Machine Learning Tools for Option Pricing

The Non-Parametric Modelling Alternative

Ed Harvey

August 28, 2023

There has been a massive expansion of the markets for derivatives of financial instruments in the last few decades, much of it following the work of Black, Scholes and Merton in the early 1970s. Their model and its successors are the methods underlying market participants' pricing and the risk management of their portfolios of trades. However, there are known issues caused by the difficulty of modelling the full complexity of human behaviour, particularly since any chosen model needs to be simple to use in a fast-moving trading environment. Many of these issues can lead to risks being mismanaged, causing unexpected gains and losses, especially at times of market upheaval such as during the credit crisis of 2007-2009 or the pandemic of 2020-2022.

Employing known mathematical distributions with certain parameters to be discovered from current market prices is often referred to as using parametric models. An alternative can be to use non-parametric models for which machine learning tools are used to create a model from all relevant market data. In this case, the machine learning algorithm is not supplied with any pre-conceived ideas of what distribution or parameters are required as it will optimise its use of the market data to obtain the required pricing or risk measures.

There is at least a thirty year history of research into non-parametric models for use in financial markets, but they have not so far been widely used by major market participants for daily pricing and risk management. However, the continued increase in available computing power and the improvement and streamlining of machine learning algorithms may make this a more attractive option in the future.

This project uses simulations of market data together with parametric model pricing, and finds that artificial neural networks can efficiently learn to match a pricing function, even when, as will be the case when obtaining real-world market data, the training data available is noisy and may included non-relevant data. Consequently, it provides evidence that non-parametric models can be a valid alternative to the parametric models in use today.

Contents

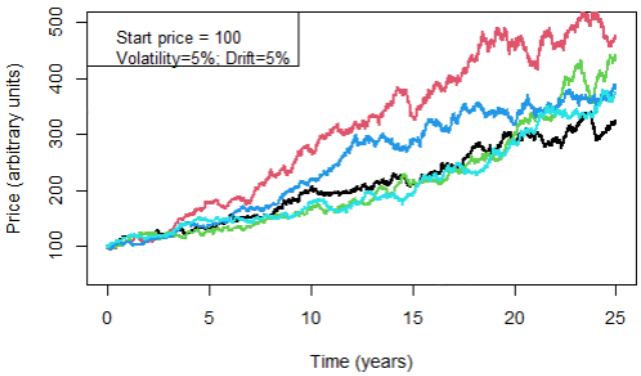
1 Introduction

There has been interest in, and research into, models to help value option contracts and manage their risks for over a century. Work in the early 1900s, for example Bachelier 1900, and Bronzin in 1908 (as described in Hafner and Zimmermann 2009), developed usable methods to do this long before the more famous publication in 1973 of the Black-Scholes-Merton (BSM) model in Black and Scholes 1973 and Merton 1973. While obtained using different methodologies and assumptions, the results of the earlier works were similar to BSM. However, they were largely forgotten until much later, possibly because they were before their time in terms of practical application.

By the 1970s, increased global use of financial markets together with increased use of computers and improved international communications, had increased the need for models, improved the ability to make use of them, and made some of the required efficient markets assumptions a little closer to reality. In addition, the gradual collapse of the Breton Woods fixed exchange rate regime between 1968 and 1973 created a strong demand for hedging foreign exchange risk which had not previously been so important. The timing of the publication of the BSM model meant that it consequently became an important landmark in the development of financial markets.

The BSM model provides simple formulae for valuing and hedging standard option contracts, but does so by employing strict assumptions. Firstly, that the expected future path of the price of the underlying asset can be described by Geometric Brownian Motion (GBM), that is, normal distribution of returns with constant volatility (see Figure 1 for examples). Secondly, that the nature of the market for both the underlying and cash instruments is both frictionless and efficient.

Figure 1: Examples of simulated GBM paths



Major benefits of the BSM model, and possibly the main reasons for its continued use in financial markets to this day, include 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. This improved understanding of their positions and how to reduce their risks, allowing 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 major issues are listed below.

- 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 equity 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 and 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.
- The efficient markets assumption means that all current and historic information about the underlying is assumed to be available and fully included in its current market price. Consequently, the normal distribution of the returns is assumed to be memoryless. This is at best contested, for example in Lo 1999, as many researchers believe that past price moves can be shown to cause both short and long-term effects on current prices.
- The frictionless markets assumption implies that it is possible to carry out costless trading in unlimited quantities of the underlying asset to hedge the risk on the option. In practice, this is not true, meaning that assumptions that any difference to theoretical price can be arbitraged away are false.
- The frictionless markets assumption also implies that it is possible to carry out continuous trading in the underlying asset to hedge the risk on the option. This is also not true 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 BSM. One 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 behave in reality. For example, only 3 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 and Potters 1997 used processes creating 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 therefore also share, to a greater or lesser extent, the main problems with using a parametric model as listed below.

- 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 periodically trained on new data and therefore adjust automatically to any changing market dynamics.

The combination of improvements in computing power, and the modern Artificial Neural Network (ANN) methods which are now used for many machine-learning tasks, may make non-parametric models an increasingly viable alternative, and they are the focus of this project. In particular, this project aims to optimise the setup for an ANN 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 overly long 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 5 but before that, in order to explain some of the terms used, Section 3 gives a background to the derivatives being reviewed in this project and Section 4 describes ANNs.

2 Abbreviations

Table 1: Abbreviations

ANN	Artificial Neural Network
ATM	At-The-Money
BSM	Black, Scholes and Merton
CPU	Central Processing Unit
GBM	Geometric Brownian Motion
GD	Gradient Descent
GPU	Graphics Processing Unit
ITM	In-The-Money
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
OTM	Out-of-The-Money
PDE	Partial Differential Equation
SGD	Stochastic Gradient Descent
SGDM	SGD with Momentum

3 Derivatives

This section briefly describes the most important aspects of derivatives markets for the purposes of this project. Many textbooks cover this subject in much more detail, for example Hull 2018.

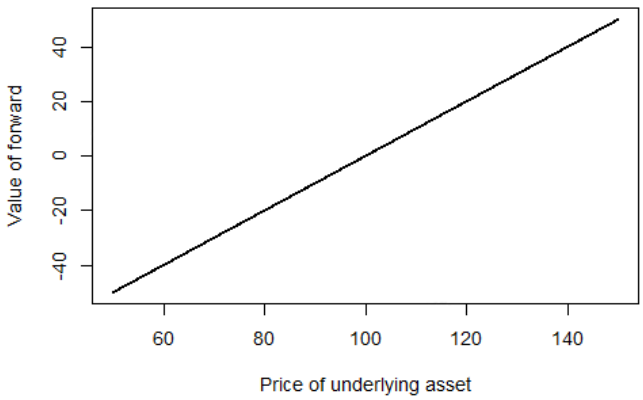
3.1 Overview

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 very many types of derivative traded in financial markets, of which, 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. When these contracts are agreed for a date other than the standard settlement date for that underlying, they are often known as forwards. 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 set of 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.

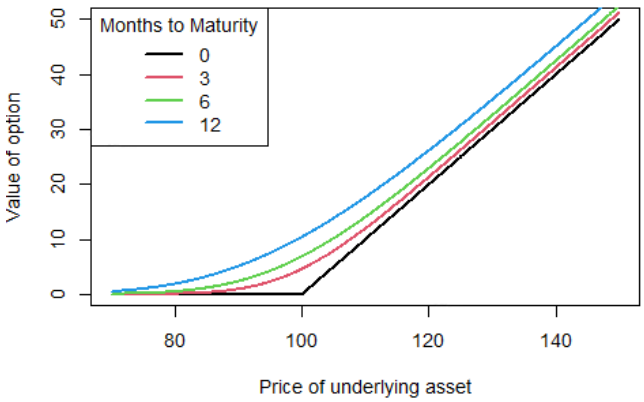
However, this project has concentrated on options as the most commonly traded example of derivatives with non-linear payoffs which cause additional complexity in valuing them.

3.2 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. 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. Market participants often use the terms in-the-money (ITM), out-of-the-money (OTM) and at-the-money (ATM) to refer to an option that, if it were to expire today, would respectively have a positive value, would have no value, or would be on the point between the two.

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 6.1.2 for details of the formulae for the BSM option value. Prior to maturity, there is always additional future value to a call (though not always for puts) over and above that of the payoff, due to the unlimited gains but limited losses. An example, for a call option with a strike price of 100, of the 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.

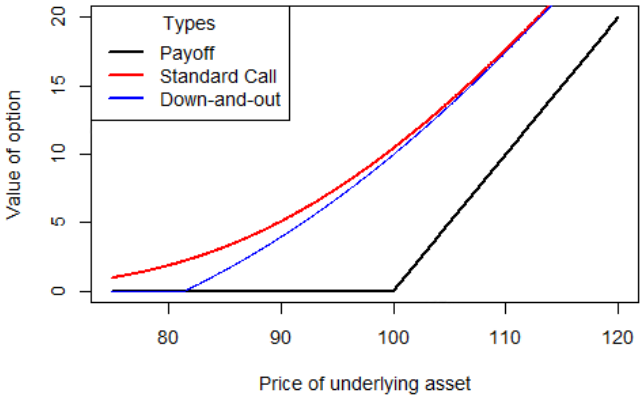
There are a number of other styles of option. Bermudans can be exercised on a limited list of pre-determined dates while Asians have a payoff which depends on the average price of the underlying over a period of time rather than at a specific maturity date. Basket options which have a payoff based on 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). The barrier may also move during the life of the trade, for example to start lower and increase 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 6.1.3 for details of the formulae for the BSM barrier option value. An example, with the values 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.

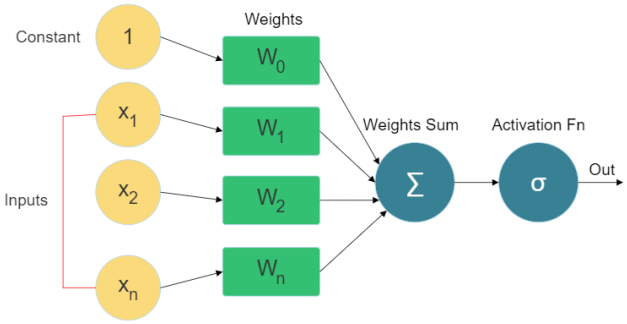
Another important operation in options markets is to be able to find the volatility that market participants are using to price their options. If a market price for an option exists for which the model being used and the other inputs are known, then the volatility that has been used to create the price can be calculated. This is often referred to as the implied volatility.

4 Artificial Neural Networks

The use of ANNs has expanded greatly in recent decades and they can be used in many forms at varying levels of complexity. The decision on which one to use is dependent on the problem being solved. For example, a standard multi-layer perceptron (MLP) may be used to produce predictions from a single datapoint. However, if other datapoints may also be important, for example, previous values in a time-series or previous words in a sentence, a long short-term memory network (LSTM) may be used. In order to understand more complex ANNs, it is first necessary to understand a single perceptron unit.

4.1 Single Perceptron

Figure 5: Single Perceptron (author's figure)

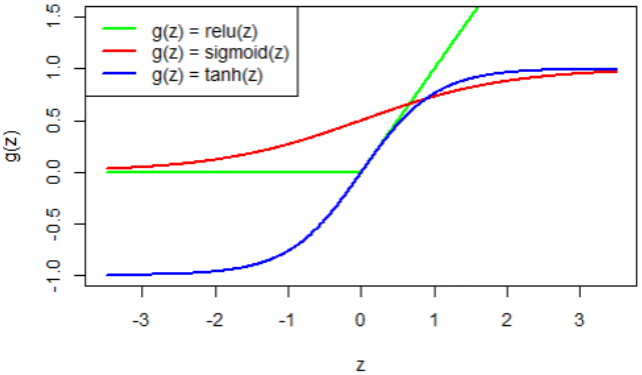


A single perceptron (see Figure 5) is a simplified artificial version of a neuron in a human brain. It was first proposed by McCulloch and Pitts 1943 and the first implementation was reported in Rosenblatt 1958. Given a vector of inputs $(1, x_1, ..., x_n)$, it applies weights $(w_0, w_1, ..., w_n)$ 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 $\hat{y} = g(\underline{x} \cdot \underline{w} + b)$.

Given a training dataset of input vectors matched with an equal number of 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 initialiser algorithms) to minimise some loss function comparing $(y_1, ..., y_m)$ with $(\hat{y}_1, ..., \hat{y}_m)$.

There are a number of commonly-used choices for the activation function $g(z)$ (illustrated in Figure 6). For example, $\sigma(z) = \frac{1}{1+e^{-z}}$ gives a result between 0 and 1, and can be used to represent a probability in a classification problem. $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$ has a similar shape but gives a result between -1 and 1. $\text{ReLU}(z) = \max(z, 0)$ gives a linearly increasing output for positive values of the input only. ReLU and its first derivative are faster to calculate and therefore generally expected to converge faster (for example, see Alex Krizhevsky and Hinton 2012).

Figure 6: Outputs of activation functions



Common loss functions are Mean Squared Error, MSE $= \sum_{j=1}^{m}(y_j - \hat{y}_j)^2$ or Mean Absolute Error, MAE $=$

$\sum_{j=1}^{m} |y_j - \hat{y}_j|$. MSE works better if there are not too many extreme outliers in the training data, for which the squared term would accentuate the effect on the model.

Common initialiser algorithms are 'glorot' as proposed in Glorot and 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 when using the trained model to predict the correct solution for new input data which was not included in the training dataset. This is often known as the trade-off between bias (high bias means that there is poor accuracy of fit to the training data) and variance (high variance means that there is high variance between the predictions by models trained on different datasets and therefore that 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 case, 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.

4.2 Loss Function Minimisation

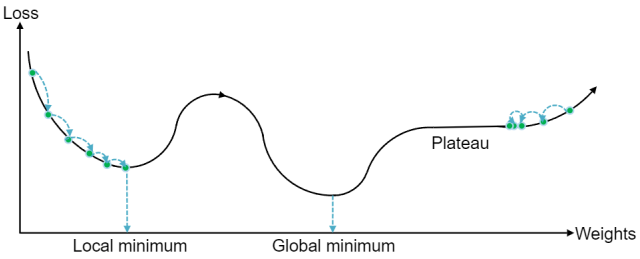
Gradient Descent (GD) is the underlying concept behind the algorithms which aim to minimise the loss function and was first studied for this purpose in Curry 1944. At each step of training the model, each weight w_i is adjusted by the negative of the partial derivative of the model's output with respect 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 and 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 Stochastic Gradient Descent (SGD) with momentum.

A second issue is that the loss function and its derivatives may take a long time to calculate, leading to long training times. SGD can be used to 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.

Long training times can also be exacerbated if the whole dataset and all the errors were loaded 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 7: Local Minimum Problem (author's figure)

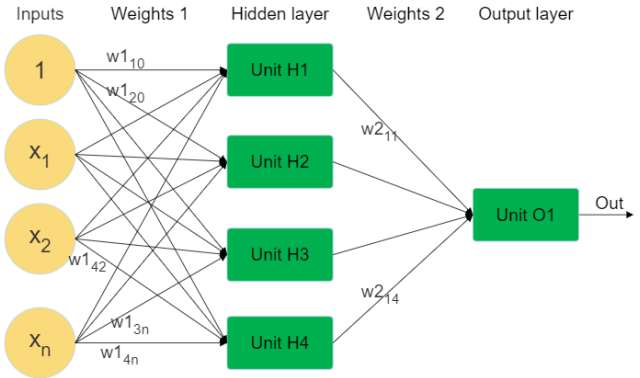


A third issue is that there may be multiple local minima or plateaus corresponding to different sets of weights (see Figure 7). It is then difficult for the algorithm to find the global minimum. A useful effect of the increased variance in the adjustments to the weights resulting from SGD, is that the 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.

4.3 Multi-Layer Perceptron

Figure 8: Diagram of a Multi-Layer Perceptron (author's figure)



A single perceptron (or single layer of perceptrons) can only learn linear relationships. 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 8 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 the experiments in this project, there is only one output in the output layer, representing the ML model calculation of the value of the option.

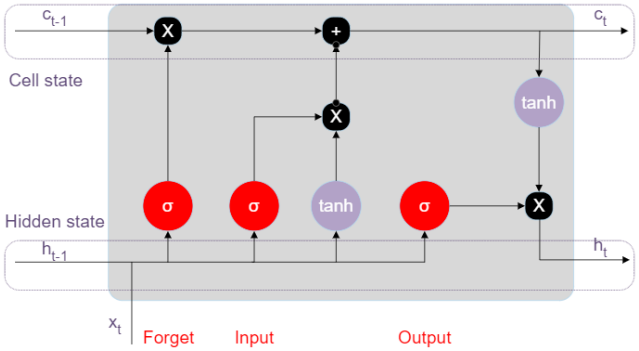
The loss minimisation in the training process requires back-propagation of adjustments from the outputs through each of the hidden layers in turn as shown experimentally in Rumelhart, Hinton, and Williams 1986, but the issues discussed in Section 4.2 still apply.

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. Regularisation, as explained earlier, can be an important technique which is used within each hidden layer. Using dropouts is a technique introduced in Hinton et al. 2012, which is often used to reduce over-fitting. A randomly-chosen proportion of the units in each layer is dropped during each round of the training process, which 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 in order to produce smoother gradients for the weights update in each training batch. This can be normalised for each batch or for the whole layer.

