Hindawi
Complexity
Volume 2021, Article ID 5511396, 15 pages
https://doi.org/10.1155/2021/5511396

Research Article

Novel ANN Method for Solving Ordinary and Time-Fractional Black–Scholes Equation

Saeed Bajalan [1] **and Nastaran Bajalan** [2]

[1] *University of Tehran, Tehran, Iran*
[2] *Eindhoven University, Eindhoven, Netherlands*

Correspondence should be addressed to Saeed Bajalan; saeedbajalan@ut.ac.ir

Received 13 February 2021; Accepted 9 July 2021; Published 30 July 2021

Academic Editor: Chongyang Liu

The main aim of this study is to introduce a 2-layered artificial neural network (ANN) for solving the Black–Scholes partial differential equation (PDE) of either fractional or ordinary orders. Firstly, a discretization method is employed to change the model into a sequence of ordinary differential equations (ODE). Subsequently, each of these ODEs is solved with the aid of an ANN. Adam optimization is employed as the learning paradigm since it can add the foreknowledge of slowing down the process of optimization when getting close to the actual optimum solution. The model also takes advantage of fine-tuning for speeding up the process and domain mapping to confront the infinite domain issue. Finally, the accuracy, speed, and convergence of the method for solving several types of the Black–Scholes model are reported.

1. Introduction

Both PDEs of the ordinary and fractional order play an important role in pricing of financial derivatives. PDEs of the ordinary order are the basis of various models proposed for pricing of different types of options. On the contrary, financial markets show fractal behavior [1–4] and fractional PDEs (FPDE) which can better reflect that the reality of them have gained a lot of interest recently. Hence, finding an accurate and efficient approach for solving both types is a critical issue in pricing.

The most famous PDE in finance is the Black–Scholes (B-S) model, which is broadly adopted for option pricing. So far, studies have presented different approaches for finding the numerical solution of this model and its variation when the exact form does not exist [5–13]. By using tick-by-tick data, Cartea and del-Castillo-Negrete [14] found that the value of European-style options satisfies a FPDE containing a nonlocal operator in time-to-maturity known as the Caputo fractional derivative. Furthermore, in order to depict the fractal structure in the financial market, the classical B-S equation has been generalized by means of fractional derivatives which have the property of self-similarity and better modeling of long-range dependency [15–17]. Although the fractional B-S is a powerful tool for explaining the hereditary and memory characteristics of the financial market, it is considerably difficult to obtain an accurate solution for it, due to the memory trait of fractional derivatives [18]. As a result, numerous researchers have tried techniques for approximating such problems. Among different analytical models presented so far to solve time-fractional B-S, the most cited articles employed integral transform [15, 19–23], wavelet-based hybrid methods [24], the separation of variables [25], the homotopy analysis and homotopy perturbation methods [26–28], and Fourier Laplace transform [29]. However, due to the high computational complexity of these solutions, numerical methods are often better alternatives for solving such mathematical models.

Cartea and del Castillo-Negrete used backward difference technique together with shifted Grunwald–Letnikov scheme to solve the spatial fractional FMLS process [14]. Investigation of convergence analysis and comparison of the

solution of three space fractional B-S modes were provided by Marom and Momoniat [30]. Finite differences' methods were used to provide a solution for the time-fractional B-S model in 2013 by Song and Wang [31]. Zhang et al. also proposed a second-order finite difference method in the following year [32]. Moreover, in 2017, the weighted finite difference method was utilized to find the numerical solution of the model [33]. Chen et al. investigated predictor-corrector approaches for the solution of American option in 2015 [34]. Khan and Ansari [35] were the first to use the Sumudu transform to solve the fractional model of European options. In the same year, Zhang provided an unconditionally stable implicit numerical scheme for the model by changing the Riemann–Liouville derivatives to the Caputo derivative [32]. As methods based on the radial basis function (RBF) are widely used for approximating the ordinary PDEs, the meshless method was proposed for finding the solution of the time-fractional method as well [18]. In 2020, the fractional model was numerically approximated by using Quintic and sixth order B-spline functions as the basis for a collocation method providing high accuracy for the generalized B-S mode [36, 37]. In order to provide to sixth order accuracy in solving the generalized Black–Scholes model, Roul and Goura used both of the Crank–Nicolson scheme and sextic B-spline collocation method.

Regarding machine learning and ANN in particular, ANN has been traditionally used for predicting option prices [38–43]; however, to the best of our knowledge except few research studies, there is no available literature for solving the B-S differential equation by ANN [44, 45]. Even these few research studies are limited to integer order PDEs, and there is no study for the solution of the fractional order B-S. In this paper, we are going to construct a 2-layered artificial neural network (ANN) to solve the Black–Scholes model of either time-fractional or ordinary orders. The Adaptive Moment Estimation (Adam), which has been specifically created to be used by neural networks, acts as the optimizer in the ANN to add the foreknowledge of slowing down when getting close to the optimal solution. To make the process faster and increase the efficiency of the method, fine-tuning is applied to the model. Also, to overcome the problem of the infinite problem domain for approximation domain mapping is used to map the whole problem to a finite interval. The rest of paper is organized as follows: Section 2 is dedicated to problem formulation, Section 3 explains the methodology, Section 4 presents the numerical results, and finally, the conclusion is provided in Section 5.

2. Problem Formulation

A put option on an underlying asset S is said to follow a Geometric Brownian Motion (GBM), where σ, r, and W are the volatility, interest rate, and Brownian motion, respectively, if it obeys the stochastic differential equation as follows:

$$dS = rSdt + \sigma Sdw. \tag{1}$$

Using equation (1) and risk-neutral valuation formula together with the classic Feynman–Kac formula, the Black–Scholes operator is formed as below:

$$LU(S,t) = -\frac{\partial U(S,t)}{\partial t} - \frac{\sigma^2}{2}S^2\frac{\partial^2 U(S,t)}{\partial S^2} - rS\frac{\partial U(S,t)}{\partial S} + rU(S,t), \tag{2}$$

where $U(S,t)$ is the unknown function which determines the option price Robbins and Monro [46]. It has been specified that this option will have a certain payoff at a certain date in the future, depending on the value(s) taken by the stock up to that date.

It is well known now that a time-fractional Black–Scholes equation with the derivative of real order α can be obtained to describe the price of an option under several circumstances such as when the change in the underlying asset is assumed to follow a fractal transmission system. Fractional derivatives, as they are called, were introduced in option pricing in a bid to take advantage of their memory properties to capture both major jumps over small periods of time and long-range dependencies in markets. Therefore, the fractional Black–Scholes model can be formulated as follows:

$$\begin{aligned} LU(S,t) = -D^{\alpha}U(S,t) + \gamma_1\frac{\partial^2 U(S,t)}{\partial S^2} + \gamma_2\frac{\partial U(S,t)}{\partial S} \\ + \gamma_3 U(S,t) + f(S,t), \end{aligned} \tag{3}$$

where D denotes the fractional derivative, α is a real number, and γ_i, $i = 1, 2,$ and 3, and $f(S,t)$ are functions dependent on the values of σ, r, and α written using these notations for simplicity. It can be shown that the time derivative $D^{\alpha}U(S,t)$ appearing in 3 equals the α-order Caputo fractional derivative by Chen et al. [47].

Based on the type of the option, the corresponding condition set is as follows:

$$\begin{cases} U(S,T) = g(S), \\ U(a,t) = M_a(t), \\ U(b,t) = M_b(t). \end{cases} \tag{4}$$

In some cases, e.g., European, $M_b(t)$ moves toward infinity, thus the problem domain is semi-infinite. Some possible strategies are defined in Section 3.4 to overcome this obstacle when pricing. The corresponding parameters and their definitions are addressed in Table 1.

3. Methodology

In this section, four main concepts employed in the present approach are explained. Then, the necessary steps to be taken for finding the solution are combined to form the proposed method.

3.1. Time Discretization. Solving multivariable equations increases the time complexity and the risk of producing inconsistent answers by computational software. In

Table 1: Option parameters and their definitions.

Parameter	Definition
T	Maturity (years)
K	Strike price
S	Underlying asset (stock price)
r	Interest rate
D	Fractional derivative
α	Real-valued derivative order
σ	Volatility
$g(S)$	Payoff function

summary, time discretization methods are useful tools for converting such models into a series of ordinary differential equations (ODE). This approach is the result of applying finite difference methods on one dimension of an equation to approximately calculate the value of derivatives with respect to that dimension.

Since here fractional equations are also investigated, ordinary and fractional time discretization approaches are discussed. Suppose that the PDE to be solved is defined on $(S,t) \in [a,b] * [0,T]$. Based on this method, $U(S,t)$ in each time step is defined as $U_i(S) = U(S, i\Delta t)$, where $U(S,t)$ is the answer, $\Delta = (T-0)/N$, and N is the number of time steps.

3.1.1. Ordinary Time Discretization. Consider the PDE to be solved as follows:

$$\frac{\partial U(S,t)}{\partial t} = \Omega\left(\frac{\partial U(S,t)}{\partial S}, \frac{\partial^2 U(S,t)}{\partial S^2}, U(S,t), S, t\right), \tag{5}$$

where $U(S,t)$ is the answer and Ω is a linear or nonlinear function. Instead of solving this problem on the two dimensions, it can be converted to a series of dependent ODEs.

By using the defined $U_i(S)$, the following equation is constructed:

$$\frac{\partial U_i(S)}{\partial t} = \theta\left(\Omega\left(\frac{\partial U_i(S)}{\partial S}, \frac{\partial^2 U_i(S)}{\partial S^2}, U_i(S), S, i\Delta t\right)\right) + (1-\theta)\left(\Omega\left(\frac{\partial U_{i-1}(S)}{\partial S}, \frac{\partial^2 U_{i-1}(S)}{\partial S^2}, U_{i-1}(S), S, (i-1)\Delta t\right)\right). \tag{6}$$

The method is implicit, if $\theta = 1$. In this case, only posterior time step is used. The method is called explicit if $\theta = 0$ where only computing time step is utilized to approximate the solution. If the value of θ is equal to 1/2, the method is the common Crank–Nicolson method which is unconditionally stable and of the second order in time, and it uses both posterior and computing time steps for approximating the solution of the model. Due to the nonsmoothness of the payoff function and the activation functions in our ANN, the Crank–Nicolson cannot reach its second-order convergence. It can also cause extra inconsistencies because of the same problem. Hence, from this point on, $\theta = 0$ is considered.

3.1.2. Fractional Time Discretization. Suppose that the time-fractional PDE (FPDE) to be solved is as follows:

$$D_t^{\alpha} U(S,t) = \Omega\left(\frac{\partial U(S,t)}{\partial S}, \frac{\partial^2 U(S,t)}{\partial S^2}, U(S,t), S, t\right), \tag{7}$$

where α denotes the Caputo derivatives of the function and $0 < \alpha < 1$. As the first step for such FPDEs, the Caputo derivative should be discretized (readers are advised to see Hadian Rasanan et al. [48] for the preliminaries and thorough information about this type of derivative). Consider the following theorem.

Theorem 1. *Suppose that* $[0,T]$ *is divided to* N *parts with step size of* $\Delta t = T/N$, $0 < \alpha < 1$, *and* $q(t) \in C^2[0, t_k]$ *where* $t_k = k\Delta t$, *and the following holds for this interval:*

$$\left|\frac{1}{\Gamma(1-\alpha)}\int_0^{t_k} \frac{q'(t)}{(t-t_k)^{\alpha}}\Delta t - \frac{\Delta t^{-\alpha}}{\Gamma(2-\alpha)} \cdot \left[b_0 q(x,t_k) - \sum_{m=1}^{k-1}(b_{k-m-1} - b_{k-m})q(t_m) - b_{k-1}q(t_0)\right]\right| \leq \frac{1}{\Gamma(2-\alpha)}\left[\frac{1-\alpha}{12} + \frac{2^{2-\alpha}}{2-\alpha} - \left(1+2^{-\alpha}\right)\right]\max_{0\leq t\leq t_k}\left|q''(t)\right|\Delta t^{2-\alpha}, \tag{8}$$

where $b_m = (m+1)^{1-\alpha} - m^{1-\alpha}$.

Proof. See the proof in the work of Suna and Wub [49]. □

According to the above theorem, equation (3) can be discretized in the following form:

$$\mathscr{D}_t^{\alpha} U_{n+1}(S) \approx \frac{1}{\Gamma(2-\alpha)}\sum_{m=0}^{n}\frac{\widehat{U}_{n+1-j}(S) - \widehat{U}_{n-j}(S)}{\Delta t^{\alpha}} b_m = H\left[\widehat{U}_{n+1}(S)\right]. \tag{9}$$

Now, the unknown function, U_i, $i = 0, \ldots, n$, should be approximated in each time step such that it satisfies equation (3) and also its initial and boundary conditions in equation (4). In this regard, the boundary conditions can be satisfied by considering $U_N(S) = g(S)$ in computations. On the contrary, to satisfy the boundary condition, the sum of least square error methods is used in Section 4.

The remaining step is satisfying equation (3), so for this purpose, the cost function is chosen as below:

$$\text{Cost}(S,W) = \frac{1}{2Nr}\sum_{i=1}^{r}\sum_{n=0}^{N-1}\left[\frac{1}{\Gamma(2-\alpha)}\sum_{m=0}^{n}\frac{\widehat{U}(S_i, t_{n+1-m}) - \widehat{U}(S_m, t_{n-m})}{\mathrm{d}t^{\alpha}} b_m - H\left[\widehat{U}(S_i, t_{n+1})\right]\right]^2, \tag{10}$$

where $S = (S_1, S_2, \ldots, S_r)$ and S_i is the ith training data. To find the optimum weights for the network, this cost function should be minimized subject to W, so the following nonlinear least square problem is obtained:

$$\min_{W} \frac{1}{2Nr} \sum_{i=1}^{r} \sum_{n=0}^{N-1} \left[\frac{1}{\Gamma(2-\alpha)} \sum_{m=0}^{n} \frac{\widehat{U}_{n+1-m}(S_i) - \widehat{U}_{n-m}(S_m)}{dt^{\alpha}} b_m - H\left[\widehat{U}_{n+1}(S_i)\right] \right]^2. \tag{11}$$

It is noteworthy that when $\alpha = 1$, the ordinary time discretization method will be used.

It should be noted that the method is based on the discretization of Caputo-type derivatives, and the result based on other types is only possible if the discretization is possible for other types.

3.2. Function Approximation. According to the universal approximation theorem, every continuous function can be approximated by a feedforward neural network [50]. This theorem states that any linear function can be approximated by an ANN without any hidden layers. But for functions of higher orders, the approximation can be well-established, if the ANN has at least one hidden layer. To calculate the price of an option based on equations (2) and (3), the value of option at each time step using a 2-layered network should be approximated as follows:

$$N(W, S) = \Psi\left(V\varphi\left(WS + B_0\right) + B_1\right), \tag{12}$$

where n is the number of neurons in the hidden layer, $B_0 = \{b_1, b_2, \ldots, b_n\}$, $B_1 = \{\beta_1\}$, and φ_i, $i \in 1, 2, \ldots, n$, are the activation functions in the hidden layer, and Ψ is the activation function for the output layer. The above formula can be seen in Figure 1.

The nodes commensurate with the edