Approximation of single-barrier options PDE using feed-forward neural network

Nneka Umeorah*[1] and Jules Clement Mba †[2]

[1]Department of Mathematics and Applied Mathematics, University of Johannesburg, Auckland Park, South Africa
[2]School of Economics, College of Business and Economics, University of Johannesburg, Auckland Park, South Africa

Abstract

Artificial neural networks are generally employed in the numerical solution of differential equation problems. In this paper, we propose an approach that deals with the combination of the feed-forward neural network method and the optimisation technique in solving the partial differential equation arising from the valuation of barrier options. The methodology entails transforming the extended Black-Scholes PDE, which defines a barrier option, into a constrained optimisation problem, and then proposing a trial solution that reduces the differential equation problem to an unconstrained one. This trial function consists of the adjustable and non-adjustable neural network parameters. We design it to be differentiable, analytic, and satisfy the initial and boundary conditions of the corresponding option pricing PDE. We compare the corresponding option values to the Monte-Carlo simulated values, Crank-Nicolson finite-difference values and the exact Black-Scholes prices. Numerical results presented in this research show that neural networks can sufficiently solve PDE-related problems with sufficient precision and accuracy. Furthermore, they can be applied in the fast and accurate valuation of financial derivatives without closed analytic forms.

Keywords: Barrier Options, Extended Black-Scholes Model, Artificial Neural network, PDE, Optimization, Monte-Carlo simulation, Variable initialization.

1 Introduction

Barrier options belong to the class of path-dependent exotic derivatives whose payoffs depend on whether the underlying entity's price attains a certain threshold level (barrier). They are generally classified as knock-out options and knock-in options. The knock-out barrier options expire worthlessly when the underlying reaches the threshold level, whereas the knock-in option becomes activated and payoff when the barrier equals the underlying price. These option types are further classified as 'down' and 'up' options. The former occurs when the barrier level is positioned below the underlying price and the latter when the barrier level is positioned above the underlying price. For this research, we will consider the down-and-out barrier call options, which pay no rebate or positive discount to the option holder in the event of a premature knock-out. Numerical approximation of barrier options is not new in finance since a lot of research has been done extensively. For instance, numerical methods such as the continuous Fourier sine transform [24], Crank-Nicolson finite difference method (FDM) [37, 38], binomial method [5], deep backward stochastic differential equation techniques [12, 40] have been applied in the barrier options pricing process. For comparison purposes, we will employ the extended Black-Scholes pricing formula as the closed-form benchmark of the pricing process, and these analytical formulas have been presented explicitly in [9, 32].

*umeorahnnekaozioma@gmail.com https://orcid.org/0000-0002-0307-5011
†jmba@uj.ac.za https://orcid.org/0000-0001-6462-6385

With regards to differential equations, several numerical techniques have been developed to aid the solution of either ordinary differential equations (ODE) or partial differential equations (PDE), constrained to certain boundary or initial conditions. These methods include finite difference, spectral methods, domain decomposition, Meshfree methods, Runge-Kutta, gradient decomposition methods, spline-based method, neural networks (NN), etc. The curse of dimensionality is often associated with solving numerical PDEs due to modelling uncertainties as random coefficients in the given equations. However, the implementation of deep NN has the theoretical guarantee of avoiding this dimensionality problem [31]. Based on such observation, Khoo et al. (2021) proposed a convolutional NN for solving problems relating to PDEs. Yadav et al. (2015) employed different forms such as the Multilayer Perceptron (MLP) NN, radial basis function NN, multiquadric radial basis NN, cellular NN, finite element NN and the wavelet NN techniques. Hussian & Suhhiem (2015) employed modified artificial NN, in which the training points were selected over a given open interval, in solving PDE. Other applications can be found in [1, 10, 20, 23, 33, 39].

Neural networks have proved essential in solving problems that have financial applications, and these can be seen in volatility forecasting, pricing and hedging of derivatives, bond rating assessment, etc. With regards to pricing and hedging, Chen & Sutcliffe (2012) conducted a comparative study on the performance of the artificial NN over the modified form of the Black model[6] to price and hedge short sterling options. Liu et al. (2009) employed a calibration NN, which is a data-driven approach, to calibrate certain financial asset price models (Bates and Heston) using the artificial NN. They further proposed that the parameters of stochastic volatility models, which are high-dimensional, can be calibrated accurately and efficiently. Anders et al. (1998) employed statistical inference for NN to improve the price of call options written on the German stock index (Deutscher Aktien Index -DAX). Their results deduced that implementing statistical specification strategies on NN led to a more efficient out-of-sample performance compared to the classical Black-Scholes pricing model. Morelli et al. (2004) implemented both the MLP and the radial basis NN to capture the non-linearity conditions which exist during the valuation of financial derivatives. They compared these techniques to the pricing and hedging of European and American options. They further deduced that MLP is a robust tool, requiring a longer training phase, whereas the radial basis function worked with limited performance. Other applications of NN in pricing and hedging of financial derivatives can be found in [4, 7, 29, 35].

A lot of research, both analytically and numerically, has been done to obtain the solutions to initial value and boundary value problems. Our focus in this work will be to implement the works of Lagaris et al. (1998), who employed the feed-forward neural network as an approximation technique in solving general problems pertaining to ODE, systems of ODE and PDE. From [23], the parameters of their proposed approximation element consist of 'weights' and 'biases' which need constant adjustment to minimize a specified error function. Their findings resulted in the construction of numerical solutions which are analytic, continuous and differentiable. Training the required network involves implementing an optimization technique that computes the gradient of the error function with respect to the parameters of the network. The choice for this numerical PDE solution is essential due to the consequence of the Universal Approximation Theorem, "Any single-layer feed-forward network can approximate a given measurable function to arbitrary accuracy irrespective of the activation function, the input space dimension, and the environment of the input space" [16, 17]. Similarly, we will apply this NN technique to solve the corresponding barrier options PDE, extend the solutions to capture the valuation process for the barrier options and compare the results to other numerical methods.

Our proposed trial solution for the barrier options PDE will be decomposed into a non-adjustable part that satisfies the boundary or the initial conditions of the corresponding options PDE and the adjustable part, which contains the parameters of the neural network. The knowledge and the valuation of the barrier options are crucial in finance because they are one of the most widely traded path-dependent exotic options in stock exchanges, and they are more suitable for hedging. Barrier options are a class of exotic options that are specifically customized to meet the risk management needs of investors. The majority of these option styles offer lower premiums compared to the clas-

sical option types. The analytical form under the extended Black-Scholes pricing model is known, and this research seeks to implement the pricing process from a different perspective. The essence of conducting a numerical solution (in this case, the NN) to the barrier option pricing is to highlight the applicability and accuracy of the NN techniques in solving problems pertaining to finance. The knowledge and application of this numerical method can offer guidelines for solving similar problems relating to the pricing of other exotic options without closed-form solutions. The analytical solution to this problem stems from an extension of the classical Black-Scholes model, and the results will be used as a benchmark in this research. Past reviews on applying this pricing model, which is well-known, have noted several limitations, such as the constant risk-free interest rate, constant volatility, risk-less arbitrage, etc.

To this end, the development of an efficient numerical valuation framework continues to be an actual problem in computational finance. Therefore, it is important to approach this pricing problem from a more non-parametric machine learning perspective, and applying the NN techniques to achieve this pricing efficiency is worth considering. Hence, the major highlights of this research are as follows:

- We extend the NN PDE techniques of Lagaris et al. (1998) to the solution of the barrier options PDE for pricing purposes.
- A neural network framework as an optimization technique in pricing the path-dependent barrier options is established.
- We introduce other numerical techniques, such as the antithetic Monte-Carlo methods and the Crank-Nicolson FDM, for comparison purposes.
- We provide numerical examples that highlight the valuation accuracy of the NN over the Crank-Nicolson FDM and the antithetic Monte-Carlo methods.

The remaining sections of this work cover the following: Section 2 highlights the exact Black-Scholes pricing model for the barrier options, as well as the implementation of the Monte-Carlo simulation and the Crank-Nicolson methods. Section 3 details the main methodology employed in this research, as it as provides an overview of the neural network implementation and its application to options pricing. Section 4 presents the numerical results and some discussion of the results obtained. This section also compares the exact Black-Scholes prices, the Monte-Carlo simulated values, Crank-Nicolson values, and the neural networks' approximated values. Finally, Section 5 concludes the study and provides guidelines for future research.