4.4 Long Short-Term Memory

Figure 9: Diagram of a Long Short-Term Memory unit (author's figure)



An MLP only uses the current datapoint to predict the required output. In many circumstances in financial markets, it may also be important to use information from datapoints at previous times. A network of LSTM units (as shown in Figure 9) aims to do this by including three sets of inputs instead of only the one from the current datapoint as used in an MLP. The additional inputs are the previous time step's output and a cell state which is able to hold data created from all previous time steps.

The process will not be described in detail here as LSTM networks are only suggested for future work rather than being used for the experiments in this project. However, the basic steps are listed below:

- A forget gate takes the output data from the previous step and from the current input data to determine what to forget from the cell state.
- An input gate also takes the output data from the previous step and from the current input data to determine what to add to the cell state.
- An output gate also takes the output data from the previous step and from the current input data to determine what to send on to the output of the cell.
- The output from the cell is determined by data from the cell state, after the forget and input gates have acted on it, and from the output gate.

Each of these steps requires a set of weights and a bias for each of the inputs to it. There are therefore many times more weights per unit to be adjusted during the learning process than there are for an MLP.

5 Literature Review of ANNs for Derivative Pricing and Hedging

ANNs 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 which is the most frequent, and is the focus here.
- 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 and 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, Lo, and Poggio 1994 trained ANNs on 2 tasks. For the first, it was trained using inputs of a series of underlying asset prices simulated using GBM with constant drift and volatility, together with a range of values for the exercise prices and the times to maturity for each of the series of prices. They used this to learn to match the output of the Black Scholes model for the same inputs. For the second, it was trained, using inputs of historical underlying asset prices, to match 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.

A development of the second experiment, as proposed by Lajbcygier and Connor 1997, could be to use the BSM model as a base together with an ANN 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 ANNs to match the output of the Bouchaud-Potters model (as explained in Bouchaud and 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 ANNs to learn to produce the results of more complex models.

An additional possible adjustment in the target of the ANN is to model hedging strategies rather than prices of options, as first proposed in Carverhill and 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 ANN 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 pointed out that the value of a standard call option is monotonically increasing with underlying price and with time-to-maturity and is also 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.

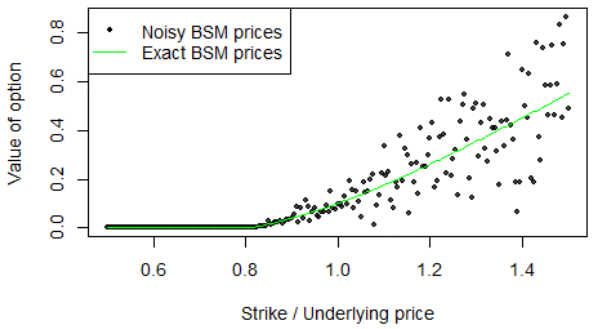
A further complexity for which ANNs may provide good answers is hedging portfolios of derivatives outside the theoretical world of frictionless markets which is very hard to do using parametric models. Bühler et al. 2018 proposed a framework for pricing and hedging, without observing option prices, and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

6 Experimental Design

The experiments carried out for this project have 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. Seven 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 while the risk-free interest rate and volatility were assumed to be constant. The training output was the 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 (as a proportion of the BSM price with mean of 0.0 and 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 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). An example of the resulting BSM prices plotted against Strike/Underlying price is shown in Figure 10.

Figure 10: Noisy BSM prices



6. As above, but with an additional dummy variable that the model should ignore - if it does not, it will over-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 but by testing an additional issue

arising from using real-world data where it is likely that less relevant inputs are included in the training data.

7. As above, but with the number of datapoints in the training data reduced from 170,000 depending on the setup to a maximum of just over 6,000 and the standard deviation of the noise reduced back down to 0.1. This was intended to test those situations where large quantities of training data were difficult or too time-consuming to obtain but the noise was not extreme - possibly the closes to a real-world situation when using real-world data.

8. As above, but with the standard deviation of the noise increased back to 0.4. This was intended to test the results of using ML models in the most unpromising of circumstances.

6.1 Target Solutions for ML Model

6.1.1 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). 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, $\frac{S(t+\epsilon)-S(t)}{S(t)}$ is normally distributed with mean $= \mu$ and constant standard deviation $= \sigma$. 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) resulting from owning 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, with zero fees or other costs, 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.

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

6.1.2 European Call

Experiments 1 and 2 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, with the result that the expected rate of return on the asset μ is not relevant to the value of the option on it. The partial differential equation (PDE) below for the option value (V) then follows.

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

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:

$$C(S,t) = N(d_1)S - N(d_2)Ke^{-r(T-t)} \qquad (2)$$

$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left(ln(\frac{S}{K}) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right)$
$d_2 = d_1 - \sigma\sqrt{T-t}$
$N(x) =$ Cumulative standard normal distribution up to x.

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 $C(S,t)$:

$$C(S,t) = K \cdot C(s,\tau) = K\left(N(d_1)s - N(d_2)e^{-r\tau}\right) \qquad (3)$$

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

6.1.3 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 PDE similar to Equation (1) but with an additional boundary condition for the barrier (from Akinyemi 2015). Using the notation $b =$ barrier level relative to strike price at maturity and $\eta =$ decay rate of b with increasing τ such that $B =$ absolute barrier level $= bKe^{-\eta\tau}$, the additional barrier condition is:

- Barrier: $C(B,t) = 0$

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

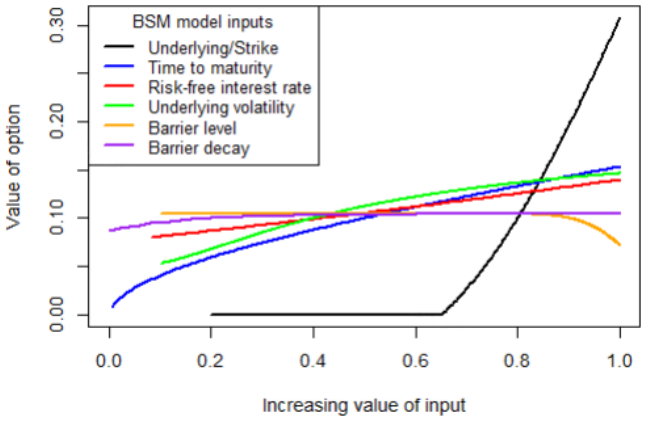
$$C_{DO}(S,t) = C(S,t) - \left(\frac{S}{B}\right)^{\left(1-\frac{2}{\sigma^2}(r-\eta)\right)} C\left(\frac{B^2}{S}, t\right) \quad (4)$$

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

$$C_{DO}(S,t) = K \cdot C_{DO}(s,\tau) \quad (5)$$

A property of the above equations which has been used in some of the research papers discussed in Section 5, is that European call option prices, whether using a barrier or not, are monotonically increasing for all inputs except for b for which the prices are monotonically decreasing. For illustration, prices for a barrier option are shown in Figure 11, in which the prices for a barrier option with base inputs of $S = 1, K = 1, T - t = 1, r = 0.05, \sigma = 0.2, b = 0.9, \eta = 0.1$ are plotted against each of the inputs being varied in turn.

Figure 11: BSM model barrier option price with increasing values for each of the inputs



6.2 R Implementation of ANNs

All analyses were performed using R Statistical Software (v4.2.2; R Core Team 2022). Code files are available at the author's GitHub repository (Harvey 2023).

Open-source packages are available, which enable the Python implementations of TensorFlow (v2.11.0; Allaire and Tang 2022) and Keras (v2.11.1; Allaire and Chollet 2023) to be used in the R computing language. They 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 new set of input data.

