investigate the performance in different parts of the input space, in-the-money options (ITM) and out-of-the-money (OTM) options are analyzed separately. We see that the mean of the option price errors, $\mathrm{ME}_{\mathrm{BS}} = n^{-1}\sum_{i=1}^{n} c_i - \hat{c}_i^{\mathrm{BS}}$ and $\mathrm{ME}_{\mathrm{MLP}} = n^{-1}\sum_{i=1}^{n} c_i - \hat{c}_i^{\mathrm{MLP}}$, differ between the models[6]. Henceforth, the subscripts are dropped unless considered to be required. The networks slightly underprice the ITM options, while Black-Scholes overprice them more heavily. The OTM options are instead slightly overpriced by the networks, while the results for Black-Scholes are more moot. Comparing all options, the network estimates show very little bias, while the Black-Scholes prices are larger than the market data. The spread is not modeled in the Black-Scholes estimates, but the Black-Scholes prices are closer to the market ask prices.

The mean errors (ME) are, of course, not good measures-of-fit, as the fluctuations around a low mean can be large. Since the network models are estimated by using the sum of squared errors, it is natural to evaluate the models with the Root Mean Squared Error (RMSE)[7]. From Table 2, we see that RMSE is lower for the MLP networks for all types of options. For the MLP networks, it is somewhat lower for the ITM options, while this is not seen for the Black-Scholes model.

Are these results statistically significant? Both Scott (1987) and Hutchinson et al. (1994) note that statistical tests are difficult to formulate because the errors in fitting the option prices are likely to be correlated across options and over time, but they give no guidance to how to construct such tests. Garcia and Gençay (2000) use the Diebold and Mariano (1995) test, which assumes that the sequence of the difference between the predictions of two models is covariance

[6] Due to the monitoring for extrapolation, one option in period 1 and seven options in period 5 are excluded from the MLP sample. In order to facilitate the comparisons, the same options are excluded from the Black-Scholes sample. These exclusions are negligible.

[7] The measures in price units are obtained by using the root.

Table 2: Bid and ask price error comparisons, ME, RMSE and ΔRMSE, for MLP and Black-Scholes with historical volatility. The comparisons are for 2113 out-of-the-money options, 906 in-the-money options, and for all 3019 options. Small numbers in paranthesis are bootstrapped 95% confidence intervals.

Options	Statistics	MLP (bid)	BS (bid)	MLP (ask)	BS (ask)
OTM	ME	0.13	-2.64	0.17	-1.28
	RMSE	2.14	5.47	2.30	4.72
	ΔRMSE	3.33 (1.54,4.92)		2.42 (1.43,3.37)	
ITM	ME	-0.26	-2.34	-0.35	0.29
	RMSE	2.82	5.39	3.00	4.64
	ΔRMSE	2.57 (0.96,3.71)		1.64 (1.05,2.47)	
ALL	ME	0.01	-2.55	0.01	-0.81
	RMSE	2.36	5.45	2.53	4.69
	ΔRMSE	3.08 (1.35,4.51)		2.16 (1.39,2.84)	

stationary. By estimating the autocovariance function, presumable with a suitable lag window, the null hypothesis of no difference in prediction accuracy between two models can be tested. It is important to note, however, that the prediction error series here, and also in Garcia and Gençay, are not pure time series. They consist of observations belonging to the same day as well as different days. Any particular ordering of the errors within each day is not obvious, but a reordering of the errors within each day will certainly change the estimate of the autocovariance function and hence, the Diebold and Mariano test statistic. Besides, differences in RMSE's are not computable with this test.

Instead, I use the overlapping block resampling in Künsch (1989). As shown in Fitzenberger (1998), this method is valid under weaker assumptions than covariance stationarity. The resampling scheme looks as follows: Draw a day with probability $1/D$ from all $D = 156$ days in the test series. Construct a block of model errors of size l, by joining the model errors for that day and the $l - 1$ consecutive days. Repeat the procedure until the bootstrapped series consist of about the same number of blocks as the original series, and compute the statistics of interest, giving one bootstrap estimate. Calculate estimates for 1000 bootstrap replicates and extract the confidence intervals from the resampling distribution[8].

