Machine Learning Tools for Option Pricing

Ed Harvey

July 28, 2023

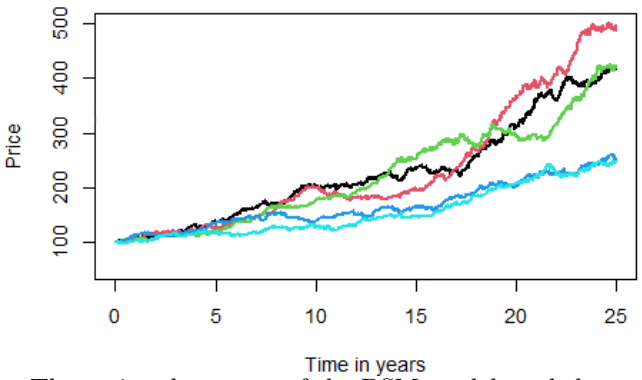
Abstract to be written at the end

Contents

1 Introduction

The publication of the Black-Scholes-Merton (BSM) model in Black & Scholes (1973) and Merton (1973) was an important landmark in the development of financial markets. The model provided formulae for valuing and hedging simple option contracts by employing stringent assumptions about the expected future path of the price of the underlying asset (normal distribution of returns with constant volatility returns both constant, see Figure 1), and the nature of the market for both the underlying and cash instruments (frictionless and efficient).

Figure 1: Simulated paths



The main advantages of the BSM model, and the reasons for its continued use in financial markets to this day, are speed of calculation and simplicity of use for traders. It has enabled banks and other financial institutions to quickly calculate the value of, and the risks associated with, large portfolios of options. The ensuing understanding of their positions and how to reduce their risks, allowed them to charge lower premiums and massively expand trading with customers.

The size of the market for derivatives has continued to expand rapidly while the assumptions behind the BSM model have been used extensively as the basic building blocks to value and hedge numerous other financial instruments. This is despite the fact that the assumptions of the BSM model have been called into question by a number of issues with its use in real-world markets.

- The model for the movement of underlying asset prices assumes that returns on the assets are normally distributed, but this has been shown not to be true in a number of studies, for example:
  - Stoyanov et al. (2011) explain that empirical distributions often exhibit fat tails, where eq-

uity prices mostly move very little on a daily basis and then, occasionally but more often than a normal distribution of returns would suggest, by a great deal. The distributions also often exhibit skew. In the case of equities, this may mean that large falls are more likely than similarly sized gains.

  - Derman & Kani (1994) show that options are traded in financial markets at prices which would imply different volatilities depending on the exercise price, implying that many of the participants are well aware of the non-normality of the distributions of returns and are adjusting the prices they quote in accordance with this understanding.

- Costless trading in unlimited quantities of the underlying asset to hedge the risk on the option is impossible in practice, meaning that assumptions that any difference to theoretical price can be arbitraged away are not entirely true.
- Continuous trading in the underlying asset to hedge the risk on the option is also impossible in practice even though times taken to agree trades have been greatly reduced in many markets.

In particular, the fat tails mean that the BSM model under-estimates the likelihood of large price moves in the underlying asset. In times of high volatility in the market, as in market crashes such as the credit crisis of 2007-2009, this can mean that sellers of options are not fully aware of their vulnerability to large losses.

Consequently, there has long been interest in improving on the BSM. The simplest approach is to continue to use BSM but to employ a volatility curve such that different volatility inputs are used for pricing dependent on the relative values of the underlying price and the strike price. Many other approaches have aimed to build models of the underlying asset dynamics which are more closely aligned with the way these markets really behave. For example, only 2 years after the original publication of the BSM model, Merton (1976) introduced the concept of occasional, large, discontinuous jumps in addition to the standard BSM distribution resulting in a greater probability of larger moves. More recently, Bouchaud & Potters (1997) used Levy processes to create distributions with fatter tails.

However, while often improving the match to real-world data of historic price movements or to market pricing of real-world options, all of these suggested models share a concept with BSM in that they are parametric models. This means that the aim is to design a sufficiently complex model of the underlying asset price dynamics, for which a series of parameters can be estimated or inferred from a mix of current market pricing and historical data. They also all therefore share, to a greater or lesser degree, the problems with using a parametric model.

- There is a trade-off between complexity and tractability. At one end, the BSM model is simple and even allows for straightforward, closed form solutions for valuing a range of derivatives. At the other end are the more complex models for which time-consuming Monte Carlo simulations or finite difference methods are required to value any derivative, even the simplest options. The complexity of using the alternative models is the major reason why the BSM model (adjusted for by traders with market-specific 'local' knowledge) is still used so extensively today despite its known failings.
- Using a parametric model together with parameters based on current market pricing will not pick up on structural changes in the underlying market which may make the model less and less relevant over time.
-

For non-parametric approaches, on the other hand, a series of economic variables which could be expected to affect the derivative's value as inputs are chosen, which are then used to train a machine learning model (ML model) to map to a given output. These approaches can avoid relying on restrictive assumptions and, while requiring significant training time, can be very fast to calculate a price for a previously unseen option once trained. They can also be continually trained on new data and therefore adjust automatically to any changing market dynamics.

The combination of improvements in computing power, and the modern neural networks methods which are now used for many machine-learning tasks, may be able to make non-parametric models an increasingly viable alternative, and they are the focus of this paper. In particular, this paper focuses on finding the ideal setup for a neural network setup with the aim of being able to predict the value of a new option in a variety of situations. The results show that very good accuracy can be achieved, without unfeasible training time being required, even in situations with noisy training data and market data being included in the training which in fact has little influence on the option value.

A literature review of prior work in this field is shown in Section 4 but before that, in order to explain some of the terms used, Section 2 gives some background on the derivatives being reviewed in this paper and Section 3 describes neural networks.

2 Derivatives

This section briefly describes the most important aspects of derivatives markets for the purposes of this paper. Many textbooks cover this subject, for example Hull (2018).

2.1 General

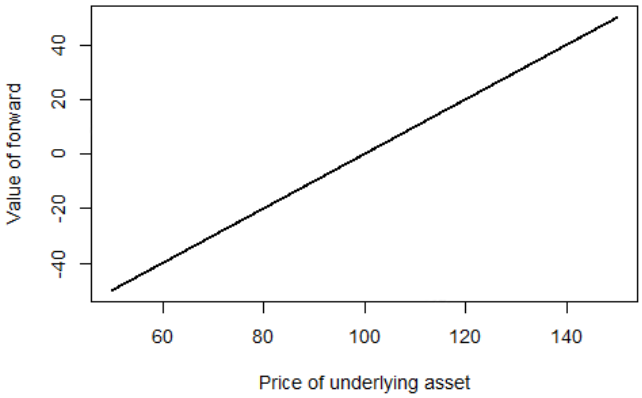
Derivatives in the context of financial markets are financial instruments whose payoffs depend on the price of some other financial instrument (the underlying). There are an uncountable number of types of derivative traded

in financial markets. In general, the more complex they are or the more bespoke to particular situations, the less frequently they are traded.

The simplest are those with a single payoff at maturity which is linearly dependent on the price of the underlying. A long position is a contract to buy the underlying at an agreed delivery price and a short position is a contract to sell. These are often known as forwards while futures are essentially a subset of these with standardised contract terms allowing them to be traded on an exchange.

Absent returns or costs resulting from owning the underlying, the value of a forward is simply the difference between the current value of the underlying and the delivery price. For example, for a long forward position with a delivery price of 100, the current value is shown in Figure 2.

Figure 2: Forward valuations



An important class of derivatives are generically called swaps because they swap one cashflow(s) for another. Most standard swaps are also linear in their payouts, for example interest rate swaps which might swap floating rate payments for fixed, but there are also more niche forms of swaps which allow specific risks included within other contracts to be hedged. For example, dividend payouts or payoffs based on realised volatility, could be swapped for fixed cash flows.

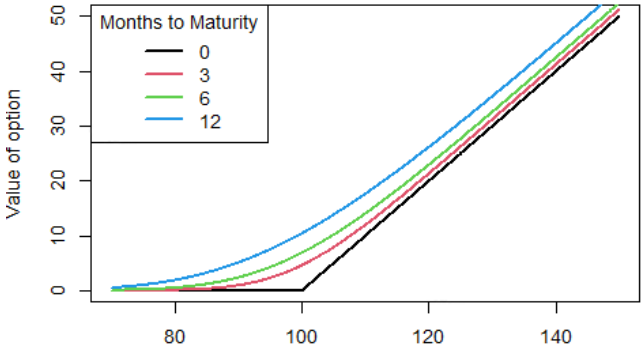
However, this project has concentrated on options as a clear example of derivatives with non-linear payoffs which cause additional complexity in valuing them.