2 Exact Black-Scholes pricing formula for barrier options

In this section, we provide a brief overview of the extended Black-Scholes pricing for the barrier options, as well as the implementation of the Monte-Carlo simulation techniques in barrier option pricing. The price evolution of a given financial underlying can be written in terms of a stochastic process, which is defined on an appropriate probability space. We consider the assumptions which define the classical Black-Scholes pricing model, and in this case, we consider the extended Black-Scholes model in pricing the path-dependent exotic options. Let the price $S(t)$ of a given underlying asset follow a geometric Brownian motion, having constant volatility σ and constant expected rate of return μ (both $\mu, \sigma > 0$) be defined as:

$$\mathrm{d}S(t) = \mu S(t)\mathrm{d}t + \sigma S(t)\mathrm{d}W(t)\,, \tag{2.1}$$

where $W(t)$ is the standard Brownian motion. The numerical approach employed in this section considers the estimation of the expected value of the discounted payoff defined under the risk-neutral pricing measure Q. Here, $\mu = r$ as defined in equation (2.1), such that $r \in \mathbb{R}_+ > 0$ is the risk-free interest rate. The price of the barrier option $V(t, S(t))$ at any time t can be estimated by:

$$V(t, S(t)) = \mathbb{E}_Q[F(\tau, S(\tau))|S(t) = S]\,, \tag{2.2}$$

where τ is the stopping time or the first time which the underlying asset price $S(t)$ hits the threshold level and $F(\tau, S(\tau))$ is the discounted payoff. Approximating the option price in equation (2.2) entails applying different numerical techniques, but the subsequent Section 2.2 employs the antithetic Monte-Carlo simulation method. In this paper, we focused on the down-and-out barrier options, in which the barrier or threshold level is positioned below the underlying price ($S(t) > B$), and the underlying price has to take a downward movement in order for the option to be null and void. Furthermore, we define the random variable τ as:

$$\tau := \beta \geq t : S(\beta) \leq B\,,$$

and the corresponding payoff function (call option), with the barrier conditions as:

$$F(\tau, S(\tau)) = \begin{cases} \mathrm{e}^{-r(T-t)} \max\{S(T) - K, 0\}, \;\; B < S(x),\; \forall x \leq T \;\; (\tau = T) \\ \mathrm{e}^{-r(T-t)} R, \;\; \tau < T \end{cases}, \tag{2.3}$$

where K and R are the strike price and the cash rebate respectively. Note that $R = 0$ since we are limiting it to zero-rebate down-and-out barrier options. Let S', t' and $V(t', S')$ denote the current price of the underlying, current time and the down-and-out barrier call option value respectively, then under the Black-Scholes pricing framework, $V(t', S')$ satisfies the following PDE

$$\frac{\partial V}{\partial t'} + \frac{\sigma^2 S'^2}{2} \frac{\partial^2 V}{\partial S'^2} + S' \frac{\partial V}{\partial S'} - rV = 0\,, \tag{2.4}$$

subject to the following terminal condition (TC) and boundary conditions (BC):

$$\begin{cases} TC : V(T, S') = \max\{S' - K, 0\} \\ \\ BC : V(t', B) = 0 \\ BC : \frac{V(t', S')}{S'} \to 1 \qquad \text{for} \quad S' \to \infty\,, \end{cases} \tag{2.5}$$

where $t' \in [0, T]$ and $S' \in [B, S_{\max}]$, where $S_{\max}$ is the maximum asset price prior to the option's expiration. The solution of the above PDE for the down-and-out call option is explicitly provided in the following theorem

Theorem 2.1 *The extended Black-Scholes pricing formula for a down-and-out barrier call option is given as [9, 32]:*

$$V(t, S') = S'(d_1) - K\mathrm{e}^{-r\tau} N(d_2) - \left[S' \left(\frac{B}{S'} \right)^{2\phi} N(d_3) - K\mathrm{e}^{-r\tau} \left(\frac{B}{S'} \right)^{2\phi - 2} N(d_4) \right] \tag{2.6}$$

$$\textit{for } d_1 = \frac{\log\left(\frac{S'}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)\tau}{\sigma\sqrt{\tau}}, \quad d_3 = \frac{\log\left(\frac{B^2}{S'K}\right) + \left(r + \frac{\sigma^2}{2}\right)\tau}{\sigma\sqrt{\tau}}, \quad d_5 = \frac{\log\left(\frac{B}{S'}\right) + \left(r + \frac{\sigma^2}{2}\right)\tau}{\sigma\sqrt{\tau}}\,,$$

where $\tau = T - t$, $\quad d_2 = d_1 - \sigma\sqrt{\tau}$, $\quad d_4 = d_3 - \sigma\sqrt{\tau}$, $\quad \phi = (2r + \sigma^2)(2\sigma^2)^{-1}$ *and* $N(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{\frac{-y^2}{2}} \mathrm{d}y$ *is the cumulative standard normal distribution function.*

Note that Equation (2.6) occurs when $K \geq B$. Furthermore, we substitute $K = B$ into d_1 and d_3 for $K < B$.

2.1 Crank-Nicolson finite difference method

The CN numerical method averages the explicit and the implicit FMD, as it attempts to solve the corresponding barrier options PDE on a discrete asset-time grid. The stability criteria of this technique is unconditionally because it takes cognisance of a minute change in the option value for any corresponding minute change of the initial condition. Its accuracy increases with an increase in the

asset and time partitions, and are more accurate than either of the implicit or the explicit FDM. Given the PDE in equation (2.4), the goal is to discretize it, together with the terminal and boundary conditions. Denote the option price

$$V(t,S) = V_{i,j} \approx V(i\Delta t, j\Delta S)\,, \quad \text{where } i = 0,1,\cdots,N; \quad j = 0,1,\cdots,M$$

Discretizing equation (2.4) using the explicit FDM and substituting gives the following:

$$\frac{V_{i,j} - V_{i-1,j}}{\Delta t} + rj\Delta t\left[\frac{V_{i-1,j+1} - V_{i-1,j-1}}{2\Delta S}\right] + \frac{(\sigma j\Delta S)^2}{2}\left[\frac{V_{i-1,j+1} - 2V_{i-1,j} + V_{i-1,j-1}}{\Delta S^2}\right] = rV_{i-1,j}\,. \quad (2.7)$$

Discretizing equation (2.4) using the implicit FDM and substituting gives the following:

$$\frac{V_{i,j} - V_{i-1,j}}{\Delta t} + rj\Delta S\left[\frac{V_{i,j+1} - V_{i,j-1}}{2\Delta S}\right] + \frac{(\sigma j\Delta S)^2}{2}\left[\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta S^2}\right] = rV_{i,j}\,. \quad (2.8)$$

Taking the average and re-arranging gives the CN expression as follows:

$$-X_jV_{i-1,j-1} + (-1 - Y_j)V_{i-1,j} - Z_jV_{i-1,j+1} = X_jV_{i,j-1} + (-1 + Y_j)V_{i,j} + Z_jV_{i,j+1} \quad (2.9)$$

for $i = N-1, N-2, \cdots, 1, 0$ and $j = 1, 2, \cdots, M-1$, where

$$X_j = \frac{\Delta t}{4}[rj - \sigma^2 j^2], \qquad Y_j = \frac{\Delta t}{2}(\sigma^2 j^2 + r) \qquad \text{and} \qquad Z_j = \frac{-\Delta t}{4}[rj + \sigma^2 j^2]\,.$$

The discretization scheme of the CN method results to a tridiagonal system of equation which is solvable at each time step, and its order of accuracy is $\mathcal{O}(\Delta t^2, \Delta S^2)$. Further research on the implementation and the solution using the CN numerical techniques can be found in [19, 30, 37].

2.2 Monte-Carlo simulation method

The Monte-Carlo simulation (MCS) method is a robust and well-known numerical method that has a series of applications in financial derivative pricing. This simulation technique is one example of a stochastic model, as it has the ability to simulate the behaviour of a portfolio based on the given probability of the asset returns. It is equally used extensively due to its coding simplicity, though it has the drawback of a low convergence rate. Several MCS variance reduction techniques have been proposed over the years, such as the antithetic MCS, quasi MCS, importance sampling, stratified sampling, etc., to facilitate the convergence rate by increasing the precision of the estimates. This work further implemented the antithetic MCS to estimate the values of the down-and-out barrier call options. From Ito's formula, the analytic solution to equation (2.1) is given as:

$$S(t) = S(0)\exp\left(\left(r - \frac{\sigma^2}{2}\right)t + \sigma W(t)\right),\ 0 \le t \le T\,. \quad (2.10)$$

Sample paths are generated using the discretised form of equation (2.10). The antithetic MCS works in such a way that two sets of normally distributed random variables which are negatively correlated are simulated. The MCS of the underlying asset prices are simulated M times, the terminal payoff $F(\tau, S(\tau))$ is discounted, and the subsequent barrier option price $V(t,S)$ are obtained by taking the average of the discounted terminal payoff. Generally, the following pseudo-code was implemented in the valuation process using the antithetic MCS:

Algorithm 1 Pseudocode for simulating down-and-out barrier call options using antithetic MCS

1: Discretize the time $[0, T]$ into N uniform subintervals, such that the time-step length $\Delta t = \frac{T}{N}$, using the grid points $t_i = i\Delta t$, for $i = 0, 1, \cdots, N$.

2: Discretize two asset price processes in equation (2.10) such that both are negatively correlated as

$$S^*(t + \Delta t) = S(t) \exp\left(\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma(\sqrt{\Delta t})\epsilon\right) \tag{2.11}$$

$$S^+(t + \Delta t) = S(t) \exp\left(\left(r - \frac{\sigma^2}{2}\right)\Delta t - \sigma(\sqrt{\Delta t})\epsilon\right), \tag{2.12}$$

3: Apply the barrier condition in equation (2.3) on the discounted payoffs. The discounted payoffs are given as:

$$v^*(t + \Delta t, S) = \mathrm{e}^{-r\Delta t} \max\{S^*(t + \Delta t) - K, 0\} \tag{2.13}$$

$$v^+(t + \Delta t, S) = \mathrm{e}^{-r\Delta t} \max\{S^+(t + \Delta t) - K, 0\}. \tag{2.14}$$

4: For M= number of Monte-Carlo simulations, set v_k to be the sum of $v^*(t+\Delta t, S)$ and $v^+(t+\Delta t, S)$, where $k = 1, 2, \cdots, M$. Finally, obtain the mean estimator (barrier option value) by averaging the discounted payoffs as:

$$V(t, S) = \frac{1}{M} \sum_{k=1}^{M} \frac{1}{2} v_k \tag{2.15}$$

3 Neural Network approximation techniques

Neural network techniques can solve both ODE and PDEs, which depend on the function approximation capacities of the feed-forward neural networks (FFNN), and the resulting solution is in its analytic form. The advantages of using the neural network over the classical numerical methods in solving differential equations involve: the steady but non-exponential increase of computational complexity as the sampling points in the given interval is increased; the ability to provide good generalisation properties; and the ability to offer differentiable solutions whose analytic forms can be applied in any subsequent calculation [39]. Furthermore, research that considered the implementation of the Multilayer Perceptron (MLP)[1] technique, together with the extended backpropagation algorithm, has been employed by He et al. (2000) to train the derivative of a given FFNN further. They focused on solving a class of the first-order PDE for input-to-state systems, which are either linearisable or approximately linearisable [15].

According to Yadav et al. (2015), employing the MLP techniques in solving a general differential equation entails transforming the given differential equation into a constrained optimization problem and then constructing an appropriate trial solution that reduces the problem to an unconstrained one. This section focuses on using the neural network methods in solving the corresponding extended Black-Scholes PDE, which defines the barrier options. We employ the numerical concept given by Lagaris et al. (1998) via the use of ANN in solving initial and boundary value problems. The method description is as follows: First, we define a general differential equation

$$F(\boldsymbol{x}, \eta(\boldsymbol{x}), \nabla\eta(\boldsymbol{x}), \nabla^2\eta(\boldsymbol{x}), \cdots, \nabla^n\eta(\boldsymbol{x})), \quad \text{with } \boldsymbol{x} \in \mathcal{D} \subset \mathbb{R}^n, \tag{3.1}$$

where $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, $\eta(\boldsymbol{x})$ is the given solution that needs to be approximated and $\mathcal{D}$ is the domain of the given differential equation. The equation (3.1) is subject to some boundary conditions, which could be Dirichlet or Neumann. Solving the above differential equation entails discretising the domain $\mathcal{D}$, as well as its boundary S into finite element set of points $\hat{D}$ and $\hat{S}$ respectively. Equation

[1] An MLP is a class of the FFNN which consists of at least three layers that are the input layer, hidden layer and the output layer.

(3.1) is then transformed into the following system of equations

$$F(\boldsymbol{x}_i, \eta(\boldsymbol{x}_i), \nabla\eta(\boldsymbol{x}_i), \nabla^2\eta(\boldsymbol{x}_i), \cdots, \nabla^n\eta(\boldsymbol{x}_i)), \quad \text{with } \boldsymbol{x}_i \in \hat{\mathcal{D}} \subset \mathbb{R}^n\,. \tag{3.2}$$

Suppose that $\eta_t(\boldsymbol{x}, \overrightarrow{p})$ represents the trial solution having $\overrightarrow{p}$ as the adjustable parameters of the network's weight and biases, then the system can be converted into an optimization problem, such that we have

$$\min_{\overrightarrow{p}} \sum_{\boldsymbol{x}_i \in \hat{\mathcal{D}}} (F(\boldsymbol{x}_i, \eta_t(\boldsymbol{x}_i, \overrightarrow{p}), \nabla\eta_t(\boldsymbol{x}_i, \overrightarrow{p}), \nabla^2\eta_t(\boldsymbol{x}_i, \overrightarrow{p}), \cdots, \nabla^n\eta_t(\boldsymbol{x}_i, \overrightarrow{p})))^2, \quad \text{with } \boldsymbol{x}_i \in \hat{\mathcal{D}} \subset \mathbb{R}^n\,. \tag{3.3}$$

Thus, solving the specified PDE involves proposing a trial solution η_t which consists of a feed-forward neural network. This trial solution satisfies the given equation, and this trial solution can be decomposed into two parts. The first part $X(\boldsymbol{x})$, is written such that the initial or the boundary conditions of the main differential equation is satisfied. The second part $Y(\boldsymbol{x})\mathcal{N}(\boldsymbol{x}, \overrightarrow{p})$, involves the neural network adjustable parameters (the biases and the weights), and it is constructed such that it will not have any impact on the initial or the boundary conditions. Generally, this can be written as

$$\eta_t(\boldsymbol{x}) = X(\boldsymbol{x}) + Y(\boldsymbol{x})\mathcal{N}(\boldsymbol{x}, \overrightarrow{p})\,, \tag{3.4}$$

$\mathcal{N}(\boldsymbol{x}, \overrightarrow{p})$ is the feed-forward neural network with parameter $\overrightarrow{p}$ and input vector $\boldsymbol{x}$. Training of the neural network involves minimizing the equation (3.3) in which the error $F(\boldsymbol{x}_i)$ corresponding to the input vector $\boldsymbol{x}_i$ has to be tend to zero. Illustrating the concept of discretised and non-descretised domain, we have the input vector $\boldsymbol{x} = (t, S) \in \mathcal{D}$, and $\mathcal{D} = [0, T] \times [B, S_{\max}]$ is the domain of the down-and-out barrier options. Then, it follows that $\boldsymbol{x}_i = (t_i, S_i) \in \hat{\mathcal{D}}$, where $\hat{\mathcal{D}} = [0, N_t\Delta t] \times [B, N_s\Delta S]$ is the discretised domain of the down-and-out barrier options. Also, note that $\mathcal{D}$ and $\hat{\mathcal{D}} \in \mathbb{R}^2$, and $\Delta t = \frac{T}{N_t}$ and $\Delta S = \frac{S_{\max}}{N_S}$. Hence, optimization techniques are then employed in the minimization of the error function and the training of the neural network, which involves the error gradient estimation with respect to the network parameters and the corresponding inputs.

3.1 Neural Network approximation techniques for barrier options PDE

In this research, however, we extend the above methodology and implement the technique in approximating the extended Black-Scholes PDE. For the down-and-out barrier call options, the domain is written as $\mathcal{D} = \{(t, S) \in [0, T] \times [B, S_{\max}]\}$. To proceed, we first discretize this numerical domain in such a way that $t_x = x\Delta t$ and $S_y = y\Delta S$, where $\Delta t = \frac{1}{N_t}$ and $\Delta S = \frac{1}{N_S}$, for $x = 0, 1, \cdots, N_t$ and $y = 0, 1, \cdots, N_S$. We impose certain restrictions on the spatial and time boundary so as to convert the terminal value problem into an initial value problem. That is, denote $t = \frac{T-t'}{T}$, and $S = \frac{S'-B}{S_{\max}-B}$, such that $t \in [1, 0]$ and $S \in [0, 1]$. Substituting $-T\partial t = \partial t'$ into the PDE (2.4), we have

$$\frac{1}{T}\frac{\partial V}{\partial t} - \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2} - rS\frac{\partial V}{\partial S} + rV = 0\,, \tag{3.5}$$

with the following transformed conditions:

$$\begin{cases} IC : V(0, S) = \max\left\{S - \frac{K-B}{S_{\max}-B}, 0\right\}, \quad \forall S \in [0, 1] \\ \\ BC : V(t, 0) = 0, \quad \forall t \in [1, 0] \\ BC : V(t, 1) = 1 - \frac{K-B}{S_{\max}-B}\,. \end{cases} \tag{3.6}$$

Solving the corresponding barrier options PDE, we propose the following trial solution:

$$\eta(t, S : \overrightarrow{p}) = X(t, S) + Y(t, S)\mathcal{N}(t, S : \overrightarrow{p})\,, \tag{3.7}$$

where $Y(t, S) = tS(1-S)$. This term is constructed and chosen so as not to contribute to the boundary conditions, since $\eta(t, S : \overrightarrow{p})$ is already designed to satisfy them. According to [23], this second part does not have any impact on the boundary conditions since it vanishes upon the imposition of the

Dirichlet boundary conditions. For instance, the term $Y(t,S)=0$ at the boundary conditions of $S=0$ and $S=1$, as well as at the initial condition of $t=0$. Furthermore, the first term $X(t,S)$, without any adjustable parameters, satisfies both the boundary and the initial conditions, and is denoted by

$$X(t,S)=(1-t)\max\left\{S-\frac{K-B}{S_{\max}-B},0\right\}+tS\left(1-\frac{K-B}{S_{\max}-B}\right).$$

The last term of equation (3.7) handles the minimization problem, as it contains the adjustable parameters of the neural network. Thus, the trial function is explicitly given as

$$\eta(t,S:\overrightarrow{p})=(1-t)\max\left\{S-\frac{K-B}{S_{\max}-B},0\right\}+tS\left(1-\frac{K-B}{S_{\max}-B}\right)+tS(1-S)\mathcal{N}(t,S:\overrightarrow{p}). \quad (3.8)$$