The choice of the block length l is an intricate issue. Clearly, large blocks will more faithfully preserve the dependence structure in the original series. On the other hand, the resampled series must vary enough to provide a good estimate of the distribution of the test quantities, which points towards small l. Hall et al. (1995) give some asymptotic results for guidance. Suppose we

[8]When bootstrapping the differences of two models, the same random draws are applied to each series.

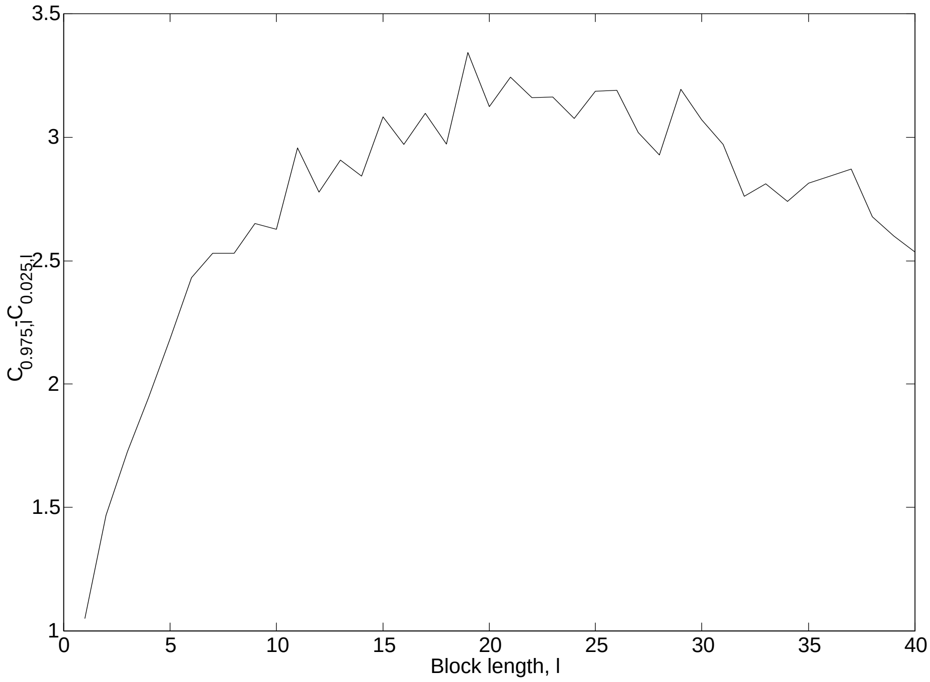

Figure 2: The width of the block bootstrapped confidence intervals for the difference in RMSE for all 3019 bid option price errors, plotted versus the block length l.

want to estimate some statistical quantity κ based on a series of length n. Hall et al. show that under suitable conditions and for large n and l, the mean squared error of the bootstrap estimate $\hat{\kappa}(n, l)$ is minimized for $l \propto n^{\frac{1}{c+2}}$. The positive integer c takes on different values depending on the statistical quantity κ, thereby ensuring that the number of blocks as well as the block length go to infinity as n goes to infinity. The constant of proportionality is not known, but Hall et al. argue that it can be estimated by comparing $\hat{\kappa}(n, l)$ to the bootstrap estimates from a subset of shorter series of length $m < n$, and different block sizes $k < l$. Once the optimal block size $\hat{k}$ for a series of length m is determined, the block size for a series of length n is given by $\hat{l} = \hat{k} \left(\frac{n}{m}\right)^{\frac{1}{c+2}}$.

This procedure has certain drawbacks. First, it is obviously very computer intensive, since it suggests bootstrapping a large number of subseries for each k. Second, the results are valid asymptotically, that is n and m are large, and block sizes k and l are large, particularly large enough to capture all serial dependences in the data. Third, as exemplified in Davison and Hinkley (1997), different choices of m will result in different $\hat{k}$ and hence, different $\hat{l}$. At best, the above procedure will result in a range of $\hat{l}$ from which the bootstrap estimate is extracted, relying to some extent on a subjective choice.