2.2 Options

European-style options are the simplest form of option. A call gives the right, but not the obligation, to purchase an agreed quantity of an underlying asset at a contracted strike price (to exercise the contract), at a single given maturity date in the future, while a put gives a similar right to sell the underlying. Consequently, on expiry at the maturity date, a call/put may expire worthless if the price of the asset at maturity is not greater (for a call) or less (for a put) than the strike price, and will otherwise have a value of the difference between the strike price and the price of the underlying.

Prior to maturity, there is additional future value to the option due to the unlimited gains but limited losses. At trade date, a premium is paid by the buyer to the seller of the option - pricing of the premium required is often used as an equivalent term to valuing the option. The non-linear nature of the payoff makes the option's valuation more complex than that of a forward. See Section 5.1 for details of the calculation. An example for a call option with a strike price of 100, of the BSM option values for a range of underlying prices at a series of dates up to maturity, is shown in Figure 3.

Figure 3: BSM Options valuations with decreasing times to maturity



American-style options are similar to European ones except that they can be exercised at any date up to the maturity date. In many situations, there is a greater value attached to the unlimited potential upside of exercising the option in the future than to the limited downside of the option expiring worthless. In this case, the value must therefore be the same as for a European since the optimal choice is always to wait and exercise later. However, under certain conditions, the optimal choice for puts may be to exercise early, and this makes the valuation more complex.

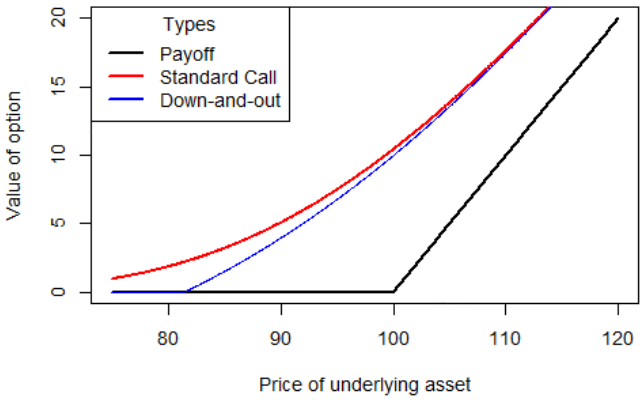
There are a number of other styles of option. Bermudans can be exercised on a limited list of pre-determined dates while Asians have a payoff which depends on the average price of the underlying over a period of time rather than at a specific maturity date. Basket options which have a payoff based on some measure of the performance of a basket of several underlyings are also commonly traded.

The second type of option reviewed in this project, in addition to the European, is the barrier option. These are generally European-style, in that they can only be exercised at a single maturity date, but they also expire worthless if, at any time up to maturity, the price of the underlying crosses (the 'out' style), or fails to cross (the 'in' style), a barrier. For example, a down-and-out call gives the holder the right, but not the obligation, to buy a specified quantity of the underlying for a specified strike price at a specified date in the future, but only if the price of the underlying never decreases to the level of a specified barrier price. The underlying price must be higher than the barrier at the moment the contract is entered into (the option value would otherwise be zero) and may also be a moving one where it starts lower and increases to the full barrier level at maturity using the

formula $B(\tau) = bKe^{-\eta(\tau)}$ where τ = Time to maturity, $B(\tau)$ = Barrier level, K = Strike price, b = Final barrier relative to K, η = Barrier decay rate with time to maturity.

There are eight possible forms of barrier option as they can be any combination of up/down, in/out and call/put. In the example of the down-and-out call, the value prior to maturity will always be below that of the standard call but will become closer as the price of the underlying increases and will deviate further as it gets closer to the barrier, hitting zero at the barrier. See Section 5.1 for details of the calculation. An example, with the value compared to that of the standard call, for a down-and-out call option with $K = 100$, $b = 0.95$, $\eta = 0.1$, $\tau = 1$, is shown in Figure 4.

Figure 4: BSM Barrier Options valuations



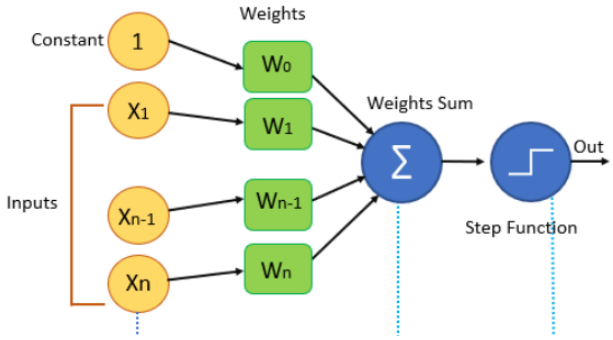
An important aspect of trading strategies in options markets is to understand not only the value of the options traded but also the change in their value that could be expected with a change in one of the inputs, that is, the first order derivative of the option value with respect to the input. These are used to help a trader to decide how much to buy or sell of some other instrument with a linear relationship to the value of the input, in order to hedge that part of the risk for a short period. In financial markets jargon, these are known as the 'Greeks' - delta (δ) refers to the derivative with respect to the price of the underlying, vega (ν) to the derivative with respect to the volatility input, and rho (ρ) to the derivative with respect to the risk-free interest rate.

3 Neural Networks

Multi-Layer Perceptrons (MLPs) are the base of modern neural networks and are described below.

3.1 Single Perceptron

Figure 5: Single Perceptron (Pedamkar 2023)



In order to understand an MLP, it is first necessary to understand a single perceptron unit (see Figure 5). This is a simplified artificial version of the neurons in a human brain. It was first proposed by McCulloch & Pitts (1943) and the first implementation was reported in Rosenblatt (1958). Given a vector of inputs $(1, x_1, ..., x_n)$, it applies weights $(w_0, w_1, ..., w_n)$ to each input (w_0 is known as the bias), sums the results, and puts the result of that through a non-linear activation function g to supply an output $\hat{y} = g(\underline{x} \cdot \underline{w} + b)$.

There are a number of commonly-used choices for $g(z)$, for example, $\sigma(z) = \frac{1}{1+e^{-z)}}$, gives a result between 0 and 1, $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$ gives a result between -1 and 1, and $\text{ReLU}(z) = \max(z, 0)$ gives a linearly increasing output for positive values of the input only.

Given a training dataset of m vectors of inputs together with m outputs $(y_1, ..., y_m)$ representing the target solutions, the model weights are adjusted from their random initial settings (calculated using one of a number of available algorithms) to minimise some loss function comparing $(y_1, ..., y_m)$ with $(\hat{y}_1, ..., \hat{y}_m)$. Common loss functions are Mean Squared Error, $\text{MSE} = \sum_{j=1}^{m}(y_j - \hat{y}_j)^2$ or Mean Absolute Error, $\text{MAE} = \sum_{j=1}^{m}|y_j - \hat{y}_j|$. Common initialiser algorithms are 'glorot' as proposed in Glorot & Bengio (2010) which scales its uniform or normal distribution by the total number of input and output units and 'he' as proposed in He et al. (2015) which scales the distribution by the total number of the input units only.

An issue with allowing a model to strictly minimise the loss for the training set is that it can over-fit, meaning that it matches the training data very well, including any noise, but does not generalise well to new input data. This is often known as the trade-off between bias (low bias means that there is high accuracy of fit to the training data) and variance (high variance means that there is high variance in the target function estimate with different sets of training data and therefore the model does not generalise well).

A regulariser term in the loss function can reduce the over-fitting by adding a penalty such as $\lambda \sum_{i=1}^{n}|w_i|$ to the loss function so that the model trains to reduce the size of the weights as well as the main loss function.

In this instance, the inputs must be normalised to be on a similar scale. Without this, the single λ would affect the weights for some inputs very differently to others.

3.2 Loss Function Minimisation

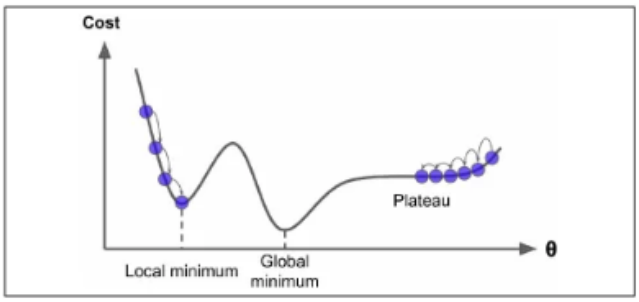
Gradient Descent (GD) is the underlying concept behind the algorithms which aim to minimise the loss function and was first studied for this purpose in Curry (1944). At each step of training the model, each weight w_i is adjusted by the negative of the partial derivative of the model's output with respect to that weight, multiplied by a learning rate η, resulting in $w_i \rightarrow w_i - \eta \frac{\partial \hat{y}}{\partial w_i}$. This continues until some convergence criteria are met (e.g. the change in the loss function is below a certain level or the number of training stages reaches a set limit).

One issue is that choosing the learning rate is key. Too large, and it will oscillate around the minimum value it is aiming for and never meet it. Too small, and it will require a very large number of training steps to find it. Momentum adjustments memorise the previous steps to allow the algorithm to increase the learning rate automatically when moving consistently in one direction. RMSprop, from Hinton (2012), and AdamW, from Loshchilov & Hutter (2019), are commonly-used optimisers which employ different methods of averaging previous steps to amend this basic momentum idea. However, some research, e.g. Wilson et al. (2017), suggests that, while these optimisers may converge quickly on an optimal training solution, they can over-fit and generalise poorly in comparison to SGD (see below) with momentum.

A second issue is that the loss function and its derivatives may take a long time to calculate. Stochastic Gradient Descent (SGD) can be used to further improve training speed, as described in LeCun et al. (2012), by using a single, randomly-chosen training data instance to calculate the derivative. This increases the variance of the adjustments to the weights at each step of the SGD process but speeds up the calculations so that training time can be reduced.

There may also be performance issues caused by loading the whole dataset and all the errors into memory at once. This can be reduced (and some over-fitting issues ameliorated) by dividing the data into batches and performing the training cycle for each batch in turn.

Figure 6: Local Minimum Problem (Yashwanth 2020)

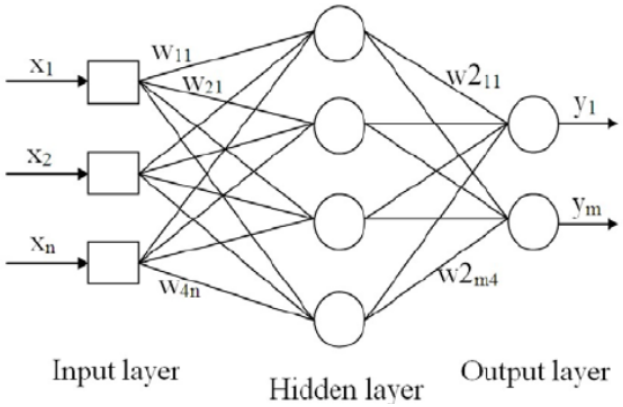


A third issue is that there may be multiple local minima or plateaus corresponding to different sets of weights (see Figure 6). It is then difficult for the algorithm to find the global minimum. A useful effect of the increased variance in the adjustments to the weights resulting from SGD, is that the optimisation may be able to break out of a search for a local minimum and find other local minima.

However, finding the global minimum can still depend heavily on the initial settings for the weights. For this reason, ensembles of models can be built, each trained with a different set of random initial weights. The average of the results of these models can be used when calculating a new price or, for speed, the model with the lowest value of the loss function as calculated on a validation dataset kept separate from the training dataset.

3.3 Multi-Layer Perceptron

Figure 7: Diagram of a Multi-Layer Perceptron (Mokhtar & Mohamad-Saleh 2013)



A single perceptron (or single layer of perceptrons) can only train on linearly separable datasets. If the function being matched is more complex, an MLP is likely to improve performance.

An MLP has hidden layers of multiple perceptrons - see Figure 7 for a 2-layer example with 1 hidden layer. Every input in a data point in the input layer feeds every unit in the first layer (meaning it is fully-connected), each with a different set of weights and biases. The outputs of that layer feed the next layer, again with a full set of different weights and biases, and so on until the outputs of the final hidden layer feed an output layer. For this project, there is only one output in the output layer, representing the ML model calculation of the value of the option.

The loss minimisation in the training process requires back-propagation of adjustments from the outputs through each of the hidden layers in turn as shown experimentally in Rumelhart et al. (1986), but the issues discussed in Section 3.2 still apply.

With the increased number of parameters available to the training process and the often more complex data being used as training inputs, the need to avoid over-fitting to the noise in the input data becomes increasingly important. In addition to the regularisation explained earlier which can be used within each hidden layer, using dropouts is a technique introduced in Hinton et al. (2012) and often used to reduce over-fitting, where a randomly-chosen proportion of the units in each layer is dropped during each round of the training process. This prevents units from 'co-working' with other units since those units may not be present in the next training batch.

A final adjustment may be made which is to normalise the outputs from one layer to another which can produce smoother gradients for the weights update in each training batch. This can be normalised for each batch or for the whole layer.

4 Literature Review of Neural Networks for Derivative Pricing and Hedging

Neural networks have been explored in previous papers as an alternative method of solving a number of issues which are most commonly dealt with using the standard parametric model approach. Examples of these issues include:

- Pricing and hedging is the most frequent and is the focus of this paper.
- The time-consuming calibration tasks for a parametric model.
- Partial differential equations which would otherwise require time-consuming numerical analysis
- Generally to replace calculation time at the point where a result is required quickly with calculation time in a training process that can take place in advance.

Concentrating on option pricing and hedging, a review of the previous literature was carried out in Ruf & Wang (2020). The major part of this records over a hundred papers on the subject between 1993 and 2020, and lists their major characteristics.

In an early example of employing machine-learning to price options, Hutchinson et al. (1994) trained neural networks on 2 tasks. First, it was trained using inputs of a simulated series of underlying asset prices and a set of values for the exercise prices and the times to maturity for each of these, learning to match the output of the Black Scholes model for the same inputs (with constant risk-free interest rate and volatility. Second, it was trained using inputs of historical underlying asset prices to match the equivalent historical option prices for a range of strikes and times to maturity. They then tested these in two ways - by calculating the correlation between prices calculated by each ML model and the BSM model, and by comparing the tracking errors of using each to hedge a set of options. In particular, for the second experiment, they found that these tracking errors were often lower for the ML model than for a naive implementation of the BSM model.

An development of the second experiment, as proposed by Lajbcygier & Connor (1997), could be to use the BSM model as a base together with a neural network to learn the difference between the modelled price and actual market prices. They used an implied volatility from a range of option prices as the volatility input to the BSM model and as a training input to the ML model. Their results showed much better risk-adjusted returns from using their model than using BSM.

Raberto (1999) trained neural networks to match the output of the Bouchaud-Potters model (as explained in Bouchaud & Potters (1997)) in addition to the BSM model. This gave results that were closer to those of observed option prices than using the BSM model and showed the validity of the concept of using neural networks to learn to produce the results of more complex models.

An additional possible adjustment in the target of the neural network is to model hedging strategies rather than prices of options, as first proposed in Carverhill & Cheuk (2003). In this case, movements in observed underlying prices were used as training inputs while movements in observed option prices were used as the training output. They used a very simple neural network resulting in large error margins but they found nonetheless that tracking errors on observed data were consistently lower using their model.

An adjustment to the training of the model to constrain the outputs it might produce to known features of the market (that is, it factors in financial domain knowledge) has been suggested in a number of papers including Dugas et al. (2009). They point out that the value of a standard call option is monotonically increasing with underlying price and with time-to-maturity and is convex with underlying price. In simulation exercises, they showed that the constrained models resulted in lower MSE losses than the unconstrained ones for any given length of training.

A further complexity for which neural networks may provide good answers is hedging portfolios of derivatives outside the theoretical world of frictionless markets which is very hard to do using parametric models. Bühler et al. (2018) propose a framework for pricing and hedging, without observing option prices, and in the presence of both liquidity restrictions on trade size and trading costs. They use convex risk measures as loss functions in training in order to find the cost of the most efficient trades in the available hedging instruments.

5 Experimental Design

This project has concentrated on one aspect of the topic. That is, determining the optimal MLP setup for matching the results of BSM formulae for problems of differing complexity. Six experiments were carried out:

1. A standard European call option on an underlying asset with no other associated cash flows (e.g. dividends, interest payments, costs of storage, etc) other than its purchase price. The training inputs to the ML model were based on the underlying price relative to the strike price and the time to maturity with the risk-free interest rate and volatility assumed to be constant while the training output was BSM valuation based on the same inputs.

2. As above but adding a range of values for the risk-free interest rate and volatility as inputs. The intention was to determine whether an increased number of inputs increases the advantages of using deeper models with more units.
3. As above but replacing the standard call with a down-and-out barrier call. The additional complexity of the option together with the additional inputs of a range of values for the barrier level and its decay rate might again be expected to require a more complex model.
4. As above but adding a normally distributed noise element (standard deviation of 0.1) to the results of the BSM formulae used for the training outputs. This was also intended to show whether more noisy training data increases the advantages of using deeper models with more units but, in addition, whether the noise creates over-fitting issues which require regularisation techniques to reduce them. It aimed to mimic the situation which would arise if historical market option prices were being used as the training output for the model instead of the exact results of the BSM model.
5. As above but with a larger noise element (standard deviation of 0.4).
6. As above but with an additional dummy variable that the model should ignore - if it does not, it will produce an over-fitted fit to the training data which will not generalise well to new data, This was also intended to test when regularisation might be expected to improve performance.

5.1 Target Solutions for ML Model

5.1.1 Black-Scholes-Merton Model

In all of the experiments, the analytical solutions for option valuations which result from the BSM model were employed as the training output for the ML models. The assumptions of the BSM model are detailed below:

- The return on the underlying has a normal distribution (i,e. the underlying price follows Geometric Brownian Motion or GBM). That is, for a small movement forward in time from t to $t+\epsilon$, and where $S(t)$ is the price of the underlying at time t, $\frac{S(t+\epsilon)}{S(t)}-1$ is normally distributed with mean $= \mu$ and standard deviation $= \sigma$ which are both constant. μ is often referred to as the expected rate of return and σ is often referred to as the volatility.
- There exists a risk-free asset whose risk-free rate of return is r.
- There are no costs or returns (e.g. dividends, interest, costs of storage) from the underlying apart from the ability to trade it in the market at a later date.

The assumptions about the markets in which the underlying and the risk-free asset are trading are:

- All market participants can continuously buy or sell any amount of the underlying or the risk-free asset.
- Consequently, there are no arbitrage opportunities to make a risk-free profit as all market participants could use them.
- There are no fees or costs incurred in making these transactions.

Several of these restrictions are relaxed in later papers extending the BSM model (e.g. time-varying but deterministic risk-free return or volatility) but this project will keep the original assumptions for simplicity.

5.1.2 European Call

The first 2 experiments modelled the standard European Call option. Given the BSM assumptions about the market above, the option can be perfectly hedged by continuously buying and selling the underlying and risk-free assets in order to obtain an overall position which is risk-free. The return on this overall position, given the condition that no arbitrage opportunities exist, must be equal to that of the risk-free asset. The partial differential equation (PDE) below for the option value (V) then follows.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

In the case of European options (with $T =$ option maturity date, $t =$ current date and $K =$ strike price), this can be solved to give a closed form solution by including the boundary conditions which, for a call option value ($C(S,t)$), are:

- $S = 0$: $C(0,t) = 0, \forall t$
- $S \to \infty$: $C(S,t) \to S - K$ as $S \to \infty$
- Maturity payoff: $C(S,T) = max\{S-K, 0\}$

The call option value is then:

$$C(S,t) = N(d_1)S - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left(ln(\frac{S}{K}) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right)$$
$$d_2 = d_1 - \sigma\sqrt{T-t}$$

For machine learning purposes, it is often helpful to reduce the number of training inputs as far as possible. In this case, using $s = \frac{S}{K}$ and $\tau = (T-t)$, the equation can be reformulated to require 2 fewer training inputs and the result simply multiplied by K to get the required value for $C(S,t)$:

$$C(S,t) = K \cdot C(s,\tau) = K\left(N(d_1)s - N(d_2)e^{-r(\tau)}\right)$$

$$d_1 = \frac{1}{\sigma\sqrt{\tau}}\left(ln(s) + \left(r + \frac{\sigma^2}{2}\right)\tau\right)$$
$$d_2 = d_1 - \sigma\sqrt{\tau}$$

5.1.3 European Down-and-Out Call

The remaining experiments modelled the European Barrier Call option with moving barrier. The same assumptions can be used to produce a similar PDE to the one in Section 5.1.2 (from Akinyemi (2015)) but with amended boundary conditions including adjustments for b = barrier level relative to strike price at maturity and η = decay rate of b with increasing τ such that B = absolute barrier level = $bKe^{-\eta\tau}$.

- $S = 0$: $C(0,t) = 0, \forall t$
- $S \to \infty$: $C(S,t) \to S - K$ as $S \to \infty$
- Maturity payoff: $C(S,T) = max\{S - K, 0\}$
- Barrier: $C(B,t) = 0$

The down-and-out call option value ($C_{DO}(S,t)$) is then:

$$C_{DO}(S,t) = C(S,t) - \left(\frac{S}{B}\right)^{1-\frac{2(r-\eta)}{\sigma^2}} C\left(\frac{B^2}{S}, t\right)$$

This can again be reformulated using $s = \frac{S}{K}$ and $\tau = (T-t)$ as:

$$C_{DO}(S,t) = K \cdot C_{DO}(s,\tau)$$

5.2 R Implementation of Neural Networks

Open-source packages are available, which enable the Python implementations of TensorFlow and Keras to be used in the R computing language. These allow a range of models to be built with a chosen number of layers and units, an activation function and regulariser to use for each layer, and a loss function and optimiser defined. These models can then be trained on a proportion of the input data, when the number of epochs and the batch size are defined, while being validated on the remaining input data. Finally, the model can be used to predict the correct output from a set of input data that was not used during training.

The code example below would build a standard MLP with 1 hidden layer of 200 units, a ReLU activation function, a regulariser, an MSE loss function and an RMSProp optimiser. It would train its training inputs and output using 5 epochs and a batch size of 128 on 70% of the input data, and validate the results on the remaining 30%. It would then be able to produce a predicted set of outputs from a new set of inputs:

```
model = function(ncols)
  {models = keras_model_sequential() %>%
  layer_dense(units = 200,
    activation = "relu",
    input_shape = ncol(training_inputs),
    regularizer_l2(0.01)) %>%
  layer_dense(units = 1) %>%
  compile(optimizer = "rmsprop",
    loss = "mse",
    metrics = c("mae"))}

learns = model %>%
  fit(x = training_inputs,
    y = training_output,
    epochs = 5, batch_size = 128,
    validation_split = 0.3,
    verbose = TRUE,)

model %>% predict(testing_inputs)
```

5.3 Data

Simulations were run of daily asset prices over a given time period, for each of which a series of datapoints with a range of values for the training inputs relevant to the experiment being run were created. For each training run to create a single model, these datapoints were used as the training inputs and an option price based on these inputs was used as the target solution in the training output. The details are listed below:

Inputs:

- Underlying asset prices. The simulations were run for a 480 day time period, based on the assumption that the distribution is described by GBM. That is, for an initial price $S(0)$ at time $t = 0$, the price $S(t)$ at time $t > 0$ is determined by the formula $S(t) = S(0)e^{\sum_{i=1}^{n} Z_i}$, where n is the number of incremental time periods δt from 0 to t, and Z_i is a draw from the standard normal distribution $N(\mu\delta t, \sigma^2\delta t)$, where μ is the annual drift (or expected return on the asset) and σ is the annual standard deviation. 240 days was considered the annual time period and 20 days was considered a month to remove the unnecessary complications of including the effects of weekends.
- Maturities. A range of maturity dates for reference European call options. These were set at 1 month (20 day) intervals from the current date of the simulation up to half a year beyond the final simulation date.
- Exercise prices. A random range of exercise prices for reference European call options. These are based on a uniform distribution between 0.5 and 2 times the current underlying asset price. Given the σ and maximum time to maturity, the function for the option price outside these values becomes close to linear.
- Risk-free interest rate and volatility. For the first experiment, r and σ were set at 0.05 and 0.2 respectively. For later experiments, r and σ were also given a random range, of (0.01, 0.12) and (0.05, 0.5) respectively.
- Barrier contract terms. For the third and later experiments, the additional contract parameters for the barrier options, b and η, were added with ranges

of (0.6, 0.99) and (0, 0.3) respectively. If the resulting barrier was greater than the underlying price, the data was removed as the value is zero by definition and does not need to be modelled.

- Dummy. For the sixth experiment, a dummy variable was included in the training data with a range between 1 and 5, although since this is normalised and is not included in the calculation of the training output below, the range of inputs does not affect the modelling.

Output:

- Option prices. Prices for the options were based on the BSM solutions described in Section 5.1 using the training inputs described above. For the fourth and later experiments, a normally-distributed noise was added to the exact BSM price. This was calculated as a proportion of the BSM price with a mean of zero and standard deviations of 0.1 for the fourth and 0.4 for the fifth and sixth experiments.

For testing of the effects of using ensembles of models (see Section 5.4.1), and for reviewing tracking errors (see Section 5.4.2), additional simulations were used to create datasets of testing inputs and outputs. These were created for only 240 days of data and the noise element was always set at zero since this is a test to check whether the model has managed to find the underlying function despite the noise in the training data.

Each of the training inputs was normalised by subtracting its mean and dividing by its standard deviation. The testing inputs were normalised using the mean and standard deviations of the training inputs in order to treat new data in exactly the same way as the data that the model was trained on.

