Solving Black-Scholes PDE using Artificial Neural Networks

Refiloe Sarah Shabe (refiloe@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Prof. Phillip Mashele
North West University (NWU), South Africa

14 May 2020

Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

Abstract

Machine learning has been a topic of interest for years in academia and most industrial fields including finance. In finance, options play a vital role when it comes to good trading of assets, investments and risk management hence their pricing is very important. In literature, numerical methods such as finite difference method are used to solve the Black-Scholes partial differential equation as an option pricing model, however these numerical methods face the "curse of dimensionality" when it comes to pricing multi-asset options. Thereby, the Monte Carlo method is used in the case of pricing a higher dimensional asset to overcome the "curse of dimensionality". In this paper, we introduce a machine learning technique called artificial neural networks to solve the Black-Scholes partial differential equation numerically. We develop a multi-layer perceptron that approximates the solution using the Black-Scholes partial different equation's boundary conditions without violating the assumptions that govern it and compare with Monte Carlo method. The artificial neural network method proved to solve the partial differential equation with a greater accuracy in comparison with the Monte Carlo method.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Refiloe Sarah Shabe, 14 May 2020

Contents

1. Introduction

For centuries, financial options world was full of uncertainty and risk which was not only uncontrollable but also compound, impossible to analyse, until Fisher Black and Myron Scholes Black and Scholes (1973) came up with the Black-Scholes partial differential equation (Black-Scholes PDE) as a model of option pricing which is now a famous and most important equation in the history of modern finance according to (McKay, 2015).

An option is a financial contract that offers a right and most importantly not an obligation, to buy or sell an underlying asset at a specified price, called strike price or exercise price, at a given time under certain conditions of the contract. These assets are sometimes referred to as financial securities and there are securities which can be traded off as options. Few examples are stocks, bonds, interest rates, stock market index and other types of indexes. The contract has two types, call and put. A call option is a contract that offers the holder (owner) a right to buy, while a put option is a contract issued to offer the holder a right to sell an underlying asset. We will only discuss about call options where underlying assets are shares of stock through out this paper.

Moreover, options have different styles but the most commonly used ones, in theories and practice, are American and European style. The significant difference between the two styles is the time to exercise an option in hand. American options provides the owner with an advantage of selling or buying the option at anytime during its lifetime, while with European options the owner is entitled to exercise the option when it expires, that is at the end of the its lifetime. The Black-Scholes PDE has been widely used for pricing these two styles of options but there are other complex styles covered by (Hull, 2003).

In this essay, we will be discoursing on the valuation of European style options. Different numerical methods have been employed to approximate the solution of Black-Scholes PDE though these methods have some drawbacks on dimensionality when having to price a multi-dimensional asset. Using numerical methods to solve Black-Scholes PDE has always been common in literature. For example, Duffy (2006) introduced Finite Difference Method (FDM) and applied it to approximate Black-Scholes PDE while Topper (2005) used Finite Element Method (FEM) but these methods are computationally expensive and impracticable when pricing a multi-dimensional asset. In contrast, Cervera (2019) solved this PDE with a machine learning technique, Artificial Neural Networks (ANN), which is mathematically proven to overcome the curse of dimensionality by Grohs et al. (2018). We will investigate the feasibility and accuracy of the ANN when it is used to numerically solve Black-Scholes PDE compared to another model with the same property of overcoming dimensionality curse, Monte-Carlo method and draw comparisons

1.1 Objective of the Study

The aim of this study is to implement the artificial neural networks and Monte Carlo simulation models that solve Black-Scholes PDE to price a one dimensional asset and analyse the feasibility and performance of the ANN model in comparison with the Monte Carlo method.

1.2 Essay Outline

We start by stating useful financial definitions when pricing options in Chapter 2 followed by a short introduction of Black-Scholes PDE in Section 2.4. We also derive the PDE in this section leading to option pricing that is covered in Section 2.5. We wrap up the chapter with background on general ANN in Section 2.6 and their application in solving partial differential equations in Section 2.7.

In Chapter.3, we implement the Artificial Neural Network in Section 3.1. We describe the formulation of the model and the architecture to give a clear picture of how it looks like schematically in Section 3.2. The Monte Carlo Method is introduced and implemented in Section 3.3.

Chapter 4 lays out the results from the analytical or the exact solution, the predicted solution using the artificial neural networks method and Monte Carlo. Section 3.1, we discuss the solutions, briefly give the possible reasons behind the errors discovered in each method.

We finally conclude in Chapter 5 by discussing the best performing method, the advantages and disadvantages of using it. We also make suggestions on further work that can be carried out to investigate the performance of the two methods introduced in this paper.

2. Preliminaries

We provide definitions that are applied in deriving and solving Black-Scholes PDE, pricing options in general and briefly introduce options, option pricing and other methods used to value options. We finally end with an introduction of the method to be employed in this essay,artificial neural networks. We address their basic representation and application in partial differential equations, the formulation, and description of the application.

2.1 Probability Theory

2.1.1 Sample space (Durrett, 2019).
A sample space is the set of all possible probability outcomes and it is denoted by Ω. The elements of a sample space are denoted by ω.

2.1.2 σ-Field (Shreve, 2004).
Let Ω be a nonempty set, and let F be a collection of subsets of Ω. We say that F is a σ-algebra or σ-field provided that:

- F,
- $A \quad F \quad = \quad A^c \quad F$,
- $A_1, A_2, \cdots \quad F \quad = \quad \overset{\infty}{n=1} A_n \quad F$.

The set (Ω, F) is a measurable space.

2.1.3 Borel σ-field (Durrett, 2019).
The smallest σ-field that contains all open intervals in R is called Borel σ-field and it is denoted by $B(\mathsf{R})$.

2.1.4 Probability measure (Shreve, 2004).
Let Ω be a nonempty set, and let F be a collection of subsets of Ω. A probability measure P is a function that, to every set $A \quad F$, assigns a number in $[0, 1]$, called the probability of A and written as $\mathsf{P}(A)$. We require:

- $\mathsf{P}(\Omega) = 1$
- $\mathsf{P}(\ \overset{\infty}{i=1} A_i) = \sum_{i=1}^{\infty} \mathsf{P}(A_i) \quad A_i \quad F, A_i \bigcap A_j = \quad$ for $i = j$

A triple (Ω, P, F) is a probability space.

2.1.5 Random variable (Shreve, 2004).
A random variable, denoted by X, is a real-value function defined on Ω with a property that for every Borel subset B of R, the subset of Ω given by

$$\{X \quad B\} = \{\omega \quad \Omega; X(\omega) \quad B\}$$

is in the $\sigma-$algebra F.

2.2 Stochastic Process

2.2.1 Stochastic process (Shreve, 2004).

A stochastic process indexed by a set T , with values in a measurable space (S, G) is a collection $X = (x_t)_{t \ T}$ of measurable maps from a probability space to (S, G).

2.2.2 Brownian Motion (Shreve, 2004) .

Let (Ω, P, F) be a probability space. For each $\omega \quad \Omega$, suppose there is a continuous function $W(t)$ of $t \geq 0$ that satisfies $W(0) = 0$ and that depends on ω. Then $W(t)$ is a Brownian motion if for all $0 = t_0 < t_1 < \cdots < t_m$, the increments

$$W(t_1) = W(t_1) - W(t_0), W(t_2) - W(t_1), \ldots, W(t_m) - W(t_{m-1}),$$

are independent and each of these increments is normally distributed with

$$\mathsf{E}[W(t_{i+1}) - W(t_i)] = 0 \text{ and } \mathsf{Var}[W(t_{i+1}) - W(t_i)] = t_{i+1} - t_i,$$

$i \quad 0, 1, \cdots, m.$

2.2.3 Geometric Brownian Motion (Dundar, 2016).

Geometric brownian motion is the continuous time stochastic process $X(t) = X_0 e^{\mu t + \sigma W(t)}$ where $W(t)$ is a standard Brownian motion (Section 2.2.2) and $X_0 = X(0)$.

2.2.4 Stochastic Differential Equation (Shreve, 2004).

Stochastic differential equation (SDE) is an equation of the form:

$$dX(u) = \beta(u, X(u))du + \gamma(u, X(u))dW(u),$$

where the functions $\beta(u, X(u))$ and $\gamma(u, X(u))$ are given and called the drift and diffusion, respectively, W is Brownian motion and $u \geq 0$.

2.2.5 Ito's lemma (Focardi and Fabozzi, 2004).

Consider the process $X_t, t \quad [0, T]$ with SDE $dX_t = a(X_t)dt + b(X_t)dW_t$. Then for a function $f(t, x)$ with at least one defined derivative in t and at least two derivatives in x defined, we have,

$$df(t, X_t) = \left(\frac{\partial f}{\partial t} + a(X_t)\frac{\partial f}{\partial x} + \frac{b^2 X_t}{2}\frac{\partial^2 f}{\partial x^2} \right) dt + b(X_t)\frac{\partial f}{\partial x}dW_t.$$

2.3 Option Theory Terms

Consider the following terms as defined in (Wilmott, 1998):

2.3.1 Premium.

The amount paid for the option initially.

2.3.2 Intrinsic value.

The payoff that would be received if the underlying asset is at its current level when the option expires.

2.3.3 In the money.
An option with a positive intrinsic value. A call option is in the money when the asset price is above the strike price.

2.3.4 Out of the money.
An option without any intrinsic value or with intrinsic value equal to 0. A call option is out of the money when the price of the underlying asset is below the strike price.

2.3.5 At the money.
An option is at the money if the price of the underlying asset has leveled up close to the strike price.

2.3.6 Long position.
A positive amount of a quantity, or a positive exposure to a quantity.

2.3.7 Short position.
A negative amount of a quantity, or a negative exposure to a quantity.

2.4 A Brief Introduction of Black-Scholes PDE

The Black-Scholes PDE is a linear parabolic partial differential equation designed to govern the progression of the European option price denoted by V. V is a function of t and S where t is current time of the option and S is the price of an underlying asset (Wilmott et al., 1993). To deduce this equation, few assumptions outlined by Hull (2003) were made for easy analysis and adaptation of the PDE.

Assumptions governing the Black-Scholes equation (Hull, 2003):

1. The stock price S follows Geometric Brownian Motion (GBM) and discrete time version of the model is given as,
$$dS = \mu S dt + \sigma S dx.$$
2. Securities can be sold short with full use of proceeds.
3. There are no transactions costs or taxes. All securities are perfectly divisible.
4. There are no dividends paid during the life of the contract.
5. Risk-less arbitrage opportunities are not permissible.
6. Securities are traded continuously.
7. The risk-free interest rate, r, is constant throughout the life of a security.

It is no secret that stock markets fluctuate all the time, they are often volatile in comparison with other parts of the economy hence emphasising these assumptions is very crucial mainly because the option price depends on the asset price (Yermukanova et al., 2016). The more volatile an asset is, the more probable a profitable outcome at expiry (Wilmott et al., 1993).

2.4.1 Derivation of the Black-Scholes equation.
Consider the first assumption, stock price S follows GBM:

$$dS = \mu S dt + \sigma S dx, \tag{2.4.1}$$

where dx is a random variable which follows a normal distribution with mean 0 and variance dt, that is

$$dx \quad N(0, dt),$$

and dx can be written as

$$dx = \epsilon \sqrt{dt},$$

where

$$\epsilon \quad N(0, 1).$$

Now, we suppose $V(S, t)$ is a smooth function without taking into account the fact that S is stochastic. If S varies with a small change dS then V will also vary by some small amount provided that there are no singularities. Since V is now smooth then its Taylor series expansion is:

$$dV = \frac{\partial V}{\partial S}dS + \frac{\partial V}{\partial t}dt + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}dS^2 + \frac{1}{2}\frac{\partial^2 V}{\partial t^2}dt^2 + \frac{1}{2}\frac{\partial^2 V}{\partial S \partial t}dSdt + H.O.T. \tag{2.4.2}$$

Note: H.O.T - High Order Terms

From Equation (2.4.1),

$$dS^2 = \mu^2 S^2 dt^2 + 2\mu\sigma S dt dx + \sigma^2 S^2 dx^2. \tag{2.4.3}$$

Since $dt \quad 1$ then dt^2 is very small, $dx^2 = \epsilon dt \approx dt$ for some $\epsilon > 0$, and $dtdx = dt\epsilon \sqrt{dt} = \epsilon\sqrt{(dt)^3} \approx 0$, Equation (2.4.3) simplifies to:

$$dS^2 = \sigma^2 S^2 dt.$$

Upon substitution into Equation (2.4.2), we get:

$$dV = \frac{\partial V}{\partial S}dS + \frac{\partial V}{\partial t}dt + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2 dt. \tag{2.4.4}$$

All the H.O.T will be in terms of $dtdx$ thus they will reduce to 0.

Next, we need to eliminate the randomness imposed by the Brownian motion term dx of dS in Equation (2.4.4). To achieve this, we will choose a portfolio Π for stock whose holder has short position of one on V and long position of $\frac{\partial V}{\partial S}$ on S. That is:

$$\Pi = \frac{\partial V}{\partial S}S - V,$$

thus,

$$d\Pi = \frac{\partial V}{\partial S}dS - dV.$$

Substituting Equation (2.4.4) in $d\Pi$ yields:

$$d\Pi = -\left(\frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)dt. \tag{2.4.5}$$

Equation (2.4.5) does not involve any random variables which means that the portfolio must be risk-less during time dt.

We can assume that the portfolio earns same return as other risk-free securities. It follows that:

$$\begin{aligned} & d\Pi = r\Pi dt \\ = & -\left(\frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right) dt + r\Pi dt = 0, \\ = & -\left(\frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right) + r\left(\frac{\partial V}{\partial S}S - V\right) = 0, \\ = & \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2 + rS\frac{\partial V}{\partial S} - rV = 0. \end{aligned} \tag{2.4.6}$$

Equation (2.4.6) is the Black-Scholes equation where σ is the volatility of stock and r denotes the risk-free interest rate.

The solution $V(S,t)$ of Equation (2.4.6) is an unbiased value of an European option at any time during its lifetime until its maturity. (Hull, 2003) also discerned that the lack of flexibility in trading of European style options makes them cheaper and most popularly utilised. They are quite simple and reduce the amount of accounting needed by a brokerage and they are often used in stock and foreign currencies in accordance with (Wilmott et al., 1993). The Black-Scholes PDE has many solutions depending on whether an option is a call or a put. The solution for a European call option is found by solving the PDE (2.4.6) under the following boundary conditions:

$$\begin{cases} V(0,t) & = 0, \\ V(S,t) & = S_t - Ke^{-r(T-t)}, \\ V(S,T) & = max\{S_T - K, 0\}. \end{cases} \tag{2.4.7}$$

where S_t, S_T are the stock prices at time t and T respectively. $V(S,T)$ is called the final condition, the price of an option at the end of its lifetime.

2.5 Option Pricing

In Section 2.4, we have addressed that the solution to Equation (2.4.6) is the option price or value. Stock option prices are affected by numerous factors including (Hull, 2003):

1. The current stock price S_0.
2. The strike price K.
3. The time to maturity or expiration T.
4. The volatility of the stock price σ.
5. The risk-free interest rate r.
6. The dividends expected during the life of the option.

The dependence of a an option's value on how much the stock costs comes with pros and cons. Firstly, the worth of an option relies on the stock price being greater or less than the strike price. An European call option is valuable if and only if the stock price exceeds the strike price by a certain amount. This is because the profit attained from exercising the option is the difference between stock price and strike price. In simpler words, the higher the stock price, the more profitable the option (Black and Scholes, 1973).

Furthermore, if the strike price exceeds the stock price by some amount then there is a very high chance that the option is close to expiring without being used, because it will have no payoff at maturity date. On the other side, when maturity date is far in future then the value of the option will be equal to stock price because the bond that earns the strike price on date of expiry will have a low price. Whereas when the option is very close to expiration then its value will approximately accumulate the difference between stock price and the strike price provided the stock price is above the strike price, otherwise the option is worthless as stated in (Black and Scholes, 1973).

In contrast, a put option behaves in an opposite way rather and the significant difference between call and put options can be seen in calculation of their payoff:

$$\text{Call}^{payoff} = \max\{S_T - K, 0\}, \tag{2.5.1}$$

$$\text{Put}^{payoff} = \max\{K - S_T, 0\}. \tag{2.5.2}$$

In practice, an option's worth decreases as maturity date approaches and the stock price does not change. The relationship between the value of an European call option and stock price is illustrated in Figure 2.1:

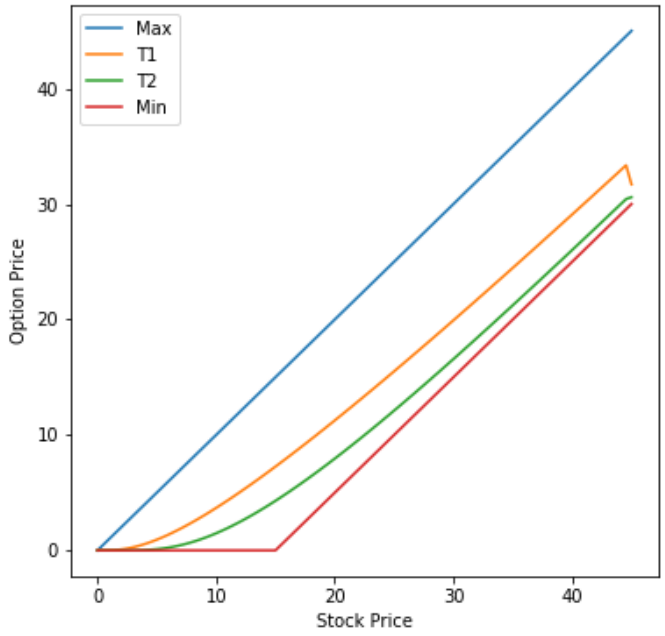

Figure 2.1: Relationship between option value and stock price.

In Figure 2.1, we suppose the strike price is R15. Line Min represents the minimum value an option can have at maturity date and this is due to the fact that an option cannot have a negative value or worth any amount less that the difference between stock price and strike price whereas line Max represents the maximum value an option can accumulate at maturity date while lines T1 and T2 represents option prices for two successive dates.

Valuation of options and their valuation formulas have been a topic of interest for many researchers years before the discovery of the Black-Scholes formula. Formulas of the same general form as the Black-Scholes formula were produced years before its discovery but unfortunately none of them were complete because of their dependence on arbitrary parameters according to (Black and Scholes, 1973).

The Black-Scholes formula is deduced from solving the Black-Scholes PDE (2.4.6) which gives a theoretical estimate of the price of European options and this is the formula which led to a rise of option trading in the early 1970's. In 1973, Robert.C Merton published a paper clarifying the mathematical understanding of option pricing models with reference to the formula Merton (1973). More than two decades later, Merton and Scholes won the Nobel Memorial Prize in Economic Sciences for their phenomenal work hence the model is sometimes referred to as Black-Scholes-Merton model (Hull, 2003).

Below is the Black-Scholes call option formula:

$$V(S,t) = SN(d_1) - Ke^{-r(T-t)}N(d_2) \tag{2.5.3}$$

where K is the strike price, T is the maturity date and $N(\cdot)$ is the cumulative distribution function for a standardised normal random variable given by:

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-1}{2}y^2} dy,$$

and

$$d_{1,2} = \frac{log(S/K) \pm (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{(T-t)}}.$$

2.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computer based systems that are motivated by the biological neural networks. They are build by an assemblage of linked units called artificial neurons or just neurons. The neurons communicate through a connection between them. Information from a neuron on one end is transmitted to another neuron on the other end by a connection between them, the neuron on the receiving end then ciphers the information and signals all the connected downstream neurons. In ANNs, neurons are typically organised in layers. Different layers accomplish different kinds of transformations given particular inputs. Communication between neurons essentially travels from the first layer through the middle layers all the way to the last layer. These layers are referred to as the input layer, hidden layer and output layer successively. Figure 2.2 is a schematic representation of a basic neural network.

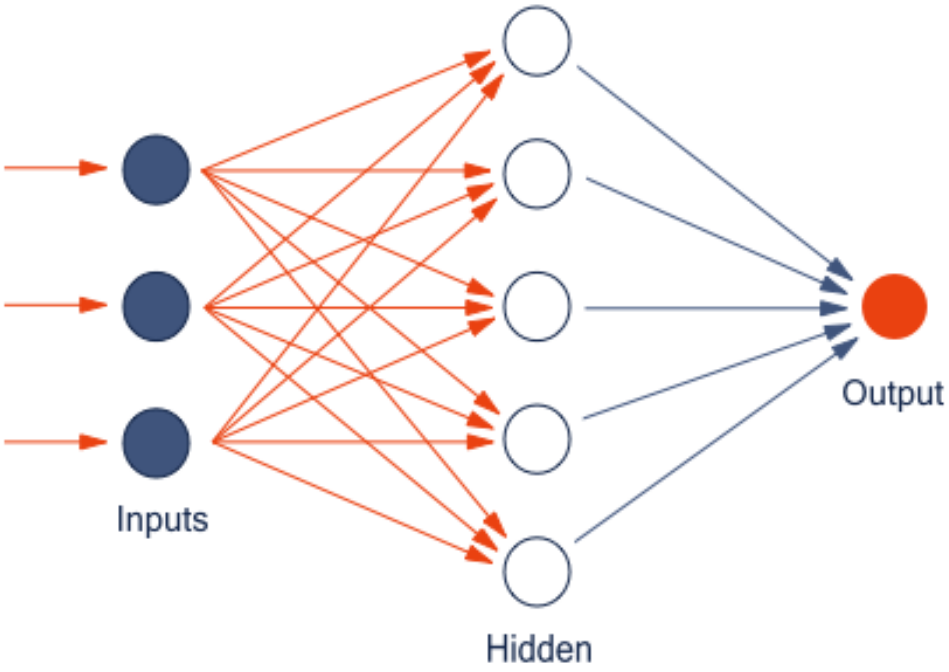

Figure 2.2: A simple neural network with one hidden layer (Castrounis, 2019).

There are many different kinds of layers of ANNs. Some commonly used layers are dense (or fully connected layers), convolutional, pulling and recurrent. These layers are distinguishing characters of all kinds of ANNs. Eventhough different layers performs different kinds of transformations, certain layers are most suited for a specific task over others. For example, convolutional layers are likely used in a model that is using images as data, recurrent layers would be used in a model that is working with time series data, and a dense layer is a layer connecting each input with each output within its layer.

The arrows in Figure 2.2 represents connections from one neuron to another. Each connection will have its own assigned weight which is just a number. The strength of connections between units is represented by weights. An input, received in the first layer, is passed on to the succeeding layer through the connection and this input will be multiplied by the weight assigned to this specific connection. The next step is to compute the weighted sum of all the connections pointing to each and every neuron in the next layer. The sum is then passed under a non-linear activation function which then changes the result to a number depending on the type of activation function.

An activation function of a neuron defines an output of that neuron given a set of inputs. It is also biologically inspired by the activity of a human brain where different neurons are fired or activated by different stimulus. For example, when one hears a sound of drumbeats , certain neurons in the brain will fire up and become activated such that a human can distinguish the sound from any other sound. Activation functions are classified under two types, linear and non-linear. Linear activation function is an identity function while non linear activation functions are sigmoid, softmax, rectified linear unit (ReLu) and hyperbolic tangent function (tanh).

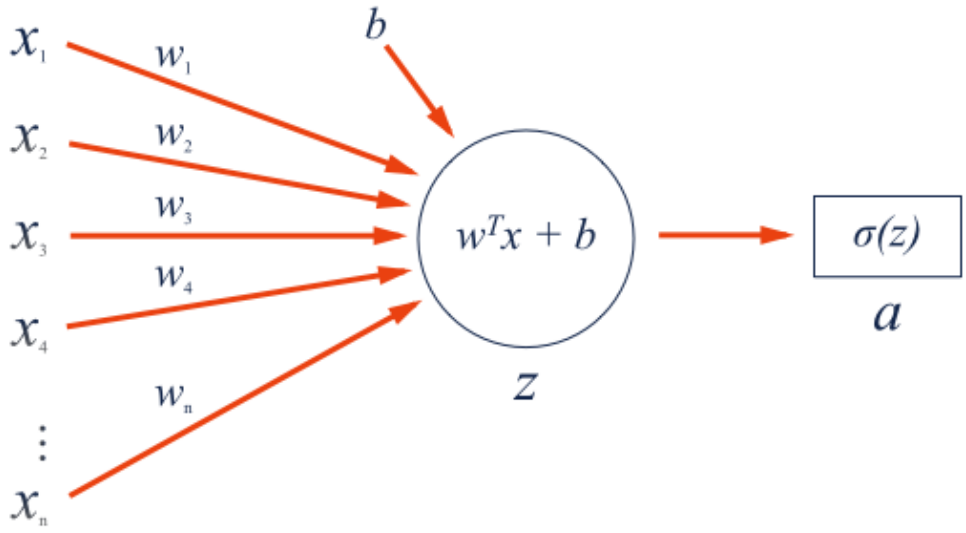

Figure 2.3: A neuron from the hidden layer with input size n (Castrounis, 2019).

In Figure 2.3, $x_1, x_2, \ldots, x_n$ are the inputs passed in the input neurons, $w_1, w_2, \ldots, w_n$ are the assigned weights to the connections, b is the bias parameter and finally a is the result after the activation function σ is applied on the weighted sum z. Figure 2.3 can be represented by the following equation:

$$a = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right). \tag{2.6.1}$$

2.6.1 Sigmoid and Hyperbolic Tangent.

Sigmoid and hyperbolic tangent (tanh) are two of the most commonly used non-linear activation functions especially in approximating solutions of continuously differentiable functions as well as optimization problems as done by Villarrubia et al. (2018). These activation functions are continuous, differentiable and bounded. The traditional sigmoid function sig and its derivative sig are defined below:

$$sig(x) = \frac{1}{1 + e^{-x}},$$

and

$$sig\ (x) = sig(x)(1 - sig(x)),$$

$sig(x)$ $[0, 1]$, for some x R.

The hyperbolic tangent and its derivative are defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

and

$$\tanh\ (x) = 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2,$$

$tanh(x)$ $[-1, 1]$ for some x R.

The graphical representations of sig and $tanh$ as well as their derivatives, are given by Figure 2.4 and Figure 2.5 respectively:

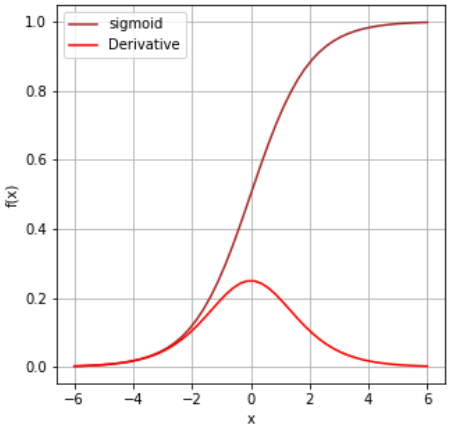

Figure 2.4: Sigmoid Graph

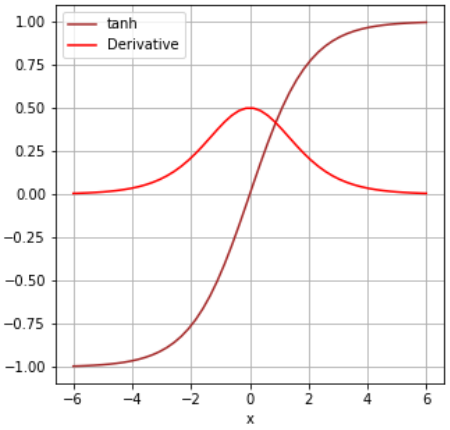

Figure 2.5: Hyperbolic tangent Graph

If the input x is a negative number then sigmoid will transform it into a number much closer to 0 while tanh transforms it to a number much closer to -1. If x is a positive number then sigmoid and tanh will transform it to a number close to 1. Lastly, if x is a number much closer to 0, sigmoid will then transform it to a number between 0 and 1 while tanh transforms it to a number between -1 and 1.

All factors addressed thus far build an ANN but we need to put it to work. To use the ANN, we need to train it so that it knows what to do whenever it is given a problem to solve. The model is fitted with data and it will learn the properties of this data in attempt to predict what the data represents. This is referred to as training.

Training an ANN is a process of solving an optimization problem which minimizes a function called the cost or loss function. The cost function is minimized using an optimization algorithm or simply an optimizer. Its objective is to find weights and biases which reduce the value of the cost function to as close as zero. As to how the optimizer attain its aim, it employs a method called backward propagation or just back propagation (Nielsen, 2015).

Back propagation is an algorithm used to compute gradients of the cost function during the tuning of the adjustable parameters of the model. During training, all parameters will be changing continuously in attempt to reach optimal values. This is called tuning. Back propagation is said to be computationally fast and gives the insights into how changing the weights and biases affects the general behaviour of the network (Nielsen, 2015).

Additionally, with the main goal being the computation of partial derivatives of the cost function, back propagation adopt two assumptions about the form of the cost function (Nielsen, 2015):

- The cost function can be written as an average of the cost functions for individual training examples.
- It can also be written as a function of the outputs of the neural network.

Emphasis on these assumptions is very crucial because we will refer to them when formulating our model.

2.7 Artificial Neural Networks for PDEs

For years, PDEs have been used in different areas to analyse the physical behaviour of systems in biology, physics and other sciences. Numerical methods are used to approximate their solutions and make interpretations based on these solutions. Hayati and Karami (2007) pointed that these methods work well and give acceptable accuracy but they are sequential and sometimes tedious, not to mention how computationally expensive they are especially in higher dimensions.

Different machine learning methods were later introduced as a new method that is less costly and very easy to follow for solving PDEs. Lagaris et al. (1997) presented a way of solving PDEs using unsupervised feed-forward neural networks. This method adopts a Multi-layer Perceptron (MLP) architecture with one hidden layer because of its well-known interesting property of approximating any function to arbitrary accuracy (Lawrence et al., 1996).

Furthermore, the neural network learns to solve the PDE analytically and it is trained in an unsupervised manner using the cost function that is derived from the differential equation itself coupled with its boundary conditions. The network will learn the analytical properties of the PDE's solution from trying to minimize the cost function with respect to the boundary conditions and then predict the behaviour of the PDE (Hayati and Karami, 2007). A closed analytic form solution which is continuously differentiable and satisfies the boundary conditions governing the given PDE, will then be constructed.

The method depends on neural network's properties of function approximation and the training of the model is not different from training a basic neural network which require computation of the gradient of error function with respect to the adjustable parameters of the model. Hence differentiability of the solution is essential and necessary. Also, the neural network architecture adds value to the importance of this method by providing the following features (Lagaris et al., 1997):

1. The solution is a closed form analytic and has unlimited differentiability
2. Generalization properties of the neural network characterise the solution.
3. Compact solution methods are obtained with very low demand on memory space.
4. The method is very general.

2.7.1 Description of the method.

We consider a general differential equation (Lagaris et al., 1997):

$$G\left(\bar{x}, \Psi(\bar{x}), \quad \Psi(\bar{x}), \quad {}^2\Psi(\bar{x}), \cdots, \quad {}^n\Psi(\bar{x})\right) = 0, \ \bar{x} \quad D \quad \mathsf{R}^n, \tag{2.7.1}$$

where G is a function of $n+2$ non-constant variables with respect to the first variable $\bar{x}$, which defines the structure of the differential equation. $\bar{x} = (x_1, x_2, \cdots, x_n)$, D denotes the domain of the differential equation, subject to the initial or boundary condition which could be of the form Dirichlet, Neumann or both (mixed), is the gradient operator, and $\Psi(\bar{x})$ is the solution to be computed.

To obtain the solution of Equation (2.7.1), the domain is divided into finite element mesh of points $\bar{x_i}$ $\hat{D}$. $\hat{D}$ is the discretized domain and the mesh depends on the discretization step such that Equation (2.7.1) is transformed into a system of equations with respect to each $\bar{x_i}$. That is:

$$G\left(\bar{x_i}, \Psi(\bar{x_i}), \quad \Psi(\bar{x_i}), \quad {}^2\Psi(\bar{x_i}), \cdots, \quad {}^n\Psi(\bar{x_i})\right) = 0. \tag{2.7.2}$$

If $\Psi_t(\bar{x}, \bar{p})$ denotes the trial solution, where $\bar{p}$ is an adjustable parameters corresponding to the weights and biases of the network, then this trial solution is constructed in such a way that it satisfies all the boundary conditions and defined as follows:

$$\Psi(\bar{x}, \bar{p}) = A(\bar{x}) + F(\bar{x}, B(\bar{x}, \bar{p})),$$

where $A(\bar{x})$ satisfies the boundary conditions and contains no adjustable parameters while $B(\bar{x}, \bar{p})$ is the single output of the neural network and F does not contribute in boundary conditions.

Now to approximate the solution, Equation (2.7.2) is transformed to a minimization problem as follows:

$$min_{\bar{p}} \sum_{\bar{x}_i \quad \hat{D}} G\left(\bar{x}_i, \Psi(\bar{x}_i), \quad \Psi(\bar{x}_i), \quad {}^2\Psi(\bar{x}_i), \cdots, \quad {}^n\Psi(\bar{x}_i)\right)^2. \tag{2.7.3}$$

At this point, the differential equation has been changed to an optimization problem without any constraints which makes it easier to handle. Minimization of Equation (2.7.3) is considered as the training of the ANN where the error $G(\bar{x}_i)$ corresponding to input $\bar{x}_i$ has to be reduced to 0. In computing the gradient of the error, we compute the gradient of the network together with gradient of its derivatives with respect to its inputs.

3. Methodology

In this chapter, we develop two models from the ANN method and Monte Carlo method of pricing options which solves Black-Scholes PDE to approximate its exact solution to the best possible accuracy.

3.1 Solving the Black-Scholes PDE using ANN

3.1.1 The Trial Solution.

To solve Equation (2.4.6) using ANN, we will employ the methodology described by Lagaris et al. (1997). As we have mentioned in Section 2.7, we want to find a closed form solution than can be easily evaluated. Firstly, we suppose $S \quad [0, S_{max}]$ and $t \quad [0, T]$ where S_{max} is the maximum value a stock can accumulate before an option expires.

Recall the final condition $V(S,T) = max\{S_T - K, 0\}$ which is the price an option has accrued up to its maturity and at this point it can be exercised, but we require to know the price movements of an option before it expires so as to take advantage of hedging opportunities that could come up before maturity hence $V(S,t)$ is the price of interest.

We will solve Equation (2.4.6) with respect to the boundary conditions (2.4.7) using a forward approach so the final condition will have to be transformed to an initial condition rather.

So, let $\tau = T - t$ then $\tau \quad [0,T]$. τ is the time taken to maturity of an option, and

$$\partial\tau = -\partial t \; = \quad -\partial\tau = \partial t.$$

Upon substitution of τ into Equation (2.4.6), we get:

$$\frac{-\partial V}{\partial \tau} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2 + rS\frac{\partial V}{\partial S} - rV = 0. \tag{3.1.1}$$

Now we need to adapt the boundary conditions (2.4.7) to the new variable τ:

$$\begin{cases} V(0,\tau) & = 0, \quad \tau \quad [0,T], \\ V(S_{max},\tau) & = S_{max} - Ke^{-r\tau}, \\ V(S,0) & = max\{S - K, 0\}, \quad S \quad [0, S_{max}]. \end{cases} \tag{3.1.2}$$

The following trial solution denoted by $U(S,\tau,\Theta)$, of Equation (3.1.1) is proposed:

$$U(S,\tau,\Theta) = A(S,\tau) + B(S,\tau)y(S,\tau,\Theta), \tag{3.1.3}$$

where,

$$A(S,\tau) = \frac{S}{S_{max}}(S_{max} - Ke^{-r\tau}) + \left(1 - \frac{\tau}{T}\right)\left(max\{S - K, 0\} - \frac{S}{S_{max}}max\{S_{max} - K, 0\}\right), \tag{3.1.4}$$

$$B(S,\tau) = \tau S\left(1 - \frac{S}{S_{max}}\right), \tag{3.1.5}$$

$$\therefore U(S,\tau,\Theta) = \frac{S}{S_{max}}(S_{max} - Ke^{-r\tau}) + \left(1 - \frac{\tau}{T}\right)\left(max\{S-K,0\} - \frac{S}{S_{max}}max\{S_{max}-K,0\}\right) + \tau S\left(1 - \frac{S}{S_{max}}\right) y(S,\tau,\Theta). \tag{3.1.6}$$

where $y(S,\tau,\Theta)$ is the single output of the neural network forward propagation and Θ are the adjustable parameters, weights and biases of the ANN. This trial solution is constructed such that it satisfies the boundary conditions (3.1.2) in terms of S and τ.

3.1.2 The Cost Function.

The next step is to discretize the domains to transform Equation (3.1.1) subject to the boundary conditions into an unconstrained minimization problem. This is because the method employed solves the PDE with discrete points or grid (Lagaris et al., 1997).

Consider $S \quad [0, S_{max}]$ and $\tau \quad [0,T]$. We choose a uniform step size for S to be $\quad S$ and for time τ be $\quad \tau$ with N_S and N_t number of steps, respectively:

$$S = \frac{S_{max}}{N_S},\ S_i = i \quad S, \quad i = 0,1,\ldots,N_S,$$

and

$$\tau = \frac{T}{N_t},\ \tau_j = j \quad \tau, \quad j = 0,1,\ldots,N_t.$$

From the discretized domain , we generate a data set consisting of mesh of all points on a grid of size $N_S \times N_\tau$. The data set will be a matrix of $N_S \times N_\tau$ rows and two columns corresponding to S_i and τ_j for all points on the grip.

We will split this data set into two parts, train and test data. The neural network will be trained on the train data and then later tested on a data set it has never seen before, which is the test data so as to satisfy the generality property of our solution.

The value of an option at time τ_j when the stock has accumulated a value of S_i is given by $U(S_i,\tau_j,\Theta) = U(i \quad S, j \quad \tau, \Theta)$ and we will denote it by $U_{i,j}$ for simplicity.

The boundary conditions play an important role of giving the exact price of an option along the boundaries. Since the trial solution (3.1.6) satisfies boundary conditions governing Black-Scholes PDE, the value of $U_{i,j}$ along the boundaries must be exact while at the rest of the grid will be approximated.

From Equation (2.4.6), we can now define the cost or loss function that the model has to minimize with respect to the adjustable parameters as:

$$L(\Theta) = \sum_{j=1}^{N_\tau}\sum_{i=1}^{N_S}\left[\frac{\partial U_{i,j}}{\partial \tau_j} - \frac{1}{2}\frac{\partial^2 U_{i,j}}{\partial S_i^2}\sigma^2 S_i^2 - rS_i\frac{\partial U_{i,j}}{\partial S_i} + rU_{i,j}\right]^2. \tag{3.1.7}$$

3.1.3 Accuracy Computation.

The error of the model is calculated using root mean squared sum of errors (RMSE). We compute the squared differences between the predictions $U_{i,j}$ of the network and the analytical solution $V_{i,j}$ which is known before hand, at every point (S_i,τ_j) of the grid then divide by the total number of points on the

grid $N_s N_t$. The error is the square root of the resultant value and it is given by the equation below:

$$RMSE = \sqrt{\frac{1}{N_t N_s} \sum_{i=1}^{N_S} \sum_{j=1}^{N_t} [V_{i,j} - U_{i,j}]^2}, \tag{3.1.8}$$

where $V_{i,j}$ is the analytical solution evaluated by Equation (2.5.3).

3.2 Architecture and Experimentation

We adapt the same architecture of the MLP used by Lagaris et al. (1997) to solve linear and non-linear PDEs. A MLP with an input layer that takes two inputs, one hidden layer and output layer with one neuron. We implement the MLP from scratch using python and taking advantage of its libraries for calculation of high order partial derivatives, integrals and to simulate pseudo-random numbers with preferable distributions.

After systematic experimentation with various models of different structures, we discovered that the best performing model is build with one fully connected hidden layer with ten sigmoid neurons and a linear output layer. We tried hyperbolic tangent (tanh) as an activation function since its derivative has a simpler expression than sigmoid thus computing the gradients would be optimized in a way but this was not the case, the gradients were converging slowly. We use Xavier Glorot initialization method to generate initial values of weights and set intial biases to zero. Initial values of adjustable parameters have a significant influence on the results of a MLP (Halawa, 2011) and sigmoid as an activation function gives good results when paired with the proposed initialization method (Glorot and Bengio, 2010).

We settle with Xavier Glorot initialization after iterative experimentation with other initialization methods. When using random initialization, we discover that we experience either exploding or vanishing gradients when using either sigmoid or tanh as the activation function. Initializing with random high values for the weights lead to large gradient values which change slowly and the model takes a long time to learn the data. On the other hand, initializing with lower values results in smaller gradient values which changes slowly and the model still takes a long time learning the data. The biases are initialised at zero to prevent saturation of the gradient at initial state of training.

To train the network, we employ mini-batch gradient descent as a back propagation optimization algorithm. Gradient descent is chosen because it can solve minimization problems of many variables (Nielsen, 2015). Since Black-Scholes equation is in terms of two variables S and t, it is sufficient to choose this optimizer. Mini-batch gradient descent divides the training data into small samples and lets the model to learn from these samples. This strategy speeds up the convergence of the cost function and increases the efficiency of the model.

Let L_x denote the cost function corresponding to the mini-batch $x = (S_i, \tau_j)$, w be the weights and b be biases. Then the weights and biases are updated using the following rules:

$$w \to w = w - \eta \frac{\partial L_x}{\partial w},$$

and

$$b \to b = b - \eta \frac{\partial L_x}{\partial b},$$

where w and b are the adjusted parameters, η is the learning rate such that $0 < \eta < 1$.

The choice of the learning rate is not entirely random. A small learning rate means a slower convergence to the global minimum, a large learning rate gives a high chance of shooting further and missing the global minimum. We choose an initial learning rate that performs well during experimentation which is $\eta = 0.0001$ and employ adaptive learning rate method called exponential decay learning to avoid the two cases discussed. Exponential decay method decreases the learning rate exponentially with each iteration or epoch depending on whether the training is done on a batch or mini-batch.

We use trial and error to find the optimal weights. This is achieved by iterating over the training data and observing the model's loss value with every epoch. We pay attention to the variation of the loss because we aim to keep it decreasing and stop the training when it starts increasing. To increase the accuracy of the model, we redefine the discretization to generate more training data points.

3.3 Monte Carlo Method

Monte Carlo (MC) method for a call option is an approximation of an option price as a value given by:

$$V = \mathsf{E}[e^{-rT}(S_T - K)^+], \tag{3.3.1}$$

$$= \mathsf{E}[e^{-rT}\max\{S_T - K, 0\}] \tag{3.3.2}$$

where V is the price of an option and in our case, it is a price of a European option, e^{-rT} is a factor used to discount the price for time $t = 0$ instead of $t = T$.

Wilmott (1998) stated that MC method is very general which means it is straight forward and easy to implement even though it is quite slow due to its dependence on a high number of iterations for a good accuracy. The validity of Monte Carlo method can not be questioned as it relies on a trusted and proven theorem, Kolmogorov's strong Law of large numbers and it states as follows:

3.3.1 Theorem (Kolmogorov's strong law of large numbers)**.** *Let* $\{X_n\}_{n\ \mathsf{N}}$ *be a sequence of independent identically distributed (I.I.D) random variables with expectation* $E(X_n) = \mu_n$ *and variance* $Var(X_n) = \sigma_n^2$*, both finite. If* $\sum_{n=1}^{\infty} \frac{\sigma_n^2}{n^2} < \infty$ *then* $\frac{1}{n}\sum_{i=1}^{n} X_i - \frac{1}{n}\sum_{i=1}^{n}\mu_i \to 0$*, as* $n \to \infty$*.*

Pricing options using MC method can be disadvantageous when an option holder wishes to take advantage of any arbitrage opportunities if they exist or hedging. This is because the holder will not know the price movements of the option they own during its life time since MC method gives the price of an European option at the current stock price and current time with no information about the value of stock at other times.

There are techniques applied to decrease the value of the variance and from Ramström (2017), they are given as follows:

- Antithetic Variable Technique,
- Control Variate Technique,
- Stratified Sampling.

We will not go into details about the above techniques in this paper.

In general, pricing of the option using this method has a few steps which are not complicated and very clear to understand. Firstly, we simulate the underlying asset price by generating random numbers and using the Brownian motion formula to create asset prices in a form of paths. For each and every path,

we calculate its payoff and average the payoffs over the number of paths generated. Finally, the value of the option at maturity is the discounted average of the payoffs.

In the case where the underlying asset is stock, the procedure of estimating the option price using MC method without additional assumptions is as follows (Hull, 2003):

- Simulate a random path for stock S in a risk-neutral world by using the fact that S follows a GBM. We divide the life of the option into N_t short intervals of length t and approximate Equation (2.4.1) as:

$$S(t + t) - S(t) = rS(t) t + \sigma S(t)\epsilon\sqrt{t}. \tag{3.3.3}$$

  This approximation allows the stock prices to be calculated recursively. Hull (2003) argued that simulating the log of stock prices gives better accuracy and precision since Equation (3.3.3) is only true in its limit as $t \to 0$.

  The log of stock $\ln S$ follows the process:

$$d\ln S = \left(r - \frac{\sigma^2}{2}\right) dt + \sigma dx, \tag{3.3.4}$$

  which follows straight from Ito's lemma. Equation (3.3.4) is then approximated by:

$$\ln S(t + t) - S(t) = \left(r - \frac{\sigma^2}{2}\right) t + \sigma\epsilon\sqrt{t}, \tag{3.3.5}$$

$$= S(t + t) = S(t)e^{\left(r - \frac{\sigma^2}{2}\right) t + \sigma\epsilon\sqrt{t}}. \tag{3.3.6}$$

  Therefore Equation (3.3.6) is the equation used to simulate the stock paths.

- Calculate the payoff or the return of the option per path as:

$$payoff = \max\{S_T - K, 0\}. \tag{3.3.7}$$

- Repeat the two steps above to get more payoff values as a sample.
- Average the sample payoffs to obtain an estimation for the expected value of the total payoff
- Discount the average payoff at a risk-free rate (multiply by e^{-rT}) to get an estimate value of the option.

To approximate the price of an option at any point (S_i, τ_j) using the procedure outline above, we simulate stock price paths for each and every S_i then calculate average payoff at this stock price. The option value at all time periods is the present value corresponding to each τ_j, that is the value obtained by discounting the average payoff with a factor $e^{-r\tau_j}$ for all $j = 1, 2, \cdots, N_t$ which is basically assuming that at every τ_j its maturity time.

The error of MC method is roughly estimated to be $\frac{1}{\sqrt{I}}$ where I is the number of iterations performed to generate the stock price paths and it follows a standard normal distribution for a large number of simulations and this error can be reduced by reducing the variance of the variance between number of simulations. The smaller the error the more accurate the results, the more efficient the method (Mohammed and Singh, 2014).

The process of pricing European call option using MC method leads to the Monte Carlo Algorithm outlined below:

Monte Carlo Algorithm

1. Create equal random stock paths $i = 0, 1, \cdots, N_s$.
2. Divide the time interval $[0, T]$ into time steps $t_0, t_1, \cdots, t_{N_t}$.
3. Simulate stock prices $S_i, i = 0, 1, \cdots, N_s$.
4. Arrange S_i in ascending order.
5. Compute the payoffs, $f(S_i) = \max(S_i - K, 0)$.
6. Compute the average of the payoffs, $g(f(S_i))$ and the discounting factor e_τ.
7. Calculate the option price:
8. **for** i in range $[1, N_s]$:

   $$V_i = e_\tau \quad g(f(S_i))$$

   end for

4. Results

In this chapter, we present the results obtained from the analytical solution and the methods discussed in Chapter 3. We carried out experiments using the real life examples of derivatives. We considered a stock price of $S \quad [0, 45]$ in rands (R), lifetime $\tau \quad [0, 1]$ in fractional years, risk-free interest rate $r = 0.01$, volatility of stock $\sigma = 0.55$ and strike price $K = R15$ and all codes are written in python.

4.1 Model Performance and Predictions

Figure 4.1, 4.2 and 4.3 represents the exact price V of an option at different time periods and stock prices on a 5×5, 10×10 and 60×60 grid respectively. It is basically Equation 2.5.3, evaluated at all equidistant points (S_i, τ_j) of each grid. We observe that for all the grids, the behaviour of the price is similar. When the stock is worth R0, the option is worth R0 too as expected but as the stock price increases, the option price increases too with very small values.

Before the stock accumulates a strike price of R15, the prices of the option starts increasing significantly for all periods until at stock price of around R22 where the price now increases drastically. At this point, since the stock price is greater than the strike price then the option is in-the-money and if it was to be exercised, the holder could buy the stock for R15 and later sell it for R22 earning the intrinsic value of R7 as profit.

The option price for all time periods converges to one price R30 at the maximum price a stock can worth. It is important to track the price movements of call options because if a call option expires in-the-money, it is possible to automatically exercise it. It can either be bought depending on whether the holder can afford to cover the strike price or rather be sold to lock the profits before it expires. Options' rewards come from knowing when it is a good time to take advantage and buy or sell the asset.

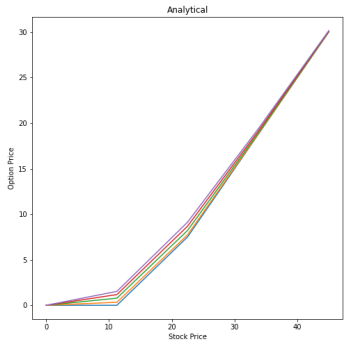

Figure 4.1: Exact solution evaluated on a 5×5 grid

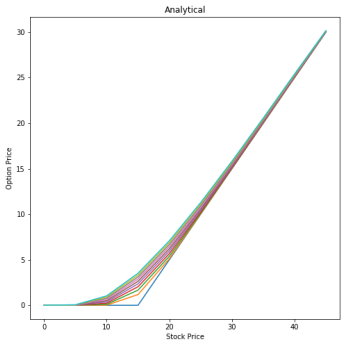

Figure 4.2: Exact solution evaluated on a 10×10 grid

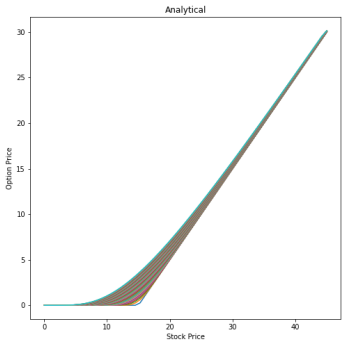

Figure 4.3: Exact solution evaluated on a 60×60 grid

We trained the MLP on the three different grids consisting of 25, 100 and 3600 points. The results for each set of points are given by Figures 4.4, 4.5 and 4.6 respectively while the Monte Carlo method predicted the prices as presented by Figures 4.7, 4.8 and 4.9:

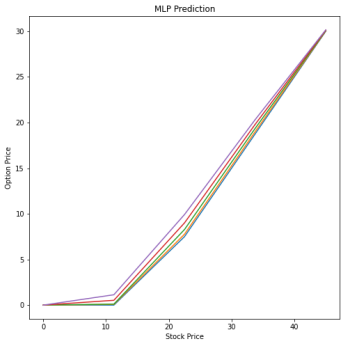

Figure 4.4: ANN prediction on a 5×5 grid

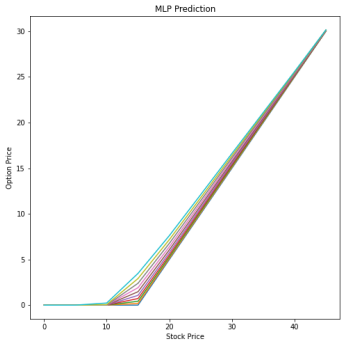

Figure 4.5: ANN prediction on a 10×10 grid

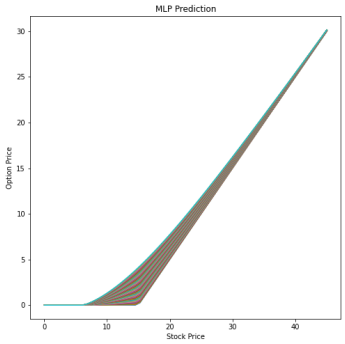

Figure 4.6: ANN prediction on a 60×60 grid

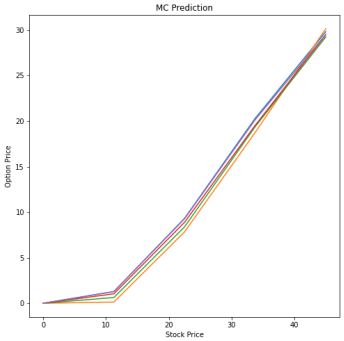

Figure 4.7: MC prediction on a 5×5 grid

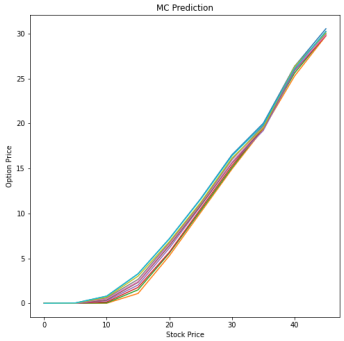

Figure 4.8: MC prediction on a 10×10 grid

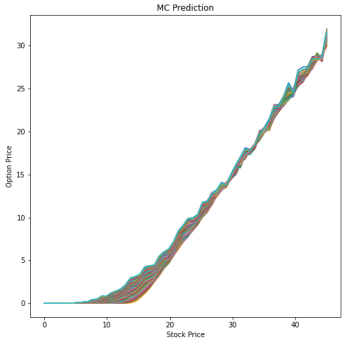

Figure 4.9: MC prediction on a 60×60 grid

We visualize the analytical solution and approximations for a grid of size 60×60 on a 3D plot in Figure 4.10. We observe the call option price movements at each time τ_j from the beginning of a contract until the end against changing stock prices. We vary the stock price because it is what happens in real life, the value of stock is always fluctuating with time but it cannot go up to an infinite number hence we set a maximum value it can accumulate. Figure 4.11 and Figure 4.12 illustrates the predictions of ANN and MC method respectively.

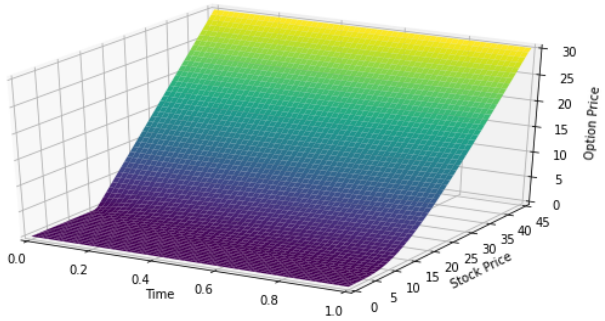

Figure 4.10: Analytical solution on 60×60

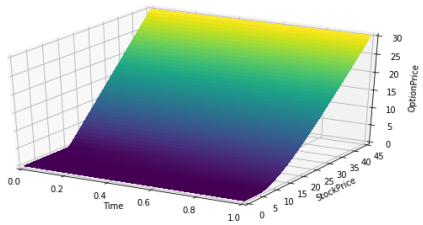

Figure 4.11: MLP prediction on 60×60

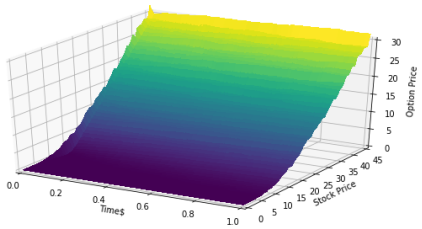

Figure 4.12: MC prediction on 60×60

4.2 Analysis of the Models

The performance of each model is analysed using root mean squared error which gives a measure of how accurately the model predicts the actual solution by computing the distance between the predicted solution and the actual solution.

Table 4.1: Table of errors

Grid Size	Root Mean Squared Error (RMSE)	
	Artificial Neural Network Method	Monte Carlo Method
5×5	0.335	0.408
10×10	0.301	0.372
60×60	0.2602	0.3916

4.3 Discussion

We calculate the value of an European call option using a feed forward neural network with one hidden layer on grids of different sizes and then apply Monte Carlo method on the same grid points to calculate the value of the same option. We compare the predictions of the two methods with the actual prices calculated using the analytical solution of Black-Scholes PDE for European call option and note the error between the actual and predicted prices.

Through iterative experimentation with the ANN method, we tried different architectures, learning methods and varied hyper parameters to make comparisons between performances of each developed MLP until we pick best performing one. A best performing MLP uses a lesser time to converge to the solution, for instance on a 60 grid, the model used 120 iterations making up 40 epochs to complete 3×3 mini batches which is computationally possible. On the other hand, MC method does not need any training but rather a larger number of iterations but the higher the number of iterations, the slower the model. Finally, with 10000 iterations the model gave approximations which are close to the actual solution.

The main difference between the predictions of the two models is that, the ANN method gives an exact the solution at the $\tau = 0$ for all stock prices S_i, at S $=$ R0 and S $=$ R45 for all τ_j and these are the boundaries of the domains so this result was expected since the ANN solution satisfies the boundary conditions while the MC method assumes the actual price is the present value of an option at all times. The assumption MC method makes acts as a drawback on its approximations hence the smoothness of price movements at all times, which can be vividly observed from figure 4.12.

In summary, from table 4.1, we observe that the ANN predictions are more closer to the actual solution than those of the MC method, therefore the ANN approximates the exact solution better when compared with MC method irregardless of number of iterations used by MC method. Also, MC method becomes infeasible when the number of iterations goes to 1000000.

5. Conclusion

5.1 Conclusion

We implemented a machine learning technique to the Black-Scholes PDE as a European call option pricing model. We developed an unsupervised multi-layer perceptron to approximate the solution of Black-Scholes PDE on grids of different sizes with the same values or risk-free interest rate, volatility and strike price. We compared the numerical solution with the Black-Scholes PDE's analytical solution and applied Monte Carlo method to solve the PDE for pricing the same option again then analysed the differences in both methods' performances and predictions. Even though our feed-forward neural network was a basic MLP, the ANN method still gave enough evidence to conclude that it feasible and more accurate when approximating Black-Schole PDE solution for European call option as compared to Monte Carlo method.

5.2 Further Work

For future, the next step will be to use the ANN method to price European options for a multi-dimensional asset and further solve Black-Scholes PDE with jump diffusion analysed in the paper by Zhang and Wang (2008). We also wish to employ deep neural networks technique for solving partial different equations used by Raissi et al. (2017) to solve PDEs such as Burger's equation to approximated Black-Scholes PDE. The method does not depend on trial functions and this fact makes it more improved and accurate.

Acknowledgements

I cannot thank God enough for the strength and tenaciousness He gave me to make this project accomplishable.

My greatest acknowledgement to my supervisor Prof. Phillip Mashele for his support, guidance and provision of material I needed to complete my essay. A special thanks to my tutors, Faraniaina Rasolofoson, Rock Stephane Koffi and Reem Omer Elmahdi for their assistance, advice and mostly for being patient with me on dark days. To our IT tutor, Lebeko Poulo, I appreciate you for being a brother I have always had.

I would like to convey my sincere gratitude and honour to AIMS founders and leaders for bestowing me the opportunity to have such a life changing experience and grow in all aspects while at it. I am truly grateful. To the academic and non-academic staff, I appreciate the support and hospitality which made my studying comfortable at all times. My heartfelt regards to AIMS 2019-20 intake which has become my family from all over Africa, I will always love you guys.

To my loving mother, my inspiration, my prayer warrior, thank you for always believing in me and to my brother and sister, "my deputy parents", you have been a true blessing in my life. I dedicate this work to my late dad, my superhero, *Kea leboha Mofokeng*.

References

Black, F. and Scholes, M. The pricing of options and coorporate liabilities. *The Journal of Political Economy*, 81:637–654, 1973.

Castrounis, A. *AI for Business People*. O'Reilly Media, Inc, 2019.

Cervera, J. A. G. Solution of black scholes equation using artificial neural networks. *Journal of Physics: Conference Series*, 2019.

Duffy, D. J. *Finite Di erence Methods in Financial Engineering: A Partial Di erential Equation Approach*. John Wiley Sons, 2006.

Dundar, S. R. *Stochastic processes and advanced mathematical finance*. PhD thesis, Department of Mathematics, University of Nebraska-Lincoln, 2016.

Durrett, R. *Probability: Theory and examples*, volume 49. Cambridge University Press, 2019.

Focardi, S. M. and Fabozzi, F. J. *The mathematics of financial modeling and investment management*, volume 138. John Wiley Sons, 2004.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Journal of Computational Chemistry*, 9:249–256, 2010.

Grohs, P., Hornung, F., Jentzen, A., and von Wurstemberger, P. A proof that artificial neural networks overcome the curse of dimensionality in numerical approximation of black-scholes partial differential equations. *math.NA*, 1, 2018.

Halawa, K. A method to improve the performance of multilayer perceptron by utilizing various activation functions in the last hidden layer and the least squares method. *Neural Process Lett*, 34:293–303, 2011.

Hayati, M. and Karami, B. Feedforward neural network for solving partial differential equations. *The Journal of Applied Science*, 19:2812–2817, 2007.

Hull, J. C. *Options, Futures And Other Derivatives*. Pearson-Prentice Hall, 2003.

Lagaris, I., Likas, A., and Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *Physics.comp-ph*, 1, 1997.

Lawrence, S., Tsoi, A. C., and Back, A. D. Function approximation with neural networks and local methods: Bias, variance and smoothness. *Australian Conference on Neural Networks*, pages 16–21, 1996.

McKay, F. *Black Scholes Solution by Finite Di erences*. PhD thesis, Schools of Mathematics and Physics, Queen's University, Belfast, 2015.

Merton, R. C. Theory of rational option pricing. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 4:141–183, 1973.

Mohammed, S. and Singh, S. Pricing options using Monte Carlo Methods. 2014.

Nielsen, M. A. *Neural networks and deep learning*. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com.

Raissi, M., Perdikaris, P., and Karniadakis, G. Data-driven solutions of nonlinear partial differential equations. *Physics Informed Deep Learning*, 1, 2017.

Ramström, A. Pricing of European and Asian options with Monte Carlo simulations : Variance reduction and low-discrepancy techniques. 2017.

Rubinstein, R. Y. and Glynn, P. W. How to deal with the curse of dimensionality of likelihood ratios in monte carlo simulation. *Stochastic Models*, 25:4, 547–568, 2009.

Shreve, S. E. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science Business Media, 2004.

Topper, J. *Financial Engineering with Finite Elements*. John Wiley Sons, 2005.

Villarrubia, G., Paz, J. F. D., Chamoso, P., and la Prieta, F. D. Artificial neural networks used in optimization problems. *Neurocomputing*, 272:10–16, 2018.

Wilmott, P. *Derivatives: The Theory and Practice of Financial Engineering*. John Wiley Sons, 1998.

Wilmott, P., Dewynne, J., and Howison, S. *Option Pricing: Mathematical models and computation*. Oxford Financial Press, 1993.

Yermukanova, B., Zhexembay, L., and Karjanto, N. On a method of solving the black-scholes equation. 2, 2016.

Zhang, K. and Wang, S. Pricing options under jump diffusion processes with fitted finite volume method. *Applied Mathematics and Computation*, 201:398–413, 2008.