Define $\eta(t,S:\overrightarrow{p})=\eta(x\Delta t,y\Delta S:\overrightarrow{p})=\eta_{x,y}$. The unknown NN parameter vector $\overrightarrow{p}$ can be estimated by considering the minimization problem which is defined with the functional as

$$G(t,S:\overrightarrow{p})=\frac{1}{2}\sum_{x=1}^{N_t}\sum_{y=1}^{N_s}\left[\frac{1}{N_t\Delta t}\frac{\partial\eta_{x,y}}{\partial t_x}-\frac{\sigma^2S_y^2}{2}\frac{\partial^2\eta_{x,y}}{\partial S_y^2}-rS_y\frac{\partial\eta_{x,y}}{\partial S_y}+r\eta_{x,y}\right]^2, \quad (3.9)$$

for $x=0,1,\cdots,N_t$, $y=0,1,\cdots,N_s$ and $G(t,S:\overrightarrow{p})$ is the error or cost function. The process can be constructed into an optimization problem

$$arg\min_{\overrightarrow{p}}G(t,S:\overrightarrow{p}), \quad (3.10)$$

given the known input-output pairs $((t,S),V(t,S))$ and a loss function $G(\overrightarrow{p})$. In solving the minimization problem of equation (3.10) after the derivative of the error function with respect to all the neural network parameters has been obtained, BFGS quasi-Newton techniques have been employed by Hussian & Suhhiem (2015). Furthermore, other series of backpropagation gradient descent techniques have been employed likewise, such as the RMSprop, Adam (which we used in this paper) and the stochastic gradient descent (SGD). Normally, the optimization algorithm is initialized, and they work in such a way that the direction of the error function reduces. Furthermore, during the training process, the weights and the bias of the network are constantly updated using the backpropagation algorithm as given in the equations below:

$$\begin{cases} w\to w' & =w-\zeta(i)\frac{\partial G}{\partial w} \\ \\ b\to b' & =b-\zeta(i)\frac{\partial G}{\partial b} \end{cases} \quad (3.11)$$

where $\overrightarrow{p}'=(w',b')$ are the adjustable parameters, $\zeta\in(0,1)$ is the learning rate which may vary during the iterations. This learning rate plays a crucial role during the training phase, as a large value of the learning rate can lead to the oscillation of the neural network's convergence. Smaller values of the learning rate, on the other hand, can result in the neural network 'learning' slowly and most likely get trapped in the regions of the local optima [25]. Furthermore, the partial derivatives of the trial function $\eta(t,S:\overrightarrow{p})$ in equation (3.9) are presented in the next section.

3.2 Gradient computation

The following section computes the gradient of the error function since, during the training stage, efficient error minimisation entails finding the optimal neural network parameter. Estimating the corresponding gradient of the error involves estimating the gradient of the neural network in connection with the gradient of the derivatives with respect to its inputs. Thus, from the trial function in equation (3.8), the following derivatives with respect to the parameters are obtained:

$$\frac{\partial \eta}{\partial t} = -\max\left\{S - \frac{K-B}{S_{\max}-B}, 0\right\} + S\left(1 - \frac{K-B}{S_{\max}-B}\right) + tS(1-S)\frac{\partial \mathcal{N}(t, S:\overrightarrow{p})}{\partial t} + S(1-S)\mathcal{N}(t, S:\overrightarrow{p}) \tag{3.12}$$

$$\frac{\partial \eta}{\partial S} = t\left(1 - \frac{K-B}{S_{\max}-B}\right) + tS(1-S)\frac{\partial \mathcal{N}(t, S:\overrightarrow{p})}{\partial S} + \mathcal{N}(t, S:\overrightarrow{p})(t-2tS) \tag{3.13}$$
whenever $S(S_{\max}-B) \leq K-B$

$$\frac{\partial \eta}{\partial S} = (1-t) + t\left(1 - \frac{K-B}{S_{\max}-B}\right) + tS(1-S)\frac{\partial \mathcal{N}(t, S:\overrightarrow{p})}{\partial S} + \mathcal{N}(t, S:\overrightarrow{p})(t-2tS) \tag{3.14}$$
whenever $S(S_{\max}-B) > K-B$

$$\frac{\partial^2 \eta}{\partial S^2} = tS(1-S)\frac{\partial^2 \mathcal{N}(t, S:\overrightarrow{p})}{\partial S^2} + 2t(1-2S)\frac{\partial \mathcal{N}(t, S:\overrightarrow{p})}{\partial S} + -2t\mathcal{N}(t, S:\overrightarrow{p})\,. \tag{3.15}$$

Next, consider a multilayer perceptron having 2 input units (t and S), one hidden layer with 10 sigmoid units, and then a linear output. Define the output of the neural network as

$$\mathcal{N}(t, S:\overrightarrow{p}) = \sum_{r=1}^{10} a_r f(w_r t + \lambda_r S + b_r)\,, \tag{3.16}$$

where a_r is the weight of the r^{th} hidden unit; b_r is the bias; w_r and λ_r are the coefficients of the time t to the r^{th} hidden unit and from the asset price S inputs to the r^{th} hidden unit respectively. Also, since f is a sigmoid activation function, then $f(\alpha_r) = (1+\mathrm{e}^{-\alpha_r})^{-1}$, where $\alpha_r = w_r t + \lambda_r S + b_r$. Then from equation (3.16), the derivatives of the neural network is given as

$$\frac{\partial \mathcal{N}(t, S:\overrightarrow{p}))}{\partial t} = \sum_{r=1}^{10} a_r w_r f(\alpha_r)(1-f(\alpha_r)) \tag{3.17}$$

$$\frac{\partial \mathcal{N}(t, S:\overrightarrow{p})}{\partial S} = \sum_{r=1}^{10} a_r \lambda_r f(\alpha_r)(1-f(\alpha_r)) \tag{3.18}$$

$$\frac{\partial^2 \mathcal{N}(t, S:\overrightarrow{p})}{\partial S^2} = \sum_{r=1}^{10} a_r \lambda_r^2 f(\alpha_r)(1-f(\alpha_r))(1-2f(\alpha_r))\,. \tag{3.19}$$

Suppose f is a tanh activation function, then $f(\alpha_r) = (1-\mathrm{e}^{-2\alpha_r})(1+\mathrm{e}^{-2\alpha_r})^{-1}$, where $\alpha_r = w_r t + \lambda_r S + b$. Then from equation (3.16), the derivatives of the neural network is given as

$$\frac{\partial \mathcal{N}(t, S:\overrightarrow{p}))}{\partial t} = \sum_{r=1}^{10} a_r w_r (1-f(\alpha_r)^2) \tag{3.20}$$

$$\frac{\partial \mathcal{N}(t, S:\overrightarrow{p})}{\partial S} = \sum_{r=1}^{10} a_r \lambda_r (1-f(\alpha_r)^2) \tag{3.21}$$

$$\frac{\partial^2 \mathcal{N}(t, S:\overrightarrow{p})}{\partial S^2} = \sum_{r=1}^{10} 2a_r \lambda_r^2 f(\alpha_r)(1+f(\alpha_r)^2)\,. \tag{3.22}$$

After defining the following derivatives of the error function with respect to all the parameters of the network, then, applying any form of minimization techniques can be straightforward. In this paper, we used the gradient descent algorithm, which involves the calculation of the partial derivatives of the error function with respect to the unknown neural network parameters.

4 Results and Discussion

In this section, all the numerical experiments and computations will be performed on an Intel Core(TM) i7-1165G7 CPU with a 2.80 GHz running on a 64-bit Windows 10 operating system. For the neural network implementation process, we used the Tensorflow (2.3.0 version), which is an open-source software library of toolboxes used in training neural network [2].

4.1 Pseudocode and variable initialization

The following pseudo-code was implemented in the valuation process using the neural network technique:

Algorithm 2 Pseudocode for pricing down-and-out barrier options using neural network in `Python`

1: **Import the `tensorflow` library** and other required libraries for computation purposes.
2: **Define the PDE function**: First, introduce the trial solution as shown in Equation (3.8), together with the actual option pricing PDE as shown in Equation (3.5) and then, compute their derivatives.
3: Discretise the training and the prediction data into the preferred grid structure.
4: **Build the neural network**: Initialize the parameters of the NN, such as the learning rate, number of the input, hidden and output units. Insert the placeholder using the `tf.placeholder()` to feed the actual training examples and then the variables using the `tf.Variable()` for the trainable variables $\overrightarrow{p} = (w, b)$.
5: **Create the NN model**: Introduce the MLP as the structure of the NN model, together with their activation functions, and define the feed-forward flow.
6: Define the loss function and use the `tf.reduce_mean()` function to estimate the average $L2$-norm which describes the differential operator. Introduce the optimizer and activate the minimization of the loss function.
7: **Start the training process**: First, initialize the global variables in the graph. Calculate the minimized loss function and the corresponding accuracy.
8: **Print** the output (option value).

For the NN architecture, we used the Mean Squared error as the error function, together with the Adam optimizer, which is an algorithm for first-order gradient-based stochastic optimization [22]. For the network construction phase, we considered 1000 training steps or iteration numbers with 100 display steps, a batch size of 100, and a learning rate of 0.001. We further built a NN architecture consisting of one input layer with two neurons (taking up values for asset and time), two hidden layers with ten neurons each, and finally, one linear output layer for the option value. We used the sigmoid activation function for each of these layers and the Adam optimizer to minimize the loss function. During the training phase, we initialized the global variables in the graph (i.e., assign default values to the weights and set the initial biases to zero). This initialization process of the adjustable parameters is essential because of the significant impact they have on the MLP values [14]. The execution stage resulted in the calculation of the batch loss and the corresponding accuracy, thereby providing the values at the output layers (option values).

In a variable generation, we used the tensorflow class of `tf.Variable()`, and when a variable is defined, it essentially means passing a tensor and its corresponding value to the graph. Before training the NN, it is important to initialise the variables because they can directly affect the NN convergence and accuracy. These variables (or learnable parameters, such as weights and bias) can be initialised

specifically globally or from other variables. We initialized the tensorflow global variables in the graph with the `ipython` command `tf.global_variables_initializer()`. Normally, these parameters are initialised to constant values such as 0's or 1's, or using samples from a normal or uniform distribution, or even from more sophisticated techniques like the *Xavier or Glorot Initialization*. In our work, we randomly initialised the weights from normal distribution to small values (close to zero) but not too small since the latter could result in numerical instabilities.

We considered 1 input layer (2×1 neurons), 2 hidden layers (2×10 neurons) and 1 output layer (1×1 neuron). Let $h1$ denote the weights[2] from the input layer to the first hidden layer, then, $h1 = 20$ values. Let $h2$ denote the weights from the first hidden layer to the second hidden layer, then, $h2 = 100$ values. Let $o1$ denote the weights from the second hidden layer to the output, then, $o1 = 10$ values. Similarly, let $b1$ be the bias[3] to the first hidden layer, then $b1 = 10$ values. Let $b2$ be the bias to the second hidden layer, then $b2 = 10$. Let $o2$ be the output, and we note that all these variables were randomly initialized from the normal distribution before the training process. Random initialization of NN weights is done because many learning algorithms and optimizations used to train the models must operate in stochastic domains. Thus, the algorithm has to optimize and estimate the best set of weights for some specified mapping functions in the dataset. In fact, weight initialization is a crucial step to be employed before proceeding with the training process, and it can speed up the whole learning process. More information on the initialization of weights can be found in [11, 28].

4.2 Application to pricing barrier options

For the pricing process, we consider the pricing of a barrier call option (down-and-out) having the following features: $K = 70, r = 0.05, t = 0, T = 0.25, S = 80, S_{\max} = 200, B = 50$ and $\sigma = 0.4$. It is noted that the option knocks out when the asset price equals the barrier level, and hence we suppose the range of asset prices to be $50 \leq S \leq 200$ while confined in the asset boundary domain. The following numerical solutions obtained were considered for different mesh parameter sizes $(5 \times 5), (10 \times 10)$ and (20×20) respectively. For the Crank-Nicolson method, we used the step size for time $\mathrm{d}T = \frac{T}{N}$ and the step size for asset $\mathrm{d}S = \frac{S_{\max}}{M}$, where $N = M = 250$. For the Monte-Carlo methods, we consider the time-step $\Delta T = \frac{T}{N}$, where $N = 100$ and 100000 number of simulations. In considering each of these grid sizes, we estimated the loss function values, the NN approximation for the option prices, as well as the comparison with the Monte-Carlo prices. For the NN values, we used the python function `np.linspace()` to generate numbers that are evenly spaced over the given interval $[0, 1]$. Hence, for each grid size, we randomly chose three values inside $[0, 1]$ to print their NN values. Here, the NN values are the option values obtained while using the neural network method. We chose these three-point values to ensure uniformity in the grids used in this research instead of printing all the NN values at each discretised point in the interval. Finally, our major focus is on the last value, that is, at $t = 1$, and this NN value will be compared to the options prices obtained using the antithetic Monte-Carlo simulation.

4.2.1 Neural network option values for (5×5) grid

In this subsection, the training data and the prediction data were transformed into a (5×5) grid set. From Figure 1, we observed that there was a steady decline when the loss values were being plotted against the step-wise iteration numbers. Prior to the discretisation of the training steps, that is, for the iteration number of 1, we observed the loss value was 1.07192; it reduced to 0.043156 when the iteration number of 100; then, at the 1000th iteration number, the loss number became 0.00345. The essence of implementing the neural network in the training process is to minimise the loss or error function as closely as possible to zero in order to ensure an accurate prediction of the option prices.

[2]The weight is the parameter of connection between one layer and the other.

[3]Bias are similar to intercept on linear equations. They are additional parameters in the NN which are used to adjust the output values together with the weighted sum of the input values to the neurons

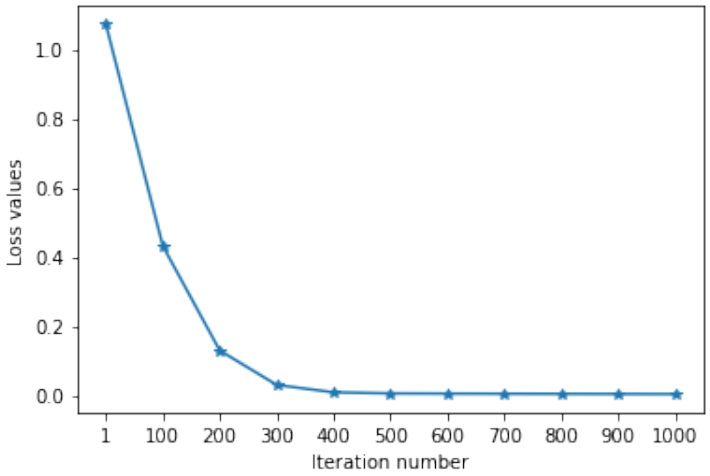
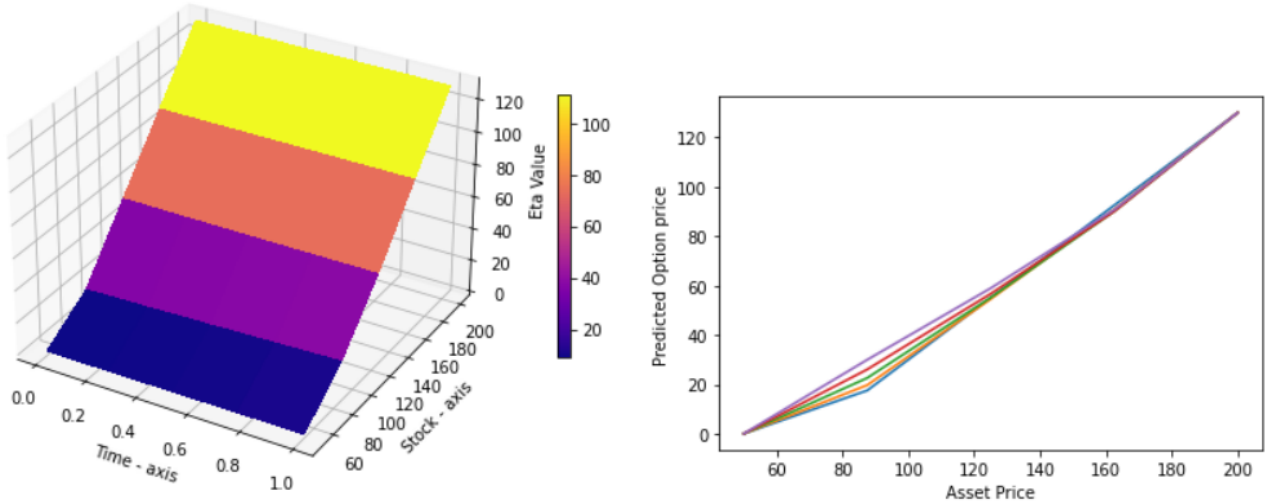


Figure 1: Loss against training time



(a) 3D-View: neural network approximation

(b) Option prices vs Asset prices

Figure 2: NN option values (5×5 grid) for different assets and time

Figure 2a gives a 3-dimensional plot of the barrier option values (denoted as 'Eta Value') against the time and stock grid. The option value increased with an increase in the stock prices, and this is expected since we are dealing with call options. For the down-and-out barrier options, the stock grid started from $S = 50$, which is equivalent to the threshold level B. Any values below this level give an option value that is worthless, that is, $V(t, S \leq B) = 0$. In Figure 2b, the predicted option price for all the discretised time period converged to $S_{\max} = 200$, and this is in line with one of the boundary conditions for the call option features. Furthermore, we expect to see the impact of strike price 70 in Figure 2b, as we plot the predicted option price against the asset or stock price. If the stock price is less than the strike price, the option values become significantly close to zero, but this is not the case as observed in Figure 2b. Thus, the use of the 5×5 grid did not capture it well, and hence we further increased the grid in the next two subsections. Finally, Table 1 compares the values of the down-and-out barrier call options using different methods. The exact prices were obtained using the extended Black-Scholes pricing model as presented in Theorem 2.1, the antithetic Monte-Carlo values as shown in Section 2.2, and the proposed neural network techniques as presented in Section 3.1. We further showed the converging property of the NN method as we considered the option values for the randomly selected time-grid $t = 0.250, t = 0.750$ and $t = 1$. Hence, it was noted that the neural network methodology approximated the option pricing PDE reasonably well and gave rise to values

that are significantly close to the exact value (for $t = 1$), in contrast to the antithetic MCS values.

Table 1: Option values using the NN (5×5 grid) for different uniform time-grid and the MCS

Asset Value	**Exact Price**	**Neural Network Values**			**Antithetic MCS Values**
		$t = 0.250$	$t = 0.750$	$t = 1.000$	
50	0.00000	0.00000	0.00000	0.00000	0.00000
87.5	19.27888	18.57066	19.67485	**19.68020**	20.98222
125	55.87765	54.59599	55.81658	**55.82417**	55.77751
162.5	93.36960	92.27423	92.37823	**92.50000**	92.16897
200	130.86955	130.00000	130.00000	**130.00000**	129.97920

4.2.2 Neural network option values for (10×10) grid

This subsection increased the training and the prediction data grid to a (10×10) grid set. Figure 3 plots the loss values against the step-wise iteration number. We observed that prior to splitting the training steps, that is, for the iteration number of 1, the loss value was 0.02131. It reduced to 0.00175 when the iteration number of 100, then at the 1000th iteration number, the loss number became 0.00046. The goal is to reduce the loss function further to be close to zero, as this directly impacts the predicted option values.

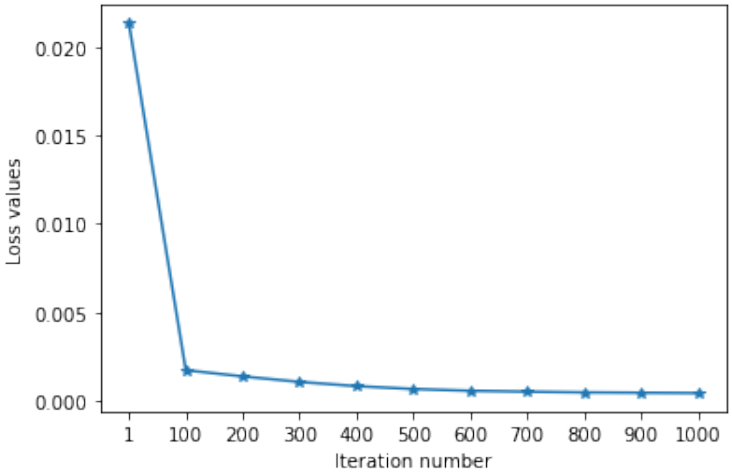
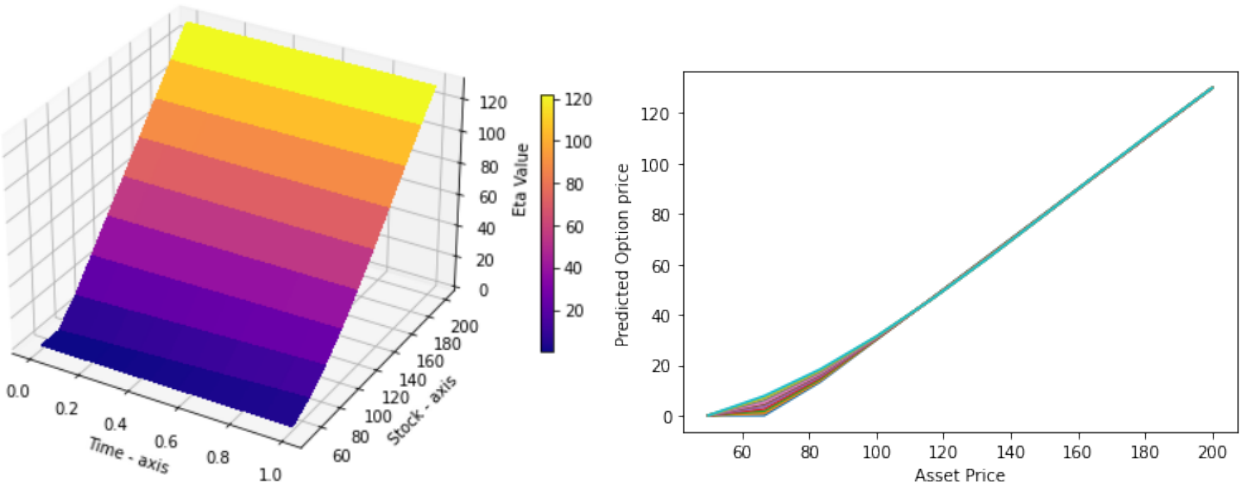


Figure 3: Loss against training time

The 3-dimensional option value plot for the 10×10 grid is presented in Figure 4a, and the option value against the underlying asset price is equally portrayed in Figure 4b. The latter figure further showed that the predicted option price for all the discretised time period converged to $S_{\max} = 200$. We also observed that the predicted barrier option values increased with an increase in the asset prices. The impact of the strike price $K = 70$ can be seen in both Figures 4a and 4b, as the option values became null and void, provided that the strike remained greater than the underlying prices.



(a) 3D-View: neural network approximation

(b) Option prices vs Asset prices

Figure 4: NN option values (10×10 grid) for different assets and time

The increment in the grid sizes used in the neural network training equally affected the performance of the approximated option values, as seen in Table 2. The table compares the exact prices with the approximated neural networks and Monte-Carlo simulated values. Additionally, after considering a 10×10 space and time grid, we randomly chose time-grid $t = 0.333, t = 0.667$ and $t = 1$ for the computation of the NN option values and to highlight its convergence property. This result further showed the high and accurate approximation capacity of neural networks over the antithetic Monte-Carlo methods.

Table 2: Option values using the NN (10×10 grid) for different uniform time-grid and the MCS

Asset Value	**Exact Price**	**Neural Network Values**			**Antithetic MCS Values**
		$t = 0.333$	$t = 0.667$	$t = 1.000$	
50	0.00000	0.00000	0.00000	0.00000	0.00000
67	4.41940	4.02705	4.16974	**4.30772**	4.56927
83	15.34719	14.99001	15.22344	**15.55932**	15.98222
100	31.08007	31.34350	31.20172	**31.05410**	31.20362
117	47.89331	46.48068	46.55566	**46.66664**	49.62802
133	63.87226	63.20105	63.24549	**63.33341**	64.24460
150	80.86981	79.70591	79.82961	**80.00001**	82.33303
167	97.86958	96.66649	96.66675	**96.74896**	99.49788
183	113.86955	113.53766	113.62662	**113.72443**	114.34060
200	130.86955	130.00000	130.00000	**130.00000**	129.97920

4.2.3 Neural network option values for (20×20) grid

This section finally considers when the grid is further increased to a (20×20) grid size. Figure 5 plots the loss values against the step-wise iteration number. We observed that prior to splitting the training steps, that is, for the iteration number of 1, the loss value was 0.00681. It reduced to 0.00215 when the iteration number of 100, then at the 1000th iteration number, the loss number became 0.00028. Furthermore, we aim to minimise the loss function to zero, as this behaviour increases the precision of the predicted option values. During the training process, as we compared the (20×20) grid size to the (10×10) and (5×5), we observed loss values reduced drastically for the larger grid sizes, and they had the tendency to get to zero faster. Thus, we can say that increasing the grid sizes of the neural network can positively affect the precision of the option values.

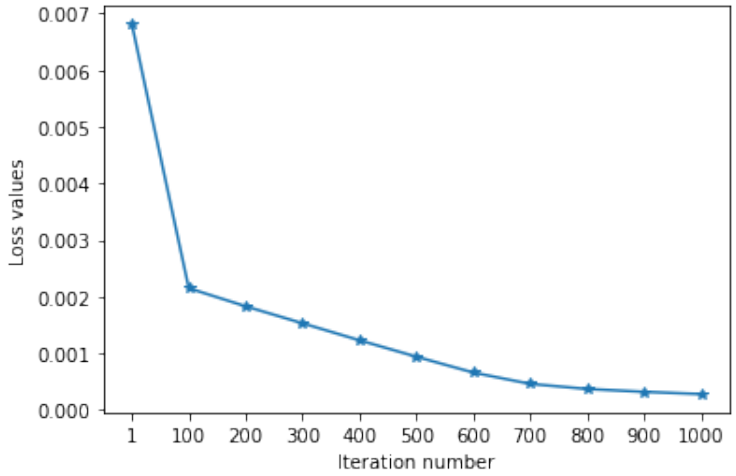
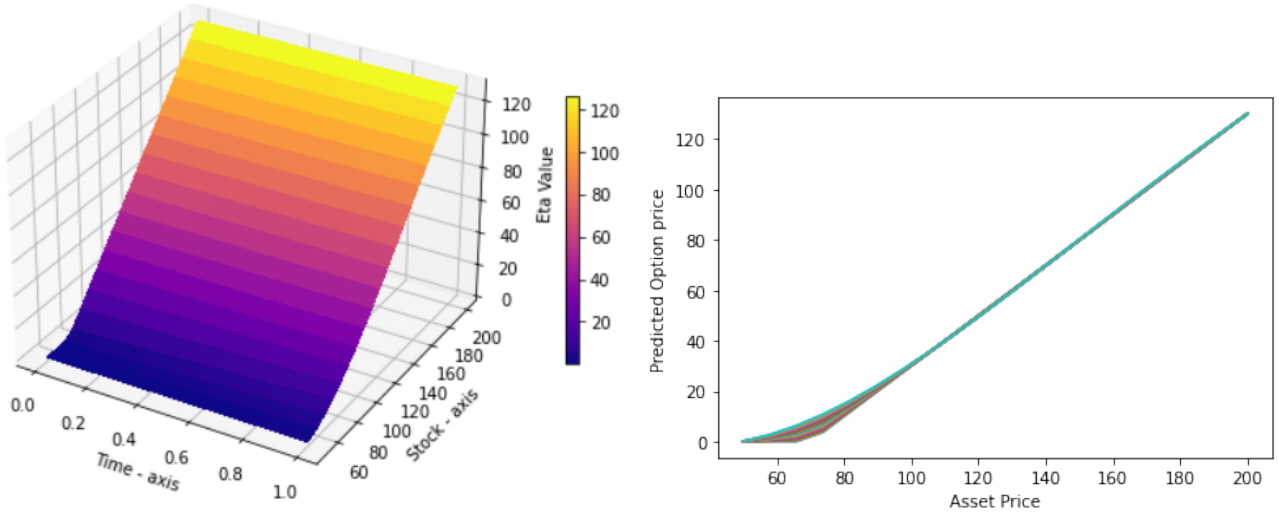


Figure 5: Loss against training time



(a) 3D-View: neural network approximation

(b) Option prices vs Asset prices

Figure 6: NN option values (20×20 grid) for different assets and time

Figures 6a and 6b give the 3-dimensional option value plot and the plot for the option value versus the asset price for the 10×10 grid size respectively. We further observed the effect of $S = K = 70$ in both figures, as they rightly captured the payoff structure, and for $S < K$, the option is out-of-the-money and worthless. It is also seen that the predicted barrier option values increased with an increase in the asset prices, thereby converging to the maximum of the underlying ($S_{\max} = 200$), as portrayed specifically in Figure 6b. Finally, Table 3 shows the results of the comparison between the exact Black-Scholes, the Monte-Carlo simulated values, and the neural network values when the grid sizes were further increased. Also, after considering a 20×20 space and time grid, we randomly chose time-grid $t = 0.368, t = 0.684$ and $t = 1$ for the computation of the NN option values and to highlight its convergence property. The neural network values (in this case, for $t = 1$) also showed promising results compared to the antithetic Monte-Carlo simulated values.

Table 3: Option values using the NN (20×20 grid) for different uniform time-grid and the MCS

Asset Value	**Exact Price**	**Neural Network Values**			**Antithetic MCS Values**
		$t = 0.368$	$t = 0.684$	$t = 1.000$	
50	0.00000	0.00000	0.00000	0.00000	0.00000
58	1.28549	1.38380	1.20805	**1.25326**	1.31271
66	3.95584	3.25977	3.57737	**3.89860**	3.79920
74	8.46066	8.79561	8.25426	**8.42306**	8.34141
82	14.50977	14.06872	14.42716	**14.49103**	14.41354
90	21.55672	21.31054	21.43981	**21.45724**	21.72772
97	28.17264	27.97890	28.04630	**28.11902**	28.45943
105	35.98245	35.27261	35.29481	**35.31392**	37.27900
113	43.90984	43.13023	43.15271	**43.15790**	44.40159
121	51.88346	50.98578	51.02626	**51.05263**	52.95742
129	59.87424	58.86531	58.91171	**58.94737**	61.50167
137	67.87111	66.76206	66.80584	**66.84211**	68.92750
145	75.87006	74.67009	74.70577	**74.73684**	76.60668
153	83.86972	82.70855	82.74461	**82.78091**	81.00258
161	91.86961	90.79569	90.84035	**90.88528**	91.99876
168	98.86957	98.82728	98.87628	**98.87628**	98.98587
176	106.86956	106.77914	106.82671	**106.87457**	107.31739
184	114.86956	114.63317	114.67254	**114.71210**	114.36625
192	122.86956	122.37645	122.40008	**122.42385**	121.99875
200	130.86955	130.00000	130.00000	**130.00000**	129.97920

4.2.4 Neural Network vs Crank-Nicolson method

This section compares results from the Crank-Nicolson FDM to the values obtained using the neural network. It is important to note that one of the disadvantages of using the Crank-Nicolson method is in the estimation of the artificial limit for the asset price, that is, $S_{\max}$. This is not the case with the neural network since a fixed value of $S_{\max} = 200$ was used throughout the implementation, serving as an edge over the use of the Crank-Nicolson method. Choosing $S_{\max}$, which is the upper bound for the computational domain of the S variable, has remained an open problem in the FDM scheme for option pricing. However, some researchers have expressed $S_{\max}$ as a a function of the asset price S[30, 36]; and others, as a function of the strike price K [for instance $S_{\max} = 2K$ [34]; $S_{\max} = 4K$ [26]; $S_{\max} = 5K$ [13]]. The wrong choice for $S_{\max}$ can result in a negative payoff, thereby leading to numerical errors and inappropriate option prices[18]. For the purpose of this research, we print results for $S_{\max} = 4K$ and $S_{\max} = 5K$ and ignore results for $S_{\max} = 2K$. This choice is due to the fact that $S_{\max} \not\leq K$, rather $S_{\max} >> K$ to avoid negative option prices[4]. Hence, we seek to place the $S_{\max}$ high enough so as not to affect the computational domain and final option values.

From Table 4, we compare barrier option values using the NN and the Crank-Nicolson numerical scheme. The values were shown for (5×5), (10×10), (20×20) asset grids for the NN with a fixed $S_{\max} = 200$; and $S_{\max}$ as a function of the strike price for the Crank-Nicolson option values. Overall, for small S and for $S_{\max} = 5K$, the option values were significantly close to the exact values. For larger values of S, the error estimates started increasing. Also, for $S_{\max} = 4K$, smaller values of S gave rise to more correct option values, and larger asset values in the domain $50 \leq S \leq 200$ led to some decrease in the option values. A remarkable feature of the zero-rebate down-and-out barrier options is that the option continues to be deep in-the-money provided that the asset prices keep increasing and so far as the barrier is not breached at any point in the life of the contract. This feature ceases to exist when we use $S_{\max} = 4K$ in the example used in this research, and one way of correcting this lag is to

[4]Consider $S_{\max} = 2K$ for $K = 70$. Then option values using the Crank-Nicolson method are -15.23418, -28.46260, -42.59637 for $S = 125, 130, 135$, respectively.

keep increasing $S_{\max}$ as large as possible, so far as the asset prices S increase. However, this is not the case with the NN, as their option values were consistent throughout the whole domain, thereby maintaining their accuracy. This behaviour further highlights the superiority of the NN over the Crank-Nicolson numerical methods in pricing the path-dependent barrier options, and the subsequent subsection analyses the error structure of the three methods.

Table 4: Option values using the NN and Crank-Nicolson method

Asset Value (5×5 Grid)	**Exact Price**	**Neural Network Values**	**Crank-Nicolson Values** $S_{\max} = 5K$	$S_{\max} = 4K$
50	0.00000	0.00000	0.00000	0.00000
87.5	19.27888	19.68020	19.28124	19.28007
125	55.87765	55.82417	55.87771	55.85888
162.5	93.36960	92.50000	93.31181	91.04683
200	130.86955	130.00000	128.41092	96.59700
Asset Value (10×10 Grid)	**Exact Price**	**Neural Network Values**	**Crank-Nicolson Values** $S_{\max} = 5K$	$S_{\max} = 4K$
50	0.00000	0.00000	0.00000	0.00000
67	4.41940	4.30772	4.42158	4.42173
83	15.34719	15.55932	15.35053	15.34808
100	31.08007	31.05410	31.08148	31.08073
117	47.89331	46.66664	47.89364	47.88916
133	63.87226	63.33341	63.87174	63.80426
150	80.86981	80.00001	80.85942	80.23592
167	97.86958	96.74896	97.76925	94.37190
183	113.86955	113.72443	113.31076	101.67852
200	130.86955	130.00000	128.41092	96.597001
Asset Value (20×20 Grid)	**Exact Price**	**Neural Network Values**	**Crank-Nicolson Values** $S_{\max} = 5K$	$S_{\max} = 4K$
50	0.00000	0.00000	0.00000	0.00000
58	1.28549	1.25326	1.28869	1.28707
66	3.95584	3.89860	3.95981	3.95805
74	8.46066	8.42306	8.46042	8.46099
82	14.50977	14.49103	14.51293	14.51103
90	21.55672	21.45724	21.55913	21.55808
97	28.17264	28.11902	28.17407	28.17324
105	35.98245	35.31392	35.98336	35.98265
113	43.90984	43.15790	43.91037	43.90821
121	51.88346	51.05263	51.88366	51.87433
129	59.87424	58.94737	59.87408	59.83774
137	67.87111	66.84211	67.86992	67.74941
145	75.87006	74.73684	75.86528	75.52038
153	83.86972	82.78091	83.85362	82.98417
161	91.86961	90.88528	91.82184	89.85601
168	98.86957	98.87628	98.75670	95.05257
176	106.86956	106.87457	106.59472	99.54524
184	114.86956	114.71210	114.25405	101.81549
192	122.86956	122.42385	121.59564	101.09163
200	130.86955	130.00000	128.41092	96.59700

4.3 Analysis of the Models

This section compares the performance of the neural network, Crank-Nicolson and the antithetic MCS. We consider the Root Mean Square Error (RMSE), also referred to as root mean square deviation, which measures the difference between the predicted values of a model and the actual observed values from the environment, which is modelled. Mathematically,

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}},$$

where the observation value is denoted by $X_{obs,i}$ and the forecast value by $X_{model,i}$.

Table 5: Root Mean Square Error analysis for different grids

			RMSE			
Asset Grid Size	**Neural Network**			**Antithetic MCS**	**Crank-Nicolson**	
	$t = 0.250$	$t = 0.750$	$t = 1.000$		$S_{\max} = 5K$	$S_{\max} = 4K$
5×5	0.905559	0.61635	0.57899	1.01446	1.09984	34.35118
	$t = 0.333$	$t = 0.667$	$t = 1.000$			
10×10	0.80407	0.74970	0.68138	0.95780	0.79795	11.55790
	$t = 0.368$	$t = 0.684$	$t = 1.000$			
20×20	0.70962	0.66214	0.63473	0.94338	0.63787	9.72751

Table 5 gives the analysis for the root-mean-square errors obtained during the pricing process when the NN and the antithetic MCS are applied in valuing the down-and-out barrier call options. The errors presented follows from the numerical result of the option values displayed in Tables 1, 2 and 3 corresponding to 5×5, 10×10 and 20×20 grid sizes respectively. Generally, from the table, we observed that the RMSE from the NN (irrespective of the grid size) were lesser than those obtained when the antithetic MCS were employed in the option valuation. It was also noted that as the grid sizes for both the asset and time increased, the RMSE kept tending to zero. Furthermore, if we keep the time constant and increase the asset's grid, the RMSE decreases, and vice versa. For the Crank-Nicolson method, the errors kept reducing with an increase in the asset grid size. For $S_{\max} = 4K$, the RMSE is huge when compared to $S_{\max} = 5K$, and the other numerical methods used in this research. This characteristic was noted due to the wrong choice of the artificial upper limit $S_{\max}$ in the valuation process. For $S_{\max} = 5K$, the RMSE were far lesser, and we can thus compare it with the NN and the Crank-Nicolson error values. The NN outperformed both the antithetic MCS and the Crank-Nicolson method, even though the CN tends to accurately price the barrier options when the value of $S_{\max}$ is carefully and correctly chosen. Thus, we can deduce that the NN method provided a highly accurate approximation of the solution of the down-and-out barrier call options.