5.4 Evaluation

5.4.1 Hyper-Parameter Optimisation

It can be seen from Section 3 that there are a large number of choices that can be made when implementing an MLP. These are often referred to as hyper-parameters to distinguish them from the weights being trained to allow the model to better calculate results for new input data. Those that have been reviewed in this project are listed below:

- The number of units in each hidden layer (the number of units in the input and output layers are determined by the problem).
- The number of hidden layers.
- The loss function.
- The activation function for each layer.
- The optimisation algorithm to employ in minimising the loss function.
- Extra inputs to the optimisation algorithms that adjust how they operate.
- The regulariser, if any, and its size.
- The level of any dropouts to be used between layers.
- The parameter initialisation algorithm.
- What inter-layer normalisation algorithm to use, if any.
- Number of epochs - the number of times to loop through all the training data during the training process.
- Batch size - the number of sets of training inputs to test before updating the weights.
- Number of models to include in an ensemble.

A range of possibilities for each of the hyper-parameters above, with the exception of the ensembles, was evaluated based on the MSE losses, balanced against the time taken to train the model and the time taken by the model to calculate values for a new unseen set of input data.

For the first three experiments, the MSE losses were calculated automatically as part of the R implementation, using the proportion of the input data held back for validation. However, for the fourth and fifth experiments testing noisy training outputs, a new set of testing data was created in a similar way to the training data but with zero noise.

There are many combinations of possibilities for setting the hyper-parameters and it is always possible that changing one will alter how changing others will affect the losses on the validation set of input data. Given that it would be impossible to test all possible combinations, and to keep the number of experiments down to an achievable level, a reasonable initial set was used and then a range of each of the hyper-parameters were tested in turn while holding all the other parameters steady. At each successive stage, the set of parameters being held steady was adjusted to include the best parameter from the previous stage.

In many cases, when setting the reasonable initial set of hyper-parameters, the default settings in the Keras package gave a useful base value for the hyper-parameters. In others, it was possible to start simple (e.g. one hidden layer) then increase model complexity until a point is reached where no significant performance improvement is obtained.

Testing the ensembles of models required a slightly different approach. There are two ways of using an ensemble - if there is a need to get quick answers at slightly lower accuracy, the model with the most successful testing results could be used while if high accuracy is most important, an average of the results of all the models in the ensemble could be used instead. At the point of training each of the models within the ensemble, neither the mean nor the minimum loss across training runs have been calculated on validation dataset. Therefore, in order to test the use of ensembles of models, the testing dataset was used instead.

The testing of the effects of using the mean or min loss ensemble method was also used to test the effects

of employing different tactics to reduce losses. Training time was quadrupled by increasing, firstly, the number of models trained, secondly, the quantity of input data used in each model and thirdly, the number of epochs. The performance benefits gained by employing these different tactics were then compared.

5.4.2 Tracking Errors

The BSM assumes 'frictionless' markets which, among other things, means that price moves are continuous and that trading in the underlying asset can be performed continuously and without cost. The possibility then exists for continuous re-balancing of a position in the underlying asset to hedge any risk that the outstanding option position changes in value. The amount of the asset required to do this, for an option on a single unit of the underlying, is given by the derivative (or 'sensitivity') of the option price with respect to the price of the underlying (the delta as explained in Section 2.2). If this process is performed continuously throughout the life of the option, then, including the cash from the option premium, the total of cash and value of underlying immediately after the the option expires will equal zero.

In the real world, among other issues, this re-balancing of the hedge can only be performed in discrete steps and also incurs trading costs. The effect of trading costs has not been modelled in this project but the simulated underlying asset prices have been used to re-balance the hedge discretely on a daily basis in a similar way to Hutchinson et al. (1994).

The notation is used that, for time t, the value of the option position is $V_c(t)$, the value of the underlying position is $V_S(t)$ and the value of the cash position is $V_b(t)$ (which it is assumed can be invested or borrowed at the risk-free interest rate). $V(t)$ is the total value of the portfolio of option, underlying and cash, and therefore:

$$V(t) = V_c(t) + V_S(t) + V_b(t)$$

The notation is used that the value of a call option on one unit of the underlying is $C(S, t, r, \sigma)$ and its delta is $\Delta(t) = \frac{\partial C(t)}{\partial S(t)}$. It is also assumed that the seller hedges the option position every day by holding a $S(0)\Delta(0)$ of the underlying. At $t = 0$, the option seller has therefore gained cash equivalent to the option premium and paid out cash equivalent to the cost of the position in the underlying.

Thus, since these transactions were carried out at fair value due to the frictionless market, the initial composition of the portfolio at time 0 for a seller of the option is:

$$V_c(0) = -C(0); V_S(0) = S(0)\Delta(0)$$
$$V_b(t) = -\left(V_S(0) + V_c(0)\right)$$

$$=> V(0) = V_c(0) + V_S(0) + V_b(0) = 0$$

Each following day until maturity at $t = T$, the amount of the underlying required to hedge the option delta will change. Therefore, the cash value in the position will change by an amount opposite to the amount of underlying bought or sold to achieve this (in addition to earning r = risk-free return on the cash that was held at the end of the previous day).

Therefore, using $\delta t = 1$ day, the composition of the portfolio at time t becomes:

$$V_c(t) = -C(t); V_S(t) = S(t)\Delta(t)$$

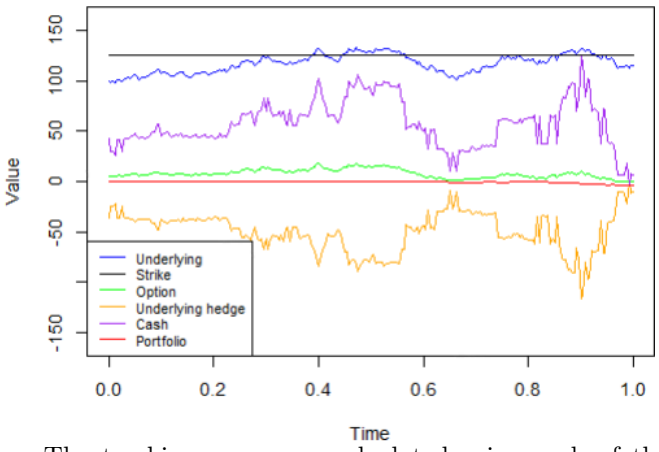
$$V_b(t) = e^{r\delta t}V_b(t - \delta t) - S(t)\left(\Delta(t) - \Delta(t - \delta t)\right)$$

.

The result is a path of valuations of the trading strategy as a whole, the absolute value of which is the 'tracking error' of the hedging strategy. An example is shown in Figure 8 using the preferred ML model with the lowest loss from the second experiment for an option with strike price 125 and a maturity at time 0 of 1 year. The lines representing the values of the option, the underlying hedge and the cash sum to give the value of the portfolio.

As expected, the value of the portfolio stays close to zero, even as the value of the underlying, and consequently the amount of the underlying required to hedge the option, vary significantly. Note that, as the option approaches maturity, even a small change in the underlying price, if it crosses the strike price, can require extreme changes in the underlying hedge requirement. This is one reason why, in the real world with transaction costs and limits on the ability to trade continuously, this hedging strategy can be difficult to carry out.

Figure 8: Values of elements of option hedging portfolio against underlying and strike prices



The tracking error was calculated using each of the BSM model and the ML model and compared. In this project, the training outputs have been based on the BSM prices and the simulations of price changes in the underlyings have been based on the assumptions included within the BSM model. Consequently, it would be expected that, on average, using the BSM model would result in smaller tracking errors than the ML model.

5.4.3 Model Comparison

The following measures were calculated for the purposes of comparisons between models:

1. The MSE of the ML model option prices with respect to the BSM option prices, referred to as losses in this paper as this is the loss calculation which is minimised for the training data during the training process. A figure closer to zero means that there is little difference between the two.
2. The R-squared measure on the series of ML model and BSM option prices. The closer this is to one, the more correlated the ML model prices are to the BSM ones.
3. The mean of the difference between the ML model and BSM prices. A figure nearer to zero means that there is little bias in the difference.
4. The time taken to train the ML model, referred to as training time here.
5. The time taken to calculate the price of a set of previously unseen options, referred to as testing time here. This will often be more important than the time take to train the model as a pricing model may be used at times when a set of prices are needed relatively urgently while training can take place in advance. However, training time cannot be completely ignored as processing power/time is not unlimited and updates to the training may be required at regular intervals or when something about the market changes.
6. The tracking errors using the ML model and BSM, as described in Section 5.4.2.

6 Results

?? Some sort of referencing to spreadsheets of results in Github to be added

6.1 Hyper-Parameter Testing

Each of the hyper-parameters and the effects of adjusting them are summarised below in turn. In the first experiment they were initially set at the values shown in Table 1. In the later experiments, the initial setting was the preferred final setting of the previous experiment. They were all tested based on averaging 4 training runs as described in Section 5.3.

Table 1: Initial Hyper-Parameter settings

Hyper-Parameter	Initial
Activation	ReLU
L2 Regularisation	Zero
Optimisation	RMSProp
Dropout	Zero
Batch size	32
Number of layers	2
Units per layer	100/50
Normalisation	None
Initialiser	Glorot_unif
Number of epochs	10

Activation function

ReLU was the best option in all 6 of the experiments, although the difference in validation losses between using ReLU and tanh was a lot smaller for the fifth experiment (down from several times lower to 20% lower) when there was greater noise in the training output but no dummy variable.

L2 regularisation

As expected, any amount of regularisation increased losses significantly for the first three experiments with no noise or dummy variable in the training output. However, for the fifth and sixth experiments with large noise elements, a small L2 regularisation factor of 0.0001 reduced losses by over a third relative to using zero and by a little more when the dummy variable was added to the training inputs.

Optimisation

For the first, simplest experiment, SGD with a high momentum was most successful in reducing losses with about 30% lower losses than adamax or nadam (two derivatives of the adam algorithm). It was possibly surprising to see that rmsprop performed very poorly for this experiment given its use as a default in the Keras package. For experiments with more training inputs and/or noise in the training output, adamax became the best-performing optimiser. SGD with momentum was a lot less successful (and rmsprop less unsuccessful) in the fifth and sixth experiments with increased noise.

Dropout

Since dropout is another regularisation technique, it showed a similar direction in results to the L2 regularisation except that it was still marginally better to use zero dropout even for the fifth and sixth experiments. Using dropout also causes the training time to increase significantly (by 15-55%).

Batch Size

The choice of batch size is generally a trade-off between increasing losses against decreasing training time as batch size is increased. This is because a smaller batch size causes a processing cost from recalculating the gradients and weights more frequently but at the same time this speeds up the convergence per epoch. The exception is the sixth experiment with the dummy training input where 128 and 64 have lower losses than 32. For this paper, the level chosen has been 64 throughout - 32 generally gives slightly lower losses but the training time has been 50% greater or more.

Number of layers

As the number of training inputs increased, the increase in training time caused by using more layers became greater. Possibly due to the relatively simple form

of the training output, e.g. the fact that it is monotonically increasing, there was little reduction in losses from increasing the number of layers past 2 even when the noise element was added. 1 layer resulted in 50%+ greater losses than 2 layers for all experiments other than the 6th for which it was only 25%. 2 layers was chosen for the preferred ML model.

Units per layer

As would be expected, increasing the number of units per layer generally reduced losses while increasing training and testing times. For the first three experiments with no noise element, the ranges tested implied that, for a given number of units, the trade-off between losses and processing time was improved for a roughly equal number of units. For the fourth to sixth experiments, this was less clear and for the sixth in particular, there was no performance advantage of moving from 100 units to 200 units in either layer.

For this paper, 200 units in each layer have been chosen as giving the best balance for experimental purposes except for the fifth experiment where 200 in the first layer and 100 in the second layer was considered the best choice and the sixth where 100 and 100 was preferred.

Normalisation

Using either batch and layer normalisation resulted in several orders of magnitude larger losses as well as increased training and testing times, with the exception that the effect on losses from layer normalisation was reduced to 3-4 times for the fifth and sixth experiments.

Initialiser

Glorot_uniform and glorot_normal generally showed only small differences in performance. With only 4 training runs, it may have been chance which one appeared slightly better for each experiment. It was perhaps surprising that he_uniform and he_normal consistently showed many times larger losses (although much lower difference of only 50% more in the fifth and sixth experiments with a large noise element) since the literature implies that these are preferred for neural networks using the ReLU activation function as those here are.

Number of epochs

As would be expected, increasing the number of epochs of training consistently reduced losses. However, the gains from doubling the training time by doubling the number of epochs quickly fell away to the extent that 10 epochs was considered sufficient for the experiments. However, a trend could be detected showing that more training inputs (and therefore a more complex training process) led to a significant advantage being gained from increasing the number of epochs while more random noise in the training output reduced this advantage. This is possibly because, after a certain point, training the model to match the noisy input data ever more closely did not help it to generalise to new data.

Model Ensembles

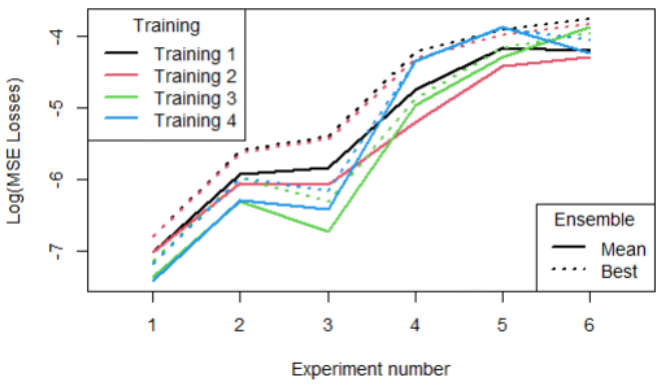
Table 2 shows the mean MSE losses across 10 testing runs for the preferred model resulting from each of the six experiments. The mean loss for each experiment is shown for each of the mean or best method of using the ensemble of models, and for the four different training conditions listed below. The aim is to determine, if a performance improvement is required and training time is to be extended to achieve this, which is the best option for performance. Figure 9 shows the same information in a chart except that the log (base 10) of the losses is used to allow them to be compared.

1. The baseline situation which is 4 training runs (and therefore 4 models in each ensemble), just under 180,000 items of training data, and 10 epochs of training.
2. As for 1, but a fourfold increase in the number of training runs to 16.
3. As for 1, but an approximately fourfold increase in the number of items of training data to around 710,000.
4. As for 1, but a fourfold increase in the number of epochs of training to 40.

Table 2: MSE Losses (1e-06) of ensembles of models per experiment

	Type	1	2	3	4	5	6
1	Mean	0.10	1.21	1.49	18.29	70.25	63.89
1	Best	0.16	2.51	3.97	61.53	123.76	172.59
2	Mean	0.09	0.16	0.85	6.43	39.04	52.59
2	Best	0.16	2.35	3.64	49.31	106.37	146.58
3	Mean	0.04	0.49	0.19	11.18	51.97	137.26
3	Best	0.07	1.03	0.49	13.63	70.56	111.25
4	Mean	0.04	0.52	0.39	45.65	138.97	59.53
4	Best	0.07	1.03	0.69	45.89	130.89	90.78

Figure 9: Log of MSE Losses of ensembles of models per experiment



There are a number points of interest in the trends shown by these results, for example:

1. The losses are generally less for Experiment 3 than for 2 despite the increase in the number of training

inputs required to price barrier options. This could possibly be due to the relatively simple linear relationship between the option price and the underlying price over the shorter range of underlying prices for which the option has been modelled (the underlying price cannot be below the barrier).

2. There are large percentage increases in losses with increasing levels of noise in the training output from Experiment 3 to 4 and then to 5.

3. The losses are fairly similar on average across the training condition options from Experiment 5 to 6 despite the introduction of a dummy variable which might be expected to introduce more noise and complexity into the training process. To be more precise, while training conditions (2 and 3) that had more training data showed larger losses for Experiment 6 than for 5, training with more epochs showed lower losses for Experiment 6 than for 5.

4. For all experiments other than 6, if lower losses and improved accuracy is required and extra training time can be employed to achieve this, then increasing the quantity of training data is the best method to achieve this. Increasing the number of epochs of training cycles is second best (and best for experiment 6) while increasing the number of training runs to create a larger selection of models in the ensemble is the least successful strategy.

Best Model Choice

The choice of 'best' model for each experiment very much depends on what purpose the model will be used for. If the model needs to be re-trained frequently and processor time might be short then low training time is important. If it will be used to generate multiple new prices that are required urgently, then low testing time is important. If it will be used in a situation where very exact prices are required then low MSE losses are important.

For the purposes of the experiments in this project, the relative level of the losses between the experiments was most important rather than obtaining the lowest absolute level of losses. However, there were frequent experiments and restrictions on the available processor power to perform them. Consequently, the choice of model has been to get the lowest possible losses while not extending the training time, or to a lesser extent the testing time, further than necessary. This also means that. for later analysis, the method of using the model with the best testing loss within an ensemble, rather than the mean of every model within an ensemble, has been chosen.