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 using the 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 testing 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)
```

6.3 Data

For each training run to create a single model, a simulation was run of daily asset prices over a given time period. For each date, a series of datapoints with a range of values for each of the training inputs relevant to the experiment were created. 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 were taken. These were set at 1 month (20 day) intervals from the start date of the simulation up to half a year (120 days) beyond the final simulation date, with maturities equal to or less than the current data of the simulation being removed. For Experiment 7, the frequency was reduced to 3 month (60 day) intervals.

- Exercise prices. A random range of exercise prices for reference European call options were taken, 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 Experiment 1, r and σ were set at 0.05 and 0.2 respectively. For later experiments, r and σ were also given a random range, based on uniform distributions of 0.01 - 0.12 and 0.05 - 0.5 respectively.
- Barrier contract terms. For Experiment 3 and later, the additional contract parameters for the barrier options, b and η, were added with random ranges from uniform distributions 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 Experiment 6 and later, a dummy variable was included in the training data with a range between 1 and 5, although since all training inputs are normalised before being used in the model and this input is not included in the calculation of the training output below, the range of inputs should not affect the modelling.
- For each of the uniform distributions, and for each simulation data and option maturity combination, 20 draws were taken for standard options and 31 for barrier options to arrive at approximately 170,000 training datapoints. For Experiments 7 and 8, this was reduced to 3 draws.

Output:

- Option prices. Prices for the options were based on the BSM solutions described in Section 6.1 using the training inputs described above.
- For Experiment 4 and later, 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 in all cases, but standard deviations of 0.1 for Experiments 4 and 7, and 0.4 for the Experiments 5, 6 and 8.

For testing of the effects of using ensembles of models (see Section 6.4.1), and for reviewing tracking errors (see Section 6.4.2), 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.

6.4 Evaluation

6.4.1 Hyper-Parameter Optimisation

It can be seen from Section 4 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 learned during the training process to enable the model to reduce losses on the training 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 setup of the experiment).
- The number of hidden layers.
- The loss function.
- The activation function for each layer.
- The optimisation algorithm to employ in minimising the loss function.
- The regularisation, 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 training datapoints for which to test for losses and calculate gradients for each update of the weights.
- Number of models to include in an ensemble.

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

For Experiments 1 to 3, 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 Experiments 4 and later 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 hyper-parameter tests 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 hyperparameters. 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 obtain quick answers at slightly lower accuracy, the model with the lowest validation losses 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 the 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.

6.4.2 Tracking Errors

As explained in Section 6.1.1, 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 rebalancing 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 (known as the Delta as explained in Section 3.2). If this process is performed continuously throughout the life of the option, then, including the cash from the option premium and the payoff at maturity, 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 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 the experiments carried out in Hutchinson, Lo, and Poggio 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:

$$V(t) = V_C(t) + V_S(t) + V_b(t) \tag{6}$$

The notation is then used that the price 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 $S(t)\Delta(t)$ of the underlying. At $t = 0$, the option seller therefore receives cash for the option premium and pays out cash for the purchase of the required amount of the underlying.

Thus, since these transactions were carried out at fair value due to the market being efficient and frictionless, and following Equation (6), the initial composition of the portfolio at time 0 for a seller of the option is:

$$V_C(0) = -C(0); V_S(0) = S(0)\Delta(0)$$

$$V_b(t) = -\left(V_S(0) + V_C(0)\right)$$

$$\therefore V(0) = V_C(0) + V_S(0) + V_b(0) = 0 \tag{7}$$

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 δt = 1 day, the composition of the portfolio at time t becomes:

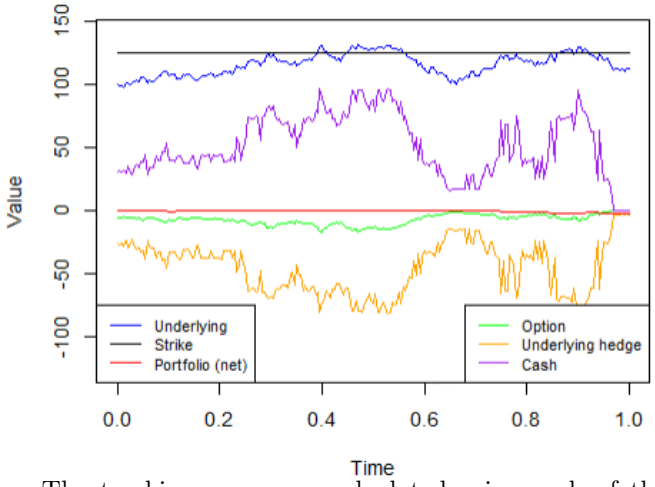
$$V_C(t) = -C(t); V_S(t) = S(t)\Delta(t)$$

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

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 12 using the preferred ML model with the lowest loss from Experiment 2 for an option with a strike price of 125 and a maturity at time 0 of 1 year. The lines representing the values of the option, the hedge in the underlying, 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 12: 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.

6.4.3 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 project 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 predict 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 the behaviour or some feature of the market changes.
6. The tracking errors using the ML model and BSM, as described in Section 6.4.2.

When evaluating the time taken to run a process, the time elapsed proved inconsistent, presumably depending on the time of day and the number of other users sharing the University lab computers. The hyper-parameter testing was carried out using a time elapsed method (`sys.time()` in the r code). This means that, while the times shown in the detailed results are generally comparable between the tests across a range of hyper-parameters which were run at the same time, they are not closely comparable between tests done on different hyper-parameters.

The final tests on the preferred model were carried out using the processor time method (`proc.time()` in the r code). They can show very different times than the elapsed time recorded for earlier tests, but they are more closely comparable to one another despite being run at different times. However, there are still a number of reasons why they can be inconsistent, for example, the competition for CPU resources at time times of heavier use and the use of a virtual environment.

7 Results

Details of the results are included in the author's GitHub repository Harvey 2023. There is an Excel file for each experiment named according to the format results-store-i.xlsx with the number of the experiment replacing the 'i'. Descriptions of the meaning of each field name are shown in the Notes sheet in results-store-7.xlsx. A shiny app has also been added to allow a user to request prices and illustrative charts from the models resulting from each of the experiments. This is also stored in the GitHub repository, in Shiny_MLOption Pricing - a user would need to copy the folder and all it's subfolders onto their local computer and run the app from RStudio, having followed the install instructions for keras in Allaire and Tang 2022).

The results are summarised in this Section.

7.1 Hyper-Parameter Testing

Each of the hyper-parameters, and the effects of adjusting them for each of the experiments listed in Section 6, are summarised below in turn. In Experiment 1, they were initially set at the values shown in Table 2. 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 (10 for Experiments 7 and 8) as described in Section 6.3.

Table 2: Initial Hyper-Parameter settings

Hyper-Parameter	Initial
Activation	ReLU
L^2 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, as stated as being expected in Section 4.1, the best option in all 8 of the experiments. However, for Experiment 5 when there was greater noise in the training output but no dummy variable, the difference in validation losses between using ReLU and tanh was down from several times lower to 20% lower.

L^2 regularisation

As expected, any amount of regularisation increased losses significantly for Experiments 1 to 3 with no noise or dummy variable in the training output. However, for Experiments 5, 6 and 7/8, a small L^2 regularisation factor of 0.0001, 0.0002 and 0.002 respectively was optimal. This reduced losses by over a third compared to using zero when the larger noise element was introduced, by a little more when the dummy variable was added to the training inputs, and by 60% when the number of training datapoints was reduced.

Optimisation

For Experiments 1 and 2 valuing standard options, 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). For Experiments three to six with more training inputs and/or noise in the training output, adamax became the best-performing optimiser and SGD with momentum was a lot less successful in Experiments 5 and 6 with increased noise. However, SGD with momentum was again marginally most successful when the number of training datapoints was reduced in Experiments 7 and 8. It was possibly surprising to see that rmsprop performed very poorly for the simpler experiments given its use as a default in the Keras package although the difference in performance was much reduced with the greater noise element from Experiment 5 onward.

Dropout