For the above reasons, a more heuristic approach is used in choosing l, when determining the 95% confidence interval for $\Delta\text{RMSE}=\text{RMSE}_{\text{BS}}-\text{RMSE}_{\text{MLP}}$. If $l=1$, dependences between days are not accounted for. But if there are interdaily dependences and l is increased, the confidence interval should widen until the block length is large enough to capture the dependences. Eventually, the width of the confidence interval falls because of the small variability in the resampled blocks. We are then able to identify a plateau region of $C_{0.975,l}-C_{0.025,l}$, where $C_{\alpha,l}$ is the bootstrapped α-percentile with a block length l, and we choose the confidence interval as the means of $C_{0.975,l}$ and $C_{0.025,l}$, with $l \in$ plateau region. If anything, we are too conservative in determining our confidence intervals.

Figure 2 shows the width of the confidence interval as a function of l for ΔRMSE, estimated from all the 3019 pairs of model bid price errors in Table 2. The plateau region is identified for l in the range 20-26, yielding the confidence interval on the lower left in Table 2. The same procedure is then applied to all other RMSE comparisons in the Table. It can be seen that in all comparisons with the Black-Scholes model, the RMSE's are significantly lower, at the 5% level, for the MLP networks. Although not pursued here, the same steps can be taken in determining the confidence intervals for the mean estimates in Table 2.

5.1.2 Hedging

At the heart of all diffusion-based option pricing theory lies the replication of an option with a portfolio of other assets. The ability to hedge an option can be used as a measure-of-fit in model comparisons and, in many cases, it is a more accurate way to proceed. The market option prices reflect the anticipation of future price variability in the underlying asset. If these expectations are not fulfilled on average, then the market option prices are wrong and, consequently, measures-of-fit based on price accuracy are of little use. The MLP networks, on the other hand, are trained on option market data, so a "true" option pricing formula is implicitly assumed to be embedded in the noisy and possibly imperfect market prices.

A hedge analysis, however, would reveal which of the competitive models works best in the following way. Suppose that, at some point in time, for each model, we buy a *theoretical* option, the price of which is given by the model, and hedge the option until some future date, possibly the maturity date, when we sell the theoretical option and close our positions. The tracking error, that is the difference between the value of the option and the hedge position, can be used as a measure of accuracy. Either the tracking error at the termination of the hedge position or some statistics based on the tracking error during the hedge can be used, but it is the absolute value of the tracking errors that is of importance; if a theoretical option is instead sold, all signs are reversed. Since all prices and hedge ratios are calculated theoretically, the model with the lowest absolute tracking error has best captured the dynamics of the underlying asset.

Hutchinson et al. (1994) perform a similar hedge analysis with the absolute value of the tracking error at the date of expiry as a measure-of-fit. They hedge, however, a market option, not a theoretical option, which cannot be correct. Suppose their model under scrutiny is the "true" one – a change in the model option price is exactly offset by a change in the hedge. If they buy a theoretical option and hedge it to expiration, the tracking error would be zero. If the market price is larger than the theoretical price, the tracking error at expiration equals this difference and, consequently, the model is considered to be less accurate. Using the market option prices in this way is certainly illuminating, but it does not clarify if market prices correctly reflect the variability of the underlying asset.

Another way of assessing model performance, addressing the question of correctness in market prices, and with a more practical value of any improvements in pricing accuracy, is to trade in mispriced options. If at all times and for all options and all competitive models, we buy those options that are underpriced according to the models, that is, if the model bid prices are higher than the market ask prices, and sell the options that are overpriced, that is, when the model ask prices are lower than the market bid prices. We then delta-hedge each option until the overpriced options become underpriced, or the underpriced options become overpriced, or the options expire, whatever happens first. If a model can successfully identify mispriced options, then the value of the terminal position should be positive, and a model yielding a higher final value is to be preferred to one with a lower final value. Thus, the sum of the terminal profits can be used to rank different models. Due to discrete rebalancing and model misspecifications, these profits are of course not without risk (not even under continuous trading). The standard deviation of the profits may then also be of interest. Here, I do use the market options, but in a different way than Hutchinson et al. (1994). If the model price is a better predictor of future variability, then we make an initial profit, which is, on average, added to the wealth. If not, then the model is truly misspecified, and we will lose money in the long run.

Chesney and Scott (1989) also trade in mispriced options. They buy underpriced options and sell overpriced options, but they close their hedges the next day regardless of the options still are mispriced or not, which is odd. Also, their gains are computed from the midpoint of the bid-ask spread and not from actual prices.

To be more specific, the delta-hedging used here works as follows: Let V_t^I, V_t^B, and V_t^C denote the amount invested in the index, the risk-free asset, and the call option at time t. If the market option is underpriced at time 0, that is if $c_0^a < \hat{c}_0^b$, we buy the option ($V_0^C = c_0^a$), sell the index ($V_0^I = -I_0\triangle_0, \triangle_0 = \frac{\partial \hat{c}_t^b}{\partial I_t} \mid_{t=0}$), and put the rest in the risk-free asset ($V_0^B = I_0\triangle_0 - c_0^a$). When we close the position, the option must be sold at the bid price, hence the expression for

$\triangle_0$. To illustrate how the hedge works, we can decompose V_0^C and V_0^B into

$$
\begin{aligned}
V_0^C &= c_0^a = \left(c_0^a - \hat{c}_0^b\right) + \hat{c}_0^b = V_0^{C'} + V_0^{C''}, \\
V_0^B &= I_0\triangle_0 - c_0^a = -\left(c_0^a - \hat{c}_0^b\right) + \left(I_0\triangle_0 - \hat{c}_0^b\right) = V_0^{B'} + V_0^{B''}.
\end{aligned}
$$

Prior to the maturity time T, and as long as the option is not overpriced, that is as long as not $c_t^b > \hat{c}_t^a$, the positions are daily rebalanced according to

$$
\begin{aligned}
V_t^I &= -I_t\triangle_t, \qquad \triangle_t = \frac{\partial \hat{c}_t^b}{\partial I_t}, \\
V_t^C &= c_t^b = \left(c_t^b - \hat{c}_t^b\right) + \hat{c}_t^b = V_t^{C'} + V_t^{C''}, \\
V_t^B &= e^{r_t}V_{t-1}^B + I_t(\triangle_t - \triangle_{t-1}) = e^{r_t}V_{t-1}^{B'} + \left(e^{r_t}V_{t-1}^{B''} + I_t(\triangle_t - \triangle_{t-1})\right) = V_t^{B'} + V_t^{B''}.
\end{aligned}
$$

Again, the option must be sold at the bid price, hence the expression for V_t^C. The value of the replicating portfolio is then

$$
V_t = V_t^{C'} + V_t^{B'} + V_t^{C''} + V_t^{B''} + V_t^I = \left(c_t^b - \hat{c}_t^b\right) + e^{\sum_{u=1}^t r_u}\left(\hat{c}_0^b - c_0^a\right) + V_t^{C''} + V_t^{B''} + V_t^I.
$$

The last three terms only involve model bid prices and derivatives, so if we have identified the correct model and rebalance continuously, the sum of these terms should be zero. In addition, our trading strategy ensures that $V_t^{C'} > 0$ and $V_t^{B'} > 0$ when we close our position at $t = term$. Since each model is used to buy low and sell high, a successful model is expected to have a high terminal value, $V_{term} = V_{term}^I + V_{term}^B + V_{term}^C$, with little variation[9]. From this, we can define a performance measure:

$$
P = \sum_i e^{-r_{0,i} \times term,i} V_{term,i}, \tag{6}
$$

which is the sum of all discounted terminal profits.

The results are in Table 3, where P is shown for all test periods. The performance measure for periods 1-8 is not just an average of all periods. At the end of each test period, the positions are closed even though, for example, an underpriced option is still underpriced. For the overall period in the last column, all options are hedged as described above. This means, for instance, that I switch MLP networks between two test periods. In Table 3, it is shown that in many periods, hedging results in loss of money. In particular, the fifth period causes problems for both models, with a large loss for Black-Scholes and a high standard deviation for the network. This

[9] Since $V_t^{C'} = c_t^b - \hat{c}_t^b$, we can terminate the hedge already if $c_t^b > \hat{c}_t^b$ in order to have a positive (expected) terminal value.

Table 3: Hedge error comparisons, Equation (6), between MLP and Black-Scholes with historical volatility, for all eight test periods, and for the overall period. Small numbers in paranthesis are bootstrapped 95% confidence intervals.

		Periods								
Models	Statistics	1	2	3	4	5	6	7	8	1-8
MLP	P	234	-145	-150	10	-47	-331	362	-137	2096 (-714,5180)
	std. dev.	5.57	4.19	3.79	2.85	19.58	7.48	4.84	5.96	11.02
	no. of hedges	216	157	105	59	108	240	175	97	1180
BS	P	426	-685	-942	515	-2302	-960	677	352	-5019 (-14334,4537)
	std. dev.	1.68	2.98	3.68	2.91	6.82	4.00	5.36	2.63	7.17
	no. of hedges	330	323	285	123	296	369	437	154	2440
	ΔP									7116 (-458,15210)

is partly due to the fact that the hedge positions at the end of the test periods must be closed, but also because I rebalance the positions on a daily basis only. A more frequent hedging would result in a smaller dispersion of the terminal values.

To some degree, Table 3 illustrates the variation of the test periods, but the relevant measure of comparison is found in the last column. The accumulated (discounted) wealth from the hedges is almost 2100 for the MLP networks, compared to a loss of 5020 for the Black-Scholes model. The worse fit to the market option data in Table 2 for Black-Scholes results in the identification of more mispriced options. The Black-Scholes trader takes 2440 hedge positions, while the MLP trader makes 1180 hedges. Unfortunately, the Black-Scholes trader does a poor job in finding mispriced options.

The significancy of the results can be checked in a way similar to the one used in the earlier section. For each model, I construct time series by summing up the terminal profits belonging to the same terminal day. The series are extended by adding zeros for those intervening days with no termination. The moving block bootstrap is then applied to the series with various block sizes and the 95% confidence intervals for the sum of each series, as well as the difference of the sums, are calculated as described earlier.

In Table 3, P is not significantly different from zero, neither for the network model nor for the Black-Scholes model. Furthermore, the difference, ΔP, is insignificant at the 5% level.

5.2 An MLP Versus Black-Scholes with Implicit Volatility

Black-Scholes formula with implicit volatility is all about relative pricing. Each day, the volatility parameter is adjusted to coincide with the price of an at-the-money (ATM) option with a certain time-to-maturity. The resulting implicit volatility is used to price other options with the same date of expiry. Because of the spread of the ATM option, there will be one implicit volatility for bid prices and one for ask prices. The underlying assumption is that ATM options are correctly priced, and consequently, all options should be priced relative to them.

The analogy in MLP regressions is to invoke the price of an ATM option with the same time-to-maturity in the input variables, that is

$$\mathbf{x}_t = \left(I_t/K, T-t, r_t(T^{cal}-t), (c_t^b/K)_{ATM}, (c_t^a/K)_{ATM}, (I_t/K)_{ATM} \right) = (x_{t1}, ..., x_{t6}). \quad (7)$$

In a similar manner as above, for each of the eight periods, I train 20 single-layer MLP networks with 8, 10, and 12 hidden neurons, choose the one with the lowest validation error, and use it for out-of-sample pricing and hedging.

5.2.1 Pricing

The price comparisons are shown in Table 4. It is not surprising that the measures-of-fit are better than in Table 2, since I now use information from a subset of the options I am pricing[10]. From the means of the model errors, it appears that the networks slightly overprice the (fewer) ITM options, while Black-Scholes underprice them, at least for the ask prices. For the OTM options, the network prices have little bias, while the Black-Scholes prices are higher than the market prices. It seems as if the well-known "volatility smile" is somewhat skewed in this market. The comparison of all options indicates a small overpricing for both models.

The root mean squared error is lower for the network models, except for the ITM bid prices. However, for the overall comparison, RMSE is lower for the MLP networks. Using the same bootstrap method to determine the 95% confidence intervals as above, we find that the differences in RMSE's are insignificant, although the difference for the OTM bid option prices is only slightly so, in favor of the neural network models. The plateau region in this particular case is identified for l in the range of 12-17.

5.2.2 Hedging

Our competitive models no longer have a clear connection to the time series properties of the underlying index. To a large extent, the results rely on the accuracy of the ATM options. If the expectations of future variability in the index, embedded in the market price, do not come true, then some other models might perform better, despite a worse fit to the market prices. In particular, the hedge performances should not only be used to compare the two models in this section, but also to compare the models in the previous section.

[10] The monitoring for extrapolation does no longer result in any exclusions of the MLP price predictions.

Table 4: Bid and ask price error comparisons, ME, RMSE and ΔRMSE, for MLP and Black-Scholes with implicit volatility. The comparisons are for 2119 out-of-the-money options, 908 in-the-money options, and for all 3027 options. Small numbers in paranthesis are bootstrapped 95% confidence intervals.

Options	Statistics	MLP (bid)	BS (bid)	MLP (ask)	BS (ask)
OTM	ME	0.10	-0.59	-0.05	-0.67
	RMSE	1.00	1.36	1.22	1.48
	ΔRMSE	0.37 (-0.00,0.80)		0.26 (-0.02,0.61)	
ITM	ME	-0.48	0.03	-0.61	1.28
	RMSE	1.70	1.64	1.85	2.10
	ΔRMSE	-0.06 (-0.31,0.13)		0.25 (-0.19,0.72)	
ALL	ME	-0.07	-0.40	-0.22	-0.09
	RMSE	1.25	1.45	1.44	1.69
	ΔRMSE	0.20 (-0.08,0.54)		0.25 (-0.05,0.61)	

Table 5: Hedge error comparisons, Equation (6), between MLP and Black-Scholes with implicit volatility, for all eight test periods, and for the overall period. Small numbers in paranthesis are bootstrapped 95% confidence intervals.

		Periods								
Models	Statistics	1	2	3	4	5	6	7	8	1-8
MLP	P	96	39	-43	-171	-405	61	-145	-73	-266 (-1918,1381)
	std. dev.	4.82	5.41	4.29	3.83	15.76	3.54	7.27	5.03	12.13
	no. of hedges	87	98	31	42	77	48	56	80	548
BS	P	23	-4	-118	-26	-709	129	-282	115	-1766 (-3300,-379)
	std. dev.	1.59	1.93	5.80	5.14	8.62	4.13	7.77	2.51	7.46
	no. of hedges	76	65	118	60	128	99	78	56	707
	ΔP									1500 (-271,3376)

Table 6: Differences in P, Equation (6), for the MLP model in section 5.1 (MLPh), and the the two models in section 5.2 (MLPi and BSi). Small numbers in paranthesis are bootstrapped 95% confidence intervals.

Models	ΔP
MLPh-MLPi	2363 (-1759,6355)
MLPh-BSi	3863 (291,7764)

The results from the delta-hedge schemes are shown in Table 5[11]. The variation of P is seen over the eight test periods. Once more, period five seems to be a difficult period for both models, with large losses and high sample standard deviations. Because of the effects of forcing the hedges to close at the end of each test period, a fair comparison is given in the last column only. Trading in both models results in losses, but the loss for the MLP trader is limited to 266, compared to 1767 for the Black-Scholes trader. The number of hedges are 548 for the MLP trader compared to 707 for the Black-Scholes trader. The larger number of hedges for the Black-Scholes model is reflected in the worse fit to market data in Table 4.

The bootstrapped 95% confidence intervals reveal that the Black-Scholes model have a P significantly less than zero. Nevertheless, the difference, ΔP, is insignificant at this level.

Finally, Table 6 shows the differences in P for the neural network model in section 5.1 and the two models in this section. The two neural network models do not perform significantly different, while the network model in section 5.1 is significantly better than the Black-Scholes model with implicit volatility estimates, at the 5% level.

6 Summary and Conclusions

In this paper, I train feedforward neural networks, more specifically MLP networks, in a nonlinear regression of some input variables on daily Swedish call option data. I use both the bid and ask prices in order to model the spread, instead of the common approach of assuming the midpoint to be the market price. I focus on two regression setups: In the first regression, I use inputs solely from the time series properties of the underlying asset (henceforth MLPh), while in the other, I infer information from the observed option prices (MLPi). The neural network pricing formulas are compared to two benchmarks, the Black-Scholes formula with historical volatility estimates (BSh), and with volatility estimated from observed ATM options (BSi), respectively. All models are compared both according to their pricing accuracy and, as argued, more importantly, to their

[11] Since the ATM options are included in the regression, Equation (7), the hedge may be improved by a suitable position in these options.

hedging performances. Unfortunately, the complicated dependences of the option price paths make statistical inference difficult. Here, a moving block bootstrap is used to determine the statistical significance of the results.