6.2 Performance Analysis

Comparable performance measures for all of the experiments are shown in Figure 10. The option to quadruple the number of items of training data in each training dataset to 710,000 was used in all cases. All testing results were averaged over 10 testing runs, each of which are for 1 year of simulated prices with around 60,000 items of testing data in each testing dataset. Training time is time taken to train all of the models within the ensemble. Testing time is time taken to price the 60,000 options in the testing dataset.

The results clearly show the reduction in accuracy as the difficulty of training the models increases due to increasing the number of training inputs from 2 to 4 to 6, from adding the noise element to the training outputs and finally from adding a dummy variable as a seventh training input with zero effect on the training output. However, even Experiment 6 shows R-squared of 99.8% and an average tracking error of 1.70 (1.07 higher than BSM) so this is an encouraging performance for a model setup that was chosen for speed in preference to absolute highest accuracy and was presented with highly noisy training data.

The charts shown in Figure 10 are based on a single testing run, plotting the ML price against the BSM price and then the error (relative to the strike) against both the level of the underlying price (relative to the strike) and time to maturity. The points about relative performance between the two experiments made above are confirmed but, in addition, the charts for Experiment 6 errors show two clear trends (both of which are confirmed if alternative testing runs are reviewed). One is for increased errors, often to over-estimate the true price, at very short times to maturity. The second is a tendency to slightly over-estimate the true price when the underlying is much lower than the strike price (and the option price consequently should be close to zero), under-estimate when the underlying is somewhat lower, and then increasingly over-estimate when the underlying is higher than strike price (and the option price consequently should be very close to the (underlying - strike) price).

6.3 Comparison to Past results

Hutchinson et al. (1994) is an influential paper which (in part 1 of the paper) tested a similar set of models and simulated paths to Experiment 1 in this paper. Although the experiments are not exactly comparable, the results of their paper show some similar trends but improved performance, presumably due to the advancement of both computing processing power and open source ML algorithms. For example, although they do not state the time required to train their model on simulated paths, they only felt able to to train it on around 6,000 data points compared to 700,000 in this paper. This is despite their training being run on a large mainframe of the time and this paper using the CPU of a standard University lab machine. There is a mention in part 2 of the paper of requiring many minutes per training cycle while this project found that 10 seconds was required per epoch for the much increased quantity of training data.

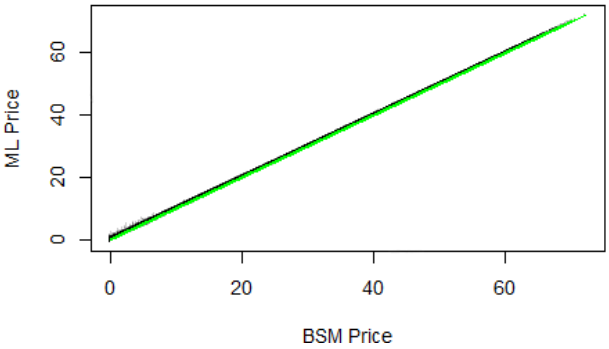
To generate the comparison figures in Table 3, 100 tests were run for Experiment 1 with a reduced time to maturity of 0.5 years and K of 0.8, 1.0 and 1.2 of the

Figure 10: Performance measures and charts illustrating the model errors

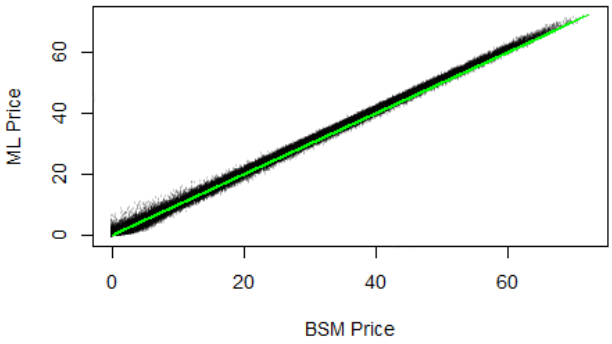
(a) Performance measures for Experiments 1 to 6

Measure	1	2	3	4	5	6
MSE (1e-06)	0.07	1.03	0.49	14.20	86.20	106.00
Mean R^2 (%)	100.000	99.995	99.999	99.969	99.827	99.673
Mean error	-0.00	-0.10	-0.01	-0.11	0.18	-0.09
ML Tracking error	0.47	1.38	0.94	1.30	1.03	1.80
BSM Tracking error	0.23	0.60	0.35	0.35	0.35	0.35
Training time (s)	380	667	833	617	364	343
Testing time (s)	1.14	1.31	1.43	1.18	1.12	1.05

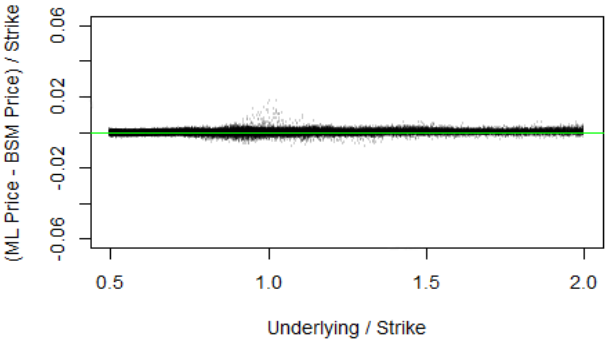
(b) Model option values vs BSM for Experiment 2



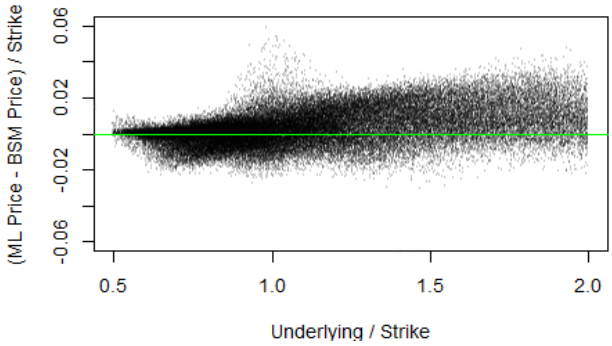
(c) Model option values vs BSM for Experiment 6



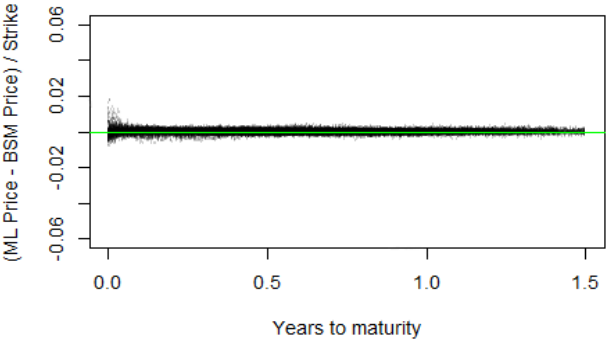
(d) Model errors/strike price vs underlying/strike for Experiment 2



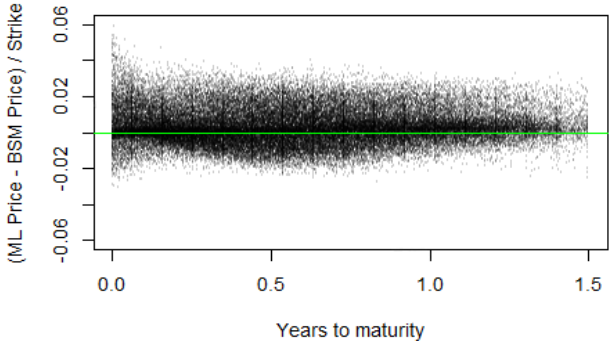
(e) Model errors/strike price vs underlying/strike for Experiment 6



(f) Model errors/strike price vs time to maturity for Experiment 2



(g) Model errors/strike price vs time to maturity for Experiment 6



starting underlying price. The means were taken for R^2 and the BSM model and ML model tracking errors. R^2 as a percentage is directly to the R^2 shown in the paper for an MLP. The tracking errors and the ones shown in Hutchinson et al were taken as a percentage of K to make them comparable.

Table 3: Comparisons to Hutchinson et al performance, mean of 100 tests

Measure	Hutchinson	Exp 1
R^2	99.48	99.9996
Tracking $0.8 \cdot S(0)$ BSM	0.176	0.066
Tracking $0.8 \cdot S(0)$ ML	0.572	0.344
Tracking $1.0 \cdot S(0)$ BSM	0.410	0.329
Tracking $1.0 \cdot S(0)$ ML	0.610	0.414
Tracking $1.2 \cdot S(0)$ BSM	0.294	0.237
Tracking $1.2 \cdot S(0)$ ML	0.428	0.353

7 Discussion

7.1 Practical Uses

Although non-parametric models have not been heavily used in pricing and hedging in financial markets to date, they could have advantages that would be advantageous in several practical situations.

1. Day-to-day trading and valuation of portfolios of options. This project shows that it is possible to closely match prices and risk measures for non-linear financial instruments that use a simple model of underlying asset price movements. It should therefore also be possible for more complex situations which would otherwise require time-consuming methods to solve, meaning a long window before a price can be given to a customer or a full valuation of a portfolio can allow a trader to review the risks that need to be hedged. For example:
   - More complex options, e.g. hybrids such as convertible bonds or basket options where the option is exercisable on more than one underlying asset.
   - More complex models of the underlying, e.g. Bouchaud-Potters as discussed in Raberto (1999), which may better model the fat tails and skew of real-world distributions.

   Pricing these using ML models could exchange the lengthy periods required to train the ML model on multiple examples of the calculated option value, against the much shorter periods required to obtain a price when it is more urgent for trading purposes.
2. Finding current market practice in valuing options. Market option prices could be used as training output and both the contract details and a range of other data in the market as training inputs to build a model of how the market participants are adjusting their models to take the issues of the BSM model into account. This project shows that creating an ML model for a non-linear financial instrument is feasible, even when, as in this usage, there would probably only be noisy data available and variables might be included in the training which which turn out to not affect the output.

   This could be useful in researching and understanding market expectations of how the dynamics of the underlying's price movements differ from the BSM model.
3. Finding market mis-pricing. Time series of historical underlying prices could be used to value a range of options (especially across exercise prices) to find where the relationships between those market prices appear inconsistent with the historical evidence of dynamics of the underlying asset's price movements.

   This could provide trading opportunities or better risk control, depending on the outlook of the market participant. It could also be part of the process to develop better understanding of the true dynamics of underlying's price movements.

The implications of the results of this project for these potential uses are discussed below, point 1 in Section 7.2 and points 2 and 3 in Section 7.3.

7.2 Learning Complex Functions

The accuracy of the fit of the results to the BSM model results, given that the training time on a single CPU was only 1-2 minutes per model with 4 in a full ensemble, is encouraging. It implies that an ML model trained with sufficient quantities of data could be substituted for time-consuming numerical analysis methods such as Monte Carlo or finite difference when pricing new trades or valuing and producing risks for portfolios.

The main challenge may well be the amount of time required for the training process. Obtaining sufficient training data over a wide enough range of all the inputs to cover all future eventualities may be a much bigger problem than training the ML model. This is because of the time required to calculate the target training output for each datapoint.

7.3 Learning from Noisy Data

The results here are also encouraging given the training times allowed and that the later experiments included large levels of noise and a dummy variable, both of which the model needed to learn to ignore in order to find the true function to match.

This provides some confidence that an ML model could find the model that market participants are effectively creating by adjusting simpler models such as BSM to price options. There are two difficulties for this use:

1. The option prices being used as the target training output are likely to be noisy due to low frequency of trading in some options and difficulties in obtaining accurate prices for all of the underlyings, traded options and other training inputs at exactly the same time. However, the this project has shown that, given sufficient training datapoints, an ML model can find the underlying function despite this noise.
2. While many will be obvious, it is not known before the training exactly which data items will prove important for the market option prices. Consequently, in order to find all important inputs, it is likely that some will be included which do not in fact affect the outcome. However, this project has also shown that, again given sufficient training datapoints, an ML model will correctly avoid giving significant weight to a dummy variable with no impact on the training output.

8 Further Work

?? Build models of how the market participants are adjusting their models to take the issues of the BSM

model into account. Be careful not to effectively include the outputs as inputs, e.g. by using volatility swaps as a market measure of volatility. So difficulty is finding market data which is not based on traders manipulating BSM - we would get a manipulated BSM model back.

?? More fundamentally, build models that better match actual market dynamics of the movements in price of the underlying. LSTMs/Transformers required due to momentum effects in real markets?

?? Speed could be much improved by using a set of GPUs rather than a single CPU.

?? Model more than one output, e.g. price and delta. Would this require more hidden layers/units/regularisation/normalisation?

References

Akinyemi, D. (2015), 'Pricing european barrier options with partial differential equations'.

Black, F. & Scholes, M. (1973), 'The pricing of options and corporate liabilities', *Journal of political economy* **81**(3), 637.

Bouchaud, J.-P. & Potters, M. (1997), *Théorie des Risques Financiers*, Aléa Saclay.

Bühler, H., Gonon, L., Teichmann, J. & Wood, B. (2018), 'Deep hedging'.

Carverhill, A. P. & Cheuk, T. H. F. (2003), 'Alternative neural network approach for option pricing and hedging', *Risk Management* .

Curry, H. B. (1944), 'The method of steepest descent for non-linear minimization problems', *Quarterly of Applied Mathematics* **2**, 258–261.
URL: *https://www.ams.org/journals/qam/1944-02-03/S0033-569X-1944-10667-3/S0033-569X-1944-10667-3.pdf*

Derman, E. & Kani, I. (1994), 'Riding on a smile', *Risk* **7**.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C. & Garcia, R. (2009), 'Incorporating functional knowledge in neural networks', *Journal of Machine Learning Research* **10**, 1239–1262.

Glorot, X. & Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks, *in* Y. W. Teh & M. Titterington, eds, 'Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics', Vol. 9 of *Proceedings of Machine Learning Research*, PMLR, Chia Laguna Resort, Sardinia, Italy, pp. 249–256.
URL: *https://proceedings.mlr.press/v9/glorot10a.html*

He, K., Zhang, X., Ren, S. & Sun, J. (2015), 'Delving deep into rectifiers: Surpassing human-level performance on imagenet classification', *CoRR* **abs/1502.01852**.
URL: *http://arxiv.org/abs/1502.01852*

Hinton, G. (2012), 'Lecture 6a overview of mini-batch gradient descent'.
URL: *http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture$_s$lides$_l$ec6.pdf*

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012), 'Improving neural networks by preventing co-adaptation of feature detectors'.

Hull, J. C. (2018), *Options, Futures, and Other Derivatives, 9th Edition*, Pearson.

Hutchinson, J. M., Lo, A. W. & Poggio, T. (1994), 'A nonparametric approach to pricing and hedging derivative securities via learning networks', *The Journal of Finace* **49-3**, 851–889.

Lajbcygier, P. & Connor, J. T. (1997), 'Improved option pricing using artificial neural networks and bootstrap methods', *International journal of neural systems* **8 4**, 457–71.

LeCun, Y., Bottou, L., Orr, G. B., & Muller, K.-R. (2012), *Neural networks: Tricks of the trade*, Springer Berlin Heidelberg.

Loshchilov, I. & Hutter, F. (2019), 'Decoupled weight decay regularization'.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**, 115–133.

Merton, R. C. (1973), 'Theory of rational option pricing', *The Bell Journal of economics and management science* pp. 141–183.

Merton, R. C. (1976), 'Option pricing when underlying stock returns are discontinuous', *Journal of Financial Economics* **3**, 125–144.

Mokhtar, K. Z. & Mohamad-Saleh, J. (2013), 'An oil fraction neural sensor developed using electrical capacitance tomography sensor data', *Sensors (Basel, Switzerland)* **13**, 11385–406.

Pedamkar, P. (2023), 'Perceptron learning algorithm'.
URL: *https://www.educba.com/perceptron-learning-algorithm/*

Raberto, M. (1999), Processi stocastici e reti neurali nell-analisi di serie temporali finanziarie, PhD thesis, Università di Genova, Facoltà di scienze matematiche, fisiche e naturali.

Rosenblatt, F. (1958), 'The perceptron: a probabilistic model for information storage and organization in the brain', *Psychology Review* **65(6)**, 386–408.

Ruf, J. & Wang, W. (2020), 'Neural networks for option pricing and hedging: a literature review'.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), 'Learning internal representations by error propagation', *Parallel distributed processing: Explorations in the microstructure of cognition* **1**.
URL: *https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf*

Stoyanov, S. V., Rachev, S. T., Racheva-Iotova, B. & Fabozzi, F. J. (2011), 'Fat-tailed models for risk estimation', *Working Paper Series in Economics* **30**.
URL: *https://www.econstor.eu/obitstream/10419/45631/-1/659400324.pdf*

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. & Recht, B. (2017), The marginal value of adaptive gradient methods in machine learning, *in* 'Advances in Neural Information Processing Systems', Vol. 30, Curran Associates, Inc.
URL: *https://proceedings.neurips.cc/paper_files/paper/−2017/file/81b3833e2504647f9d794f7d7b9bf341 − Paper.pdf*

Yashwanth, N. (2020), 'Understanding gradient descent'.
URL: *https://medium.com/analytics-vidhya/gradient-descent-and-beyond-ef5cbcc4d83e*