As expected, since dropout is another regularisation technique, it showed a similar direction in results to the L^2 regularisation except that it was still marginally better to use zero dropout even for Experiments 5 and later. 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 exceptions are Experiments 6 and 7 with the dummy training input where 128 and 64 have lower losses than 32. The levels chosen for the final preferred ML models have been 64 throughout - 32 sometimes gives slightly lower losses but the training time was 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. There was little reduction in losses from increasing the number of layers past 2, even when the noise element was added. Using 1 layer resulted in 50%+ greater losses than using 2 layers for all except Experiment 6 for which the difference was only 25%. 2 layers was chosen for the preferred ML model for Experiments 1 to 7.

Experiment 8 with the reduced number of training datapoints and high noise was an exception. In this case, the training time was not increased greatly by using more layers but the losses were reduced (by 40% between using 1 and 4 layers and 20% between using 2 and 4 layers) so 4 layers was chosen for the preferred model.

Units per layer

As would be expected, increasing the number of units per layer generally reduced losses while increasing training and testing times. However, this effect was reduced once noise was added to the training inputs. For Experiments 1 to 3 with no noise element, the tests implied that, for a given total number of units with roughly equal training and testing times, losses were reduced by using an equal number of units in each of the layers rather than using more in one layer than the other. For Experiments 4 to 8, 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 project, 200 units in each of the layers has been chosen as giving the best balance for experimental purposes except for the Experiment 5 with 200/100, Experiments 6 and 7 with 100/100 and Experiment 8 with 100/100/100/100 for its four layers.

Normalisation

Using either of batch or 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 Experiments 5 to 7.

Initialiser

Using glorot_uniform and glorot_normal generally resulted in only small differences in losses. With only 4 training runs, it is likely to 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 Experiments 5 to 8 with a large noise element) since He et al. 2015 found that these produce better results for ANNs 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 Experiments 1 to 6. 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. In Experiment 8 with a reduced number of training datapoints but still with higher noise, there was no advantage gained from increasing the number of epochs from 25 to 50.

Model Ensembles

The mean loss across 10 testing runs for each of the seven experiments was calculated for each of the mean or best method of using the ensemble of models, and for the four different training conditions listed below, resulting in eight outcomes for each experiment.

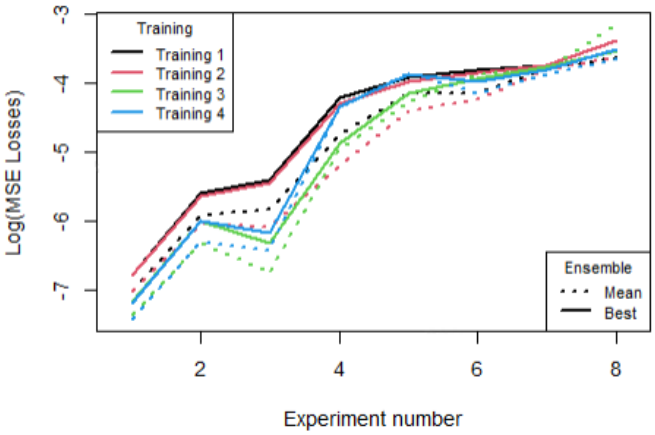
1. The baseline situation which, for Experiments 1 to 6, is 4 training runs (and therefore 4 models in each ensemble), just under 180,000 items of training data, and 10 epochs of training. Because of the reduced number of training datapoints and the resulting much faster training times per epoch, the baseline for Experiments 7 and 8 is 10 training runs, just over 6,000 items of training data and 25 epochs of training.
2. As for 1, but a fourfold increase in the number of training runs to 16 (40 for Experiments 7 and 8).
3. As for 1, but an approximately fourfold increase in the number of items of training data to around 710,000 (24,000 for Experiments 7 and 8).
4. As for 1, but a fourfold increase in the number of epochs of training to 40 (100 for Experiments 7 and 8).

The aim of these calculations was to determine, given that a performance improvement is required and training time is to be extended to achieve this, which is the best option for performance. Table 3 shows the results with each experiment in a column and the rows denoted by the number of the training condition as listed above and 'B' or 'M' representing the best or mean methods respectively. Figure 13 shows the same information in a chart except that the log (base 10) of the losses is used to allow them to be compared.

Table 3: MSE Losses (1e-06) of ensembles of models per experiment

	1	2	3	4	5	6	7	8
1M	0.10	1.21	1.49	18.3	70.2	70.3	159.9	223.7
1B	0.16	2.51	3.97	61.5	123.8	154.9	182.0	415.5
2M	0.09	0.16	0.85	6.4	39.0	56.9	155.6	239.5
2B	0.16	2.35	3.64	49.3	106.4	142.2	182.5	415.5
3M	0.04	0.49	0.19	11.2	52.0	132.4	156.0	688.9
3B	0.07	1.03	0.49	13.6	70.6	113.4	174.0	294.4
4M	0.04	0.52	0.39	45.6	139.0	69.3	135.0	216.1
4B	0.07	1.03	0.69	45.9	130.9	106.2	153.8	302.0

Figure 13: 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:

- 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 close-to-linear relationship between the option price and the underlying price, together with the shorter range of underlying prices for which the option has been modelled (the underlying price cannot be below the barrier).
- There are large percentage increases in losses with increasing levels of noise in the training output between Experiments 3, 4 and 5 and again with the reduction in training datapoints in Experiment 8.
- There is only a small increase in losses 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. However, while training conditions employing 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.

- For Experiments 1 to 5, if lower losses and improved accuracy are required and extra training time can be used to achieve this, then increasing the quantity of training data is the best strategy, assuming that the additional data is available. If it is not, then increasing the number of epochs of training cycles is second best (and best for Experiments 6 to 8) while increasing the number of training runs to create a larger selection of models in the ensemble is the least successful strategy.

- Given that the motivation for Experiments 7 and 8 were to mimic a situation where market data for training purposes was not available in large quantities, an extra test was run to see if increasing the number of epochs would further reduce losses. The results showed little clear improvement beyond 25 epochs.

- The trends in which strategy to employ generally become less clear with the later experiments as the learning task is made more difficult by successively increasing the noise, introducing the dummy variable and reducing the number of training datapoints.

7.2 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 or processor time might be limited, 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 more important 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 made to obtain 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. The final chosen parameters for each experiment are shown in Table 4.

7.3 Performance Analysis

Comparable performance measures for all of the experiments are shown in Table 5. The option to quadruple the number of items of training data in each training dataset was used in all cases except Experiments 7 and 8, where the concept was that there was a limit on training data availability and the alternative option of increasing the number of epochs did not significantly reduce losses. All testing results were averaged over 10 testing runs, each of which comprise 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 each testing dataset.

The results clearly show the reduction in accuracy as the complexity of training the models increases due to increasing the number of training inputs from 2 to 4 to 6, adding the noise element to the training outputs for 4 and later , adding a dummy variable as a seventh training input with zero effect on the training output for 6 and later, and finally reducing the number of training datapoints for 7 and 8. However, even Experiment 8 shows R-squared of 99.7% and an average tracking error of 1.80 (1.45 higher than BSM's tracking error), which 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 in Figure 14 illustrate the way in which the errors increase with complexity of the experiment. Experiment 2 is shown as an example of a noiseless experiment and then each of Experiments 4 to 8. The charts 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 charts for Experiments 4 to 8 errors show two clear trends, both of which are confirmed if alternative testing runs are reviewed. One trend is for increased errors, more often to over-estimate the true price for Experiment 6, at very short times to maturity.

The second trend, in the case of Experiment 6, 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), to under-estimate when the underlying is a little lower than the strike, and then increasingly over-estimate when the underlying is higher than the strike price (and the option price consequently should be very close to the underlying price less the strike price). The charts for Experiments 4 and 5, and to a lesser extent 7 and 8, show similar trends to 6, except the trends of the errors are in the opposite direction, especially for high underlying prices.

An important factor to bear in mind when using ANNs, is that they may not predict prices well when extrapolating outside the training data ranges. To illustrate this, testing of Experiment 2 was re-run but with the strike price randomly created for 0.25 to 4 times the underlying price when the training range was 0.5 to 2. Figure 14 shows the resulting errors against the value of the underlying relative to the strike. This shows clearly how the pricing errors increase outside the range of the training data. The MSE, R^2 and mean error data are also shown in Table 5 with the header "2OR".

Table 4: Final chosen parameters for Experiments 1 to 7

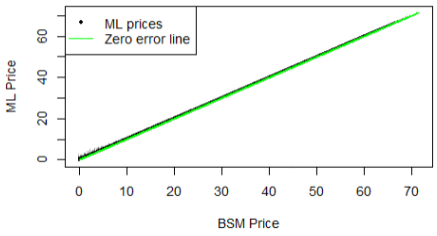
	1	2	3	4	5	6	7	8
Exp Detail								
Input: S/K, T-t	✓	✓	✓	✓	✓	✓	✓	✓
Input: r, σ		✓	✓	✓	✓	✓	✓	✓
Input: b, η			✓	✓	✓	✓	✓	✓
Input: dummy						✓	✓	✓
Noise	0	0	0	0.1	0.4	0.4	0.1	0.4
Training datapoints	170,000	170,000	170,000	170,000	170,000	170,000	6,000	6,000
Hyper-parameter								
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
L^2 Regularisation	Zero	Zero	Zero	Zero	0.0001	0.0002	0.002	0.002
Optimisation	sgdm0.99	sgdm0.995	adamax	adamax	adamax	adamax	sgdm0.95	sgdm0.925
Dropout	Zero	Zero	Zero	Zero	Zero	Zero	Zero	Zero
Batch size	64	64	64	64	64	64	64	64
Number of layers	2	2	2	2	2	2	2	4
Units per layer	200/200	200/200	200/200	200/200	200/100	100/100	100/100	All 100
Normalisation	None	None	None	None	None	None	None	None
Initialiser (Glorot)	norm	unif	unif	unif	unif	norm	norm	norm
Number of epochs	10	10	10	10	10	10	25	25
Mean/Best of ensemble	Best	Best	Best	Best	Best	Best	Best	Best
Training condition	3	3	3	3	3	3	1	1

Table 5: Performance measures for Experiments 1 to 8 (OR refers to use of out-of-range testing data

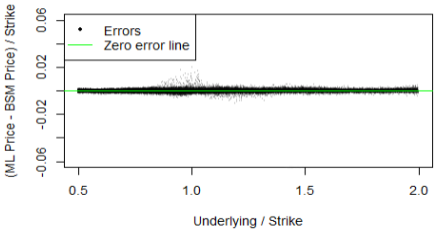
	1	2	2OR	3	4	5	6	7	8
Exp Detail									
Input: S/K, T-t	✓	✓	✓	✓	✓	✓	✓	✓	✓
Input: r, σ		✓	✓	✓	✓	✓	✓	✓	✓
Input: b, η				✓	✓	✓	✓	✓	✓
Input: dummy							✓	✓	✓
Noise	0	0	0	0	0.1	0.4	0.4	0.1	0.4
Training datapoints	170,000	170,000	170,000	170,000	170,000	170,000	170,000	6,000	6,000
Measure									
MSE (1e-06)	0.07	1.03	23.68	0.49	14.20	86.20	106.00	182.47	415.47
Mean R^2 (%)	100.000	99.995	99.915	99.999	99.969	99.827	99.673	99.438	98.993
Mean error	-0.00	-0.10	-0.14	-0.01	-0.11	0.18	-0.09	0.10	0.28
ML Tracking error	0.47	1.38	N/A	0.94	1.30	1.03	1.80	3.56	2.39
BSM Tracking error	0.23	0.60	N/A	0.35	0.35	0.35	0.35	0.35	0.35
Training time (s)	402	513	N/A	357	416	441	194	18	22
Testing time (s)	0.86	0.82	N/A	1.14	0.86	0.84	0.77	0.82	0.67

Figure 14: Performance measures and charts illustrating the model errors

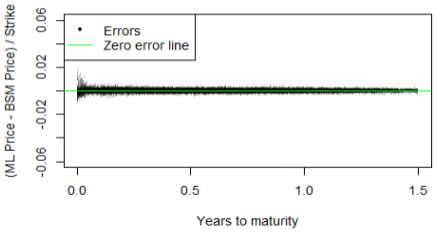
(a) Model option values vs BSM for Experiment 2



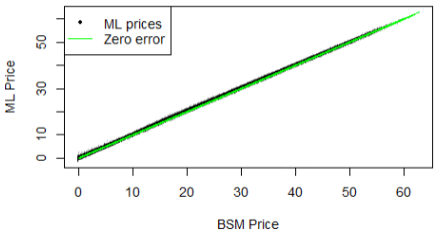
(b) Model errors/strike price vs underlying/strike for Experiment 2



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

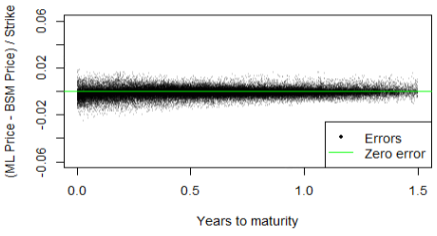


(d) Model option values vs BSM for Experiment 4

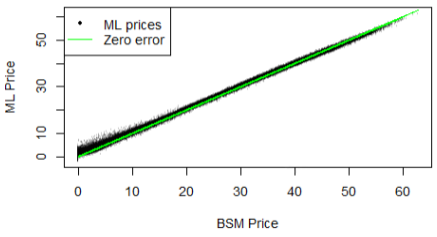


(e) Model errors/strike price vs underlying/strike for Experiment 4

(f) Model errors/strike price vs time to maturity for Experiment 4

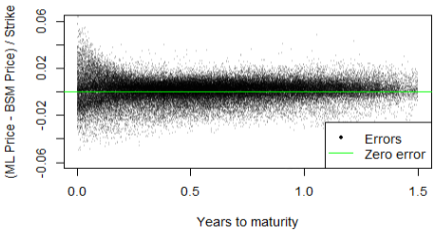


(g) Model option values vs BSM for Experiment 5

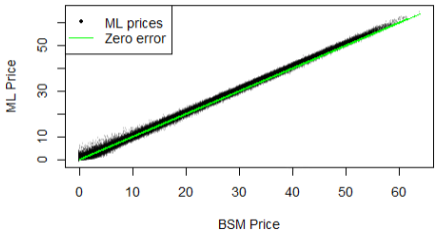


(h) Model errors/strike price vs underlying/strike for Experiment 5

(i) Model errors/strike price vs time to maturity for Experiment 5

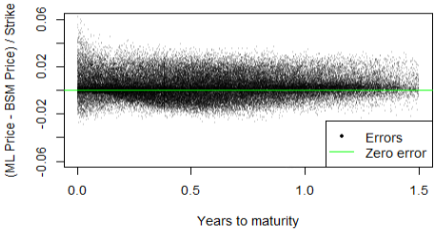


(j) Model option values vs BSM for Experiment 6

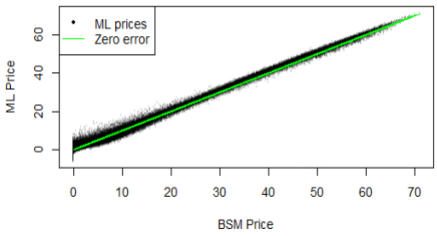


(k) Model errors/strike price vs underlying/strike for Experiment 6

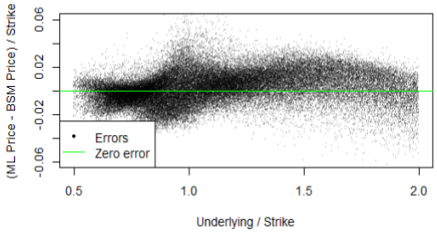
(l) Model errors/strike price vs time to maturity for Experiment 6



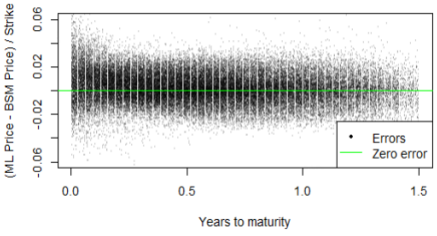
(m) Model option values vs BSM for Experiment 7



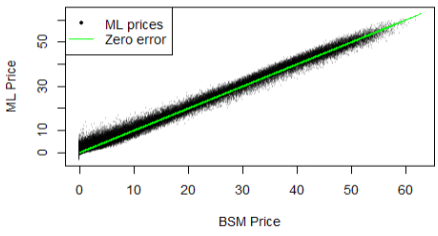
(n) Model errors/strike price vs underlying/strike for Experiment 7



(o) Model errors/strike price vs time to maturity for Experiment 7



(p) Model option values vs BSM for Experiment 8



(q) Model errors/strike price vs underlying/strike for Experiment 8

(r) Model errors/strike price vs time to maturity for Experiment 8

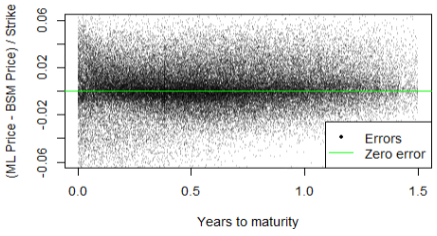
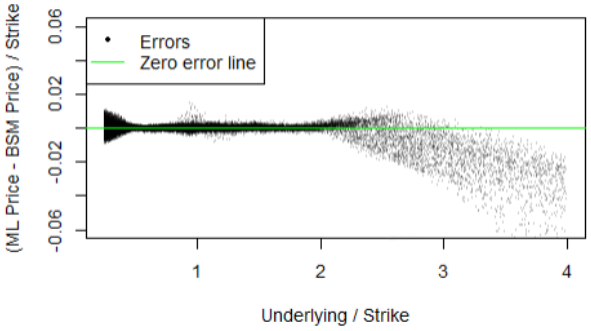


Figure 15: Model errors/strike price vs underlying/strike for Experiment 2 with use of out-of-range testing data



7.4 Comparison to Past results

Hutchinson, Lo, and Poggio 1994 is an influential paper in which, in their first experiment, they tested a similar set of models and simulated paths to Experiment 1 in this project. Although the experiments are not exactly comparable, the results of their paper show some similar trends but worse performance, presumably due to the advancement of both computing processing power and open source ML algorithms since the paper was published. For example, although they do not state the time required to train their model on simulated paths, they only felt able to train it on around 6,000 data points compared to 700,000 in this project. This is despite their training being run on a large mainframe of the time and the training of the experiments in this project using the CPU of a standard University lab machine. There is a mention in part 2 of their 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 6, 100 tests were run for Experiment 1 with a reduced time to maturity of 0.5 years and strike prices of 0.8, 1.0 and 1.2 times the starting underlying price. The means were taken for R^2 and the BSM model and ML model tracking errors. Being a percentage, R^2 is directly comparable between this project and their paper. Hutchinson et al used a starting underlying price of 50 while this project used 100 so their tracking errors were multiplied by 2 to make them comparable. The table shows that R^2 was far closer to 1 for this project while the tracking errors were consistently lower but possibly not by as much as would be expected given the much higher R^2.

Table 6: 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

8 Discussion

8.1 Setting Hyper-Parameters

Although there were wide differences in the performance of the models between the experiments, Table 4 shows that there were only a few differences between experiments in terms of which hyper-parameters were most successful. The results for some of the hyper-parameters were expected, as explained in Section 7.1. Some of the more notable results are discussed further below:

- The optimal level of regularisation increased a little with increasing noise and reducing numbers of datapoints in the training data, each of which would otherwise increase the chance that the model will learn to match outliers in the training data, and therefore over-fitting and causing a failure to generalise well to new input data. However, even for Experiments 7 and 8, it was noticeable that the level of L^2 regularisation required was still small at 0.002.

  Using any level of dropout as a regularisation method did not improve performance although the differences were small for Experiment 8, implying that further increases in noise or reductions in quantities of training data may have made dropout more useful.

- For the optimisation, the results of the experiments provided some support for using SGDM with a high momentum rather than one of the algorithms (see Section 4.2) which dynamically adjust the momentum.

- There was little support for increasing the number of layers in the MLP except for in Experiment 8. The lack of reduced losses when increasing the complexity of the model may have been due to the relatively simple form of the training output, e.g. the fact that it is monotonically increasing as shown in Section 6.1.3. The fact that more layers, but not more units, did result in a noticeable reduction in losses for Experiment 8 (which was the same as Experiment 6 except with far fewer training datapoints) is interesting and could be investigated further in the future.

- As was pointed out in Section 7.1, in most cases the losses could best be reduced by increasing the number of training datapoints. This is not always available in the real-world, so the second-best option of increasing the number of epochs of training may often be the most practical. However, with significant noise and the lower numbers of training datapoints in Experiments 7 and 8, the results showed that increasing the number of epochs did not reduce the losses significantly, or indeed at all for 8. This implies that this was the limit as to how good a fit the ML model can supply without more training datapoints.

- In many real-world use cases, speed of calculation of a price for a new, unseen option will be important. Consequently, the faster solution of choosing the model that is found to perform best for a testing dataset is likely to be more practically useful than using the average of multiple models, even though this can be expected to cause the accuracy of the price to be lower. The amount by which it is less accurate varies between the experiments and the training conditions. As expected, it is more when there are more models to average over, but averaging over more models makes the difference in calculation speed even greater.

  This relationship starts to break down in the experiments with greater noise and, especially for Experiments 6 and 8 in training condition 3 when the number of training datapoints is quadrupled, the relationship is reversed, and the best model approach is more accurate. This could be investigated further in the future as it could point to either a methodological issue or an unexplored facet of MLPs.

8.2 Practical Uses

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

1. Day-to-day trading and valuation of large portfolios of options. The ML models used in this project were trained to match prices and risk measures for financial instruments with non-linear payoffs that use a simple model of underlying asset price movements. The ML models could instead be trained to match prices in more complex situations which would otherwise require time-consuming methods to solve. These time-consuming methods result in a long window before a price can be given to a customer or a full valuation of a large portfolio can allow a trader or trading team to review the risks that need to be hedged. Major types of more complex situations are listed below.
   - 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 allow much faster results when it is more urgent for trading purposes, at the expense of lengthy periods required to train the ML model in advance on multiple examples of the calculated option value.
2. Finding current market practice in valuing options. Market option prices could be used as the training output, and both the contract details and a range of other data in the market could be used 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 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 the underlying's price movements.

The implications of the results of this project for these potential uses are discussed below, point 1 in Section 8.3 and points 2 and 3 in Section 8.4.

8.3 Learning Complex Functions

The first practical use listed in Section 8.2 requires the ML model to learn the known results of a complex function with known input parameters. When learning the results of a complex function, there may be some noise in the data if the target for the training output has been created using Monte Carlo simulations. However, the Monte Carlo method would not be in real-world use to calculate prices and risks of live trades if it gave significantly different answers each time it was run. Consequently, the noise generated would not be expected to be large.

In the absence of noise (i.e. in Experiments 1 to 3), the closeness of the fit of the ML model results to the BSM model results on the testing sets is very high. Given that the training time on a single CPU was only 1-2 minutes per model within the ensemble, this is encouraging. It suggests that an ML model trained with sufficient quantities of data could be a substitute for time-consuming numerical analysis methods such as Monte Carlo or finite difference when pricing new trades or valuing and producing risks for large portfolios.

A major challenge is likely to be related to the time required for the overall training process. In the experiments carried out for this project, data availability was not a problem. As many datapoints of training inputs can be created as desired and the BSM equations to calculate the training outputs are very fast. The only limit on the quantity of training data arose from the limits of available computing power to run the training cycles to create the trained model.

However, the time required for the numerical analysis method to produce a price for the option is the problem

that it is hoped will be reduced by using an ML model instead. The number of training datapoints would ideally be very high, especially taking into consideration the result that the ML model gives very poor results outside its training range so datapoints are needed for all conceivable eventualities. Therefore, obtaining sufficient training data over a wide enough range of all the inputs may be a much larger problem than training the ML model.

8.4 Learning from Noisy Data

The second and third practical uses in Section 8.2 require the use of real-world market data to provide the target for the training output as well as the training inputs. There are two main additional difficulties when training an ML model with real-world market data in this way:

1. The option prices being used as the target training output are likely to be noisy due to, firstly, low frequency of trading in some options and, secondly, difficulties in obtaining accurate prices for all of the underlyings, traded options and other training inputs at exactly the same time. However, Experiments 4 and later have shown that, given sufficient training datapoints, an ML model can find a good fit to the underlying function despite this noise.
2. It is not known before the training exactly which data items will prove relevant for calculating 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 target price to a great degree. However, Experiment 6 has 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.

The results of the experiments are also encouraging for this purpose given the training times employed. This provides some confidence that an ML model could learn the techniques that market participants are using to adjust simpler models such as BSM to price options.

The major challenges are likely to be related to setting up the training process in the optimal manner. Examples of these challenges are listed below:

- In the real-world, there are likely to be many possible factors that could affect the price of the option. They could be related either to the individual underlying asset or to the market that it is a part of. Finding proxies for these factors with readily available values is unlikely to be straightforward.
- Each of the ML models in this project were trained with 710,000 datapoints. Obtaining sufficient quantities of each of the real-world training inputs and outputs is likely to be an issue and decisions required on whether reduced accuracy for fewer datapoints is an acceptable compromise.
- In many cases, market data for the values of the options being priced may not be available. For example, many options in financial markets are traded 'over-the-counter' rather than on any public exchange where prices are publicly available. A proxy for this market data may be required such as similar publicly-traded options or employing alternative methods such as creating a hedged portfolio, working out what is its minimum expected value, and setting that to be the price. These alternative methods may result in complex loss functions to be minimised by the training process.
- Every underlying probably exhibits slightly different behaviour even to other outwardly very similar underlyings. Consequently, there is a question around how fine-grained the training of the ML models should be. Training for every underlying in the financial markets would be impossible so some sort of grouping would be required which could, using equities as an example, be at the level of all equities, equities per geographical region, equities per market sector, etc.
- The behaviour of the assets will change over time. Decisions would be required on, for example, how often to re-train the models, how long a history to use, whether to weight more recent history more heavily in the loss function, etc.
- Interest rates and underlying prices generally have long histories of data and obtaining them should not be difficult. Many publicly-traded options markets list options on a large range of underlyings with a number of strike prices and maturities. Multiply this range of options by a history of daily prices going back decades so again there should be plenty of data availability for the ML model to be trained with.

  However, it is possible that some of the training inputs will not have this level of availability. For example, if the decision has been made to train separate models for individual or small groups of underlyings, there may be fewer standard contracts or less liquid markets leading to less frequent pricing. Frequent re-training of models may also produce practical difficulties in obtaining all the training data required.

  Experiments 7 and 8 showed that the reduction in accuracy resulting from a reduction in training data quantity need not be so large as to rule out the ML model approach, but the results could not be improved by increasing training length or adjusting the hyper-parameters. However, in such a situation with only a low level of noisy training data available, it is likely that the parametric model approach is also less reliable. In this case, testing as discussed in Section 8.5 would be very important, in particular comparing the results of the parametric and non-parametric approaches.

8.5 Testing and Regular Certification

For use in real-world situations, market participants such as banks, investment funds, etc would need to check that the models continued to be appropriate for use. This could be both for their own internal risk management purposes and, especially for banks, to provide evidence to external regulators that their business was being run appropriately. The requirements for this would depend on what the ML model was being used for:

- If it was being used to match the known answer of a standard parametric model, as in the first practical use listed in Section 8.2, then the testing of that parametric model would cover most of the issues requiring ongoing review. However, there would be an additional issue due to the poor performance of ML models outside the range of the training data. Any time one of the items of market data that had been used as a training input started to approach the limit of the range that had been used for the training of the ML model, additional training with a wider range outside that limit would be required. Rigorous checks would need to be put in place to ensure that this occurred.

- If it was being used to price or hedge options directly, as in the second and third practical uses, then the regular testing requirements would potentially be far more extensive. Regular additional training on the latest data would be required, not just due to the range of inputs as above, but due to the fact that changes in the behaviour of a market can occur at any time. For example:

  - Regular testing would be required to check that the pricing supplied by the ML model matches prices in the market. In the final analysis, pricing and valuation must match market prices as those are the prices that the trader could reverse the position at, in order to remove all risk, whether identified or not, from the portfolio.
  - Regular updates of the latest market data would need to be added to the training of the ML models.
  - If markets that used to behave similarly begin to diverge, or those that used to behave differently begin to converge, then the decisions on how underlyings are grouped for training purposes may need to be revisited.
  - There would need to be regular checks of market liquidity to check both that the training inputs are still reliable and being regularly traded, and that there is no new market data available which should be added to the model. If there needs to be changes in the training inputs then there will need to be a lengthy time series of the new training inputs or else data availability problems will be caused. Therefore, even an apparently very good source of data may not be usable until it has existed for a number of years.