From a pricing perspective, I find that MLPi has the lowest RMSE, followed by BSi, MLPh, and BSh. On the other hand, using the models to trade and hedge mispriced options, I find that MLPh is preferred to MLPi, followed by BSi and BSh. The results indicate that although pricing options relative to observed ATM options, as in MLPi and BSi, gives a better fit to market data, this may not be the best way to proceed. Relying on the connection between the options and the underlying asset, as modeled in MLPh, gives a better hedge. When using a hedge analysis to compare models, I find that MLPh is significantly better than BSi, but I cannot, for example, conclude that MLPh is better than BSh, even if it is likely.

An important result from a market efficiency perspective is that I cannot say that market prices are wrong, since no model gives a significant profit, at least not when the hedge positions are rebalanced on a daily basis. Trading on BSi gives a significant loss, however. This could imply that ATM options are wrong, but MLPi, also using ATM options, gives no significant loss. Hence, the conclusion would rather be that BSi does not give the proper mapping from the ATM options to other options with the same time-to-maturity.

The overall conclusion is that although the neural network models are superior, the results are often insignificant at the 5% level. Furthermore, the network models outperform the Black-Scholes formula, but the results may alter when compared to more elaborate parametric pricing models.

In practice, daily hedging is not very realistic, but I see no reason why the same methods cannot be used for data with higher frequency, or ultimately, tick data. With irregular sampling intervals, the time between trades must be included in the regressions in section 5.1, where time series properties from the underlying asset are used.

References

Black, F., and Scholes, M. (1973), "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, 81, 637-654.

Chesney, M., and Scott, L. (1989), " Pricing European Currency Options: A Comparison of the Modified Black-Scholes Model and the Random Variance Model", *Journal of Financial and Quantitative Analysis*, 24, 267-284.

Cybenko, G. (1989), "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals and Systems*, 2, 303-314.

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

Diebold, F., and Mariano, R. (1995), "Comparing Predictive Accuracy", *Journal of Business and Economic Statistics*, 13, 253-263.

Efron, B. (1979), "Bootstrap Methods: Another Look at the Jackknife", *Annals of Statistics*, 7, 1-26.

Fitzenberger, B. (1998), "The Moving Blocks Bootstrap and Robust Inference for Linear Least Squares and Quantile Regressions", *Journal of Econometrics*, 82, 235-287.

Fletcher, R. (1990), *Practical Methods of Optimization*, 2nd ed., John Wiley & Sons Ltd.

Gallant, A., and White, H. (1992), "On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks", *Neural Networks*, 5, 128-138.

Garcia, R., and Gençay, R. (2000), "Pricing and Hedging Derivative Securities with Neural Networks and a Homogeneity Hint", *Journal of Econometrics*, 94, 93-115.

Hall, P., Horowitz, J., and Jing, B. (1995), "On Blocking Rules for the Bootstrap with Dependent Data", *Biometrika*, 82, 561-574.

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

Hornik, K. (1989), "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, 2, 359-366.

Hornik, K., Stinchcombe, M., and White, H. (1990), "Universal Approximation of an Unknown Mapping and its Derivatives", *Neural Networks*, 3, 551-560.

Hutchinson, J., Lo, A., and Poggio, T. (1994), "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks", *Journal of Finance*, 49, 851-889.

Künsch, H. (1989), "The Jackknife and Bootstrap for General Stationary Observations", *Annals of Statistics*, 17, 1217-1241.

Peterson, C., and Rögnvaldsson, T. (1991), *An Introduction to Artificial Neural Networks*, Department of Theoretical Physics, Lund University, Lund.

Rumelhart, D., Hinton, G., and Williams, R. (1986), "Learning Internal Representation by Error Propagation", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, ed. by Rumelhart, D., and McClelland, J., MIT Press, Cambridge, Ma.

Scott, L. (1987), "Option Pricing When the Variance Changes Randomly: Theory, Estimation, and an Application, *Journal of Financial and Quantitative Analysis*, 22, 419-438.

Weigend, A., Huberman, B., and Rumelhart, D. (1990), Predicting the Future: A Connectionist Approach", *International Journal of Neural Systems*, 3, 193.