5 Conclusion and Future Work

This research proposed a feed-forward neural network-based framework in the pricing and calibration of financial models, particularly the extended Black-Scholes model for barrier options. This framework is combined with optimisation techniques in solving the PDE-related problems of the path-dependent barrier options. Our methodology constructed a trial solution that converted a constrained optimisation issue to a simplified unconstrained problem. We focused on the down-and-out barrier options, which are options that become null and void once the underlying asset decreases and touches the threshold level. From the results, we compared the approximated values of the neural network, Crank-Nicolson values and the antithetic Monte-Carlo simulated option values to the exact Black-Scholes price, and the NN proved more accurate.

With regards to the grid points, we observed that the option values obtained using the 5×5 grid, 10×10 grid, and 20×20 grid points generally performed better than the antithetic simulated option values. It was also observed that better accuracy is obtained when the grid points become larger, as there was a significant reduction in the error function, which tended to zero faster than when a lower grid point is considered. Notwithstanding, one must be careful not to use extremely large grid points for the approximation scheme not to be computationally expensive. In conclusion, the numerical experiments we presented in this research proved that the neural network is an excellent approximation of option pricing PDE functions. The results further highlighted the accuracy and the precision of the neural network implementation in solving problems pertaining to finance.

Future research can be channelled to the use of this methodology in pricing other forms of barrier options, such as the up-and-out, down-and-in, up-and-in barrier options, as well as moving barrier options. Additionally, more work can be done to implement other forms of neural networks, such as radial basis networks and recurrent deep learning networks, in derivatives pricing, especially derivatives without closed-form solutions. More research could be done by applying the above-proposed methods to other financial models, such as the GARCH family models and the Heston pricing model.

Finally, it was noted that computational complexity does not necessarily increase with an increase in sampling points for the neural network method. This is not the case when other numerical methods, such as the finite difference and Monte-Carlo, are used in the valuation process. The computational cost quantifies the number of resources (power, number of computations, time) that the NN utilises during the training and price prediction phases. Finally, it will be worth considering the non-exponential computational complexity of the NN in solving similar problems in computational finance.

Acknowledgement

This research was sponsored by the University of Johannesburg Global Excellence and Stature (GES) 4.0. We wish to acknowledge the University of Johannesburg, South Africa.

Compliance with Ethical Standards

Conflict of Interest: Authors 1 and 2 declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] Aarts, L. P., & Van Der Veer, P. (2001). Neural network method for solving partial differential equations. *Neural Processing Letters*, 14(3), 261-271.

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, J., Levenberg, M., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[3] Anders, U., Korn, O., & Schmitt, C. (1998). Improving the pricing of options: A neural network approach. *Journal of forecasting*, 17(5-6), 369-388.

[4] Aziz, S., Dowling, M. M., Hammami, H., & Piepenbrink, A. (2019). Machine learning in finance: A topic modeling approach. *Available at SSRN 3327277*.

[5] Boyle, P. P., & Lau, S. H. (1994). Bumping up against the barrier with the binomial method. *The Journal of Derivatives*, 1(4), 6-14.

[6] Black, F. (1976). The pricing of commodity contracts. *Journal of Financial Economics* 3(1–2): 167–179.

[7] Cervera, J. G. (2019, June). Solution of the Black-Scholes equation using artificial neural networks. In *Journal of Physics: Conference Series* (Vol. 1221, No. 1, p. 012044). IOP Publishing.

[8] Chen, F., & Sutcliffe, C. (2012). Pricing and hedging short sterling options using neural networks. *Intelligent Systems in Accounting, Finance and Management*, 19(2), 128-149.

[9] Derman, E., & Kani, I. (1997). The ins and outs of barrier options: Part 2. Derivatives Quarterly, 3, 73-80.

[10] Dissanayake, M. W. M. G., & Phan-Thien, N. (1994). Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3), 195-201.

[11] Dolezel, P., Skrabanek, P., & Gago, L. (2016). Weight initialization possibilities for feedforward neural network with linear saturated activation functions. *IFAC-PapersOnLine*, 49(25), 49-54.

[12] Ganesan, N., Yu, Y., & Hientzsch, B. (2020). Pricing barrier options with DeepBSDEs. *arXiv preprint arXiv:2005.10966*.

[13] Giles, M.B. & Carter, R. (2006). Convergence analysis of Crank-Nicolson and Rannacher time-marching. *Journal of Computational Finance*, 9(4), 89 - 112.

[14] Halawa, K. (2011). 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 processing letters*, 34(3), 293-303.

[15] He, S., Reif, K., & Unbehauen, R. (2000). Multilayer neural networks for solving a class of partial differential equations. *Neural networks*, 13(3), 385-396.

[16] Hornik, K. (1991). Approximation capabilities of multilayer feed-forward networks. *Neural networks*, 4(2), 251-257. DOI:10.1016/0893-6080(91)90009-T.

[17] Hornik, K., Stinchcombe, M., & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multi-layer feed-forward networks. *Neural networks*, 3(5), 551-560. DOI:10.1016/0893-6080(90)90005-6.

[18] Hugger, J. (2006). Wellposedness of the boundary value formulation of a fixed strike Asian option. *Journal of Computational and Applied Mathematics*, 185(2), 460-481.

[19] Hull, J. (2006). *Options, futures, & other derivatives*. India: Pearson Education.

[20] Hussian, E. A., & Suhhiem, M. H. (2015). Numerical solution of partial differential equations by using modified artificial neural network. *Network and Complex Systems*, 5(6), 11-21.

[21] Khoo, Y., Lu, J., & Ying, L. (2021). Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3), 421-435.

[22] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[23] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5), 987-1000.

[24] Le, N. T., Zhu, S. P., & Lu, X. (2016). An integral equation approach for the valuation of American-style down-and-out calls with rebates. *Computers & Mathematics with Applications*, 71(2), 544–564.

[25] Liu, S., Borovykh, A., Grzelak, L. A., & Oosterlee, C. W. (2019). A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9(1), 1-28.

[26] Mashayekhi, S., & Hugger, J. (2016). Kα-Shifting, Rannacher time stepping and mesh grading in Crank-Nicolson FDM for Black-Scholes option pricing. *Communications in Mathematical Finance*, 5(1), 1-31.

[27] Morelli, M. J., Montagna, G., Nicrosini, O., Treccani, M., Farina, M., & Amato, P. (2004). Pricing financial derivatives with neural networks. *Physica A: Statistical Mechanics and its Applications*, 338(1-2), 160-165.

[28] Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1), 291-322.

[29] Niranjan, M. (1996). Sequential tracking in pricing financial options using model based and neural network approaches. *Advances in neural information processing systems*, 9, 960-966.

[30] Nwozo, C. R., & Fadugba, S. E. (2012). Some numerical methods for options valuation. *Communications in Mathematical Finance*, 1(1), 51-74.

[31] Poggio, T., & Liao, Q. (2018). Theory I: Deep networks and the curse of dimensionality. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, 66(6).

[32] Rich, D. R. (1994). The mathematical foundations of barrier option-pricing theory. Advances in futures and options research (Vol. 7). UK: JAI Press Inc. Retrieved from `SSRN:https://ssrn.com/abstract=5730`

[33] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

[34] Tagliani, A., & Milev, M. (2009). Discrete monitored barrier options by finite difference schemes. *Math. Edu. Math*, 38, 81-89.

[35] Tseng, C. H., Cheng, S. T., Wang, Y. H., & Peng, J. T. (2008). Artificial neural network model of the hybrid EGARCH volatility of the Taiwan stock index option prices. *Physica A: Statistical Mechanics and its Applications*, 387(13), 3192-3200.

[36] Umeorah, N. (2017). Pricing barrier and lookback options using finite difference numerical methods [Master's thesis, North-West University, South Africa]. Bokola Institutional Repository. `https://repository.nwu.ac.za/handle/10394/25906`.

[37] Umeorah, N., & Mashele, P. (2019). A Crank-Nicolson finite difference approach on the numerical estimation of rebate barrier option prices. *Cogent Economics & Finance*, 7(1), 1598835.

[38] Wade, B. A., Khaliq, A. Q. M., Yousuf, M., Vigo-Aguiar, J., & Deininger, R. (2007). On smoothing of the Crank–Nicolson scheme and higher order schemes for pricing barrier options. *Journal of Computational and Applied Mathematics*, 204(1), 144-158.

[39] Yadav, N., Yadav, A., & Kumar, M. (2015). An introduction to neural network methods for differential equations (pp. 13-15). Berlin: Springer.

[40] Yu, B., Xing, X., & Sudjianto, A. (2019). Deep-learning based numerical BSDE method for barrier options. *arXiv preprint arXiv:1904.05921*.