Many of these testing issues are not unique to an ML model. Parametric models also require a great deal of testing to prove they match the dynamics of a market and report risks and valuations correctly.

9 Further Work

Given the encouraging results discussed in Section 8, there are a number of further experiments that could be run to develop this work further. These can be divided into two groups.

Firstly, there are the following options to add to the proof-of-concept experiments carried out in this project.

- Investigate by how much the training and testing time could be reduced by using a GPU or set of GPUs rather than a single CPU.

- Investigate the results if more than one target training output were modelled, e.g. price and Delta. Firstly, optimal performance might require changes in the model setup such as more hidden layers or units while further regularisation or normalisation techniques might also reduce losses. Secondly, it could be useful to understand whether the results would be better if one model were used for both outputs or separate models for each output.

- Investigate what changes in the model setup might be required if the function shape were more complex. As described in Section 6.1.3, both the standard call option and the barrier modelled here are monotonically increasing or decreasing with all of the inputs. It is possible that a more complex function would also require more hidden layers, units, regularisation or normalisation.

Secondly, there are options to model real-world data and obtain a non-parametric model of option pricing.

- Build models of how the market participants are adjusting their prices to take the issues of their BSM or other parametric model into account. The aim behind these experiments is not to discover the true distribution of the underlying's price movements but to model the way in which the main market participants price trades.

  The training outputs would clearly be the market option prices. The training inputs could include any available information but would very likely need to include measures that either directly or implicitly act as proxies for the inputs used for the BSM model. Direct proxies might include the yield on government bonds for the risk-free interest rate, and some measure such as implied volatility for an ATM option for the volatility of the underlying, while more indirect proxies might include recent realised volatility of markets. Additional information not directly

related to markets such as economy-wide factors such as inflation or GDP growth might also be shown to be relevant if included in the training inputs of the ML model.

Care would need to be taken so as to avoid effectively including the outputs as inputs so that the model fails to learn anything not already included in the inputs. For example, using prices of volatility swaps for a range of strike prices as training inputs would already be including the effects of traders' adjustments to their parametric models.

- More fundamentally, build models which price options using actual market information on the movements in price of the underlying. In this case, more advanced forms of ANN such as LSTMs may be required due to the possibility that some form of memory of price moves at previous time steps might affect current price moves.

  Training inputs would be likely to be similar to those in the point above, except that it would be especially important to gain full time series of data for these, given that the model may be expected to learn how historic price moves affect future ones.

  There would be many options for the training outputs but a useful example might be the realised gains and losses on options. For each maturity of option being used in the training data, the gain or loss resulting from the underlying price on the maturity date could be used as the training output.

10 Conclusions

The parametric models currently being used in financial markets to price and hedge derivatives suffer from two major problems:

1. They do not accurately model the way that underlying prices move and therefore cannot accurately price derivatives of those underlyings. Consequently, market participants are forced to use adjustments to the results of the models based on their understanding of the market for those derivatives.

2. When more complex derivatives are priced or more complex models to improve the match to the price moves of the underlying are used, it is often very time-consuming to calculate an accurate result. This can be an issue either for large portfolios of trades for which risks need to be managed on a frequent basis, or for individual trades for which the decision to trade needs to be fast.

Consequently, there would be great utility in finding a practical alternative to these models. One potential alternative may be to train ML models with inputs from a range of available market data and an output of a market price or value of the derivative being priced. This could provide a non-parametric pricing model, that is, one that does not require a pre-defined distribution of expected future underlying prices.

This project has reviewed the performance of MLPs in matching the functions which result from existing parametric models. It has done this in conditions of increasing difficulty for the model by modelling with increasing numbers of training inputs, increasing noise in the training output, adding a dummy variable with no effect on the output and decreasing the quantity of training data.

The results have shown that a close fit to the underlying function can be achieved even in the most difficult scenario. This provides strong evidence that MLPs could also learn an unknown function assuming that all the major relevant market data are included within the training inputs. The suggested future work could progress the idea further and provide more detailed evidence on whether this approach could work practically for market participants.

References

Akinyemi, David (2015). *Pricing European Barrier Options with Partial Differential Equations*.

Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25. URL: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf.

Allaire, JJ and François Chollet (2023). *keras: R Interface to 'Keras'*. R package version 2.11.1. URL: https://CRAN.R-project.org/package=keras.

Allaire, JJ and Yuan Tang (2022). *tensorflow: R Interface to 'TensorFlow'*. R package version 2.11.0. URL: https://CRAN.R-project.org/package=tensorflow.

Bachelier, Louis (1900). "Théorie de la spéculation". PhD thesis. University of Paris.

Black, Fishcer and Myron Scholes (1973). "The pricing of options and corporate liabilities". In: *Journal of political economy* 81.3, p. 637.

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

Bühler, Hans et al. (2018). "Deep Hedging". In: *Swiss Finance Institute Research Paper Series* 19-80. URL: https://arxiv.org/pdf/1802.03042.pdf.

Carverhill, Andrew P. and Terry H. F. Cheuk (2003). "Alternative Neural Network Approach for Option Pricing and Hedging". In: *Risk Management*.

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

Derman, Emanuel and Iraj Kani (Jan. 1994). "Riding on a Smile". In: *Risk* 7, pp. 277–284.

Dugas, Charles et al. (June 2009). "Incorporating Functional Knowledge in Neural Networks". In: *Journal of Machine Learning Research* 10, pp. 1239–1262.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

Hafner, Wolfgang and Heinz Zimmermann (2009). *Vinzenz Bronzin's Option Pricing Models: Exposition and Appraisal*. Springer. URL: https://vdocuments.site/download/vinzenz-bronzins-option-pricing-models-exposition-and-appraisal.html.

Harvey, Edward (2023). *UoSMScDissertation*. GitHub repository. URL: https://github.com/xtceh/UoSMScDissertation.

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *CoRR* abs/1502.01852. arXiv: 1502.01852.

Hinton, Geoffrey (2012). *Lecture 6a Overview of mini-batch gradient descent*. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Hinton, Geoffrey E. et al. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv: 1207.0580 [cs.NE].

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

Hutchinson, James M., Andrew W. Lo, and Tomaso Poggio (1994). "A nonparametric approach to pricing and hedging derivative securities via learning networks". In: *The Journal of Finace* 49-3, pp. 851–889.

Lajbcygier, Paul and Jerome T. Connor (1997). "Improved Option Pricing Using Artificial Neural Networks and Bootstrap Methods". In: *International journal of neural systems* 8 4, pp. 457–71.

LeCun, Yann et al. (2012). *Neural networks: Tricks of the trade*. Springer Berlin Heidelberg, pp. 9–48.

Lo, Andrew W. (1999). *A non-random walk down Wall Street*. Princeton University Press. URL: http://assets.press.princeton.edu/chapters/s6558.pdf.

Loshchilov, Ilya and Frank Hutter (2019). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 [cs.LG].

McCulloch, Warren S. and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5, pp. 115–133.

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

Merton, Robert C. (1976). "Option Pricing when Underlying Stock Returns are Discontinuous". In: *Journal of Financial Economics* 3, pp. 125–144.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Raberto, Marco (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, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychology Review* 65(6), pp. 386–408.

Ruf, Johannes and Weiguan Wang (2020). *Neural networks for option pricing and hedging: a literature review*. arXiv: 1911.05620 [q-fin.CP].

Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams (1986). "Learning Internal Representations by Error Propagation". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf.

Stoyanov, Stoyan V. et al. (2011). "Fat-tailed models for risk estimation". In: *Working Paper Series in Economics* 30. URL: https://www.econstor.eu/obitstream/10419/45631/1/659400324.pdf.

Wilson, Ashia C. et al. (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://arxiv.org/abs/1705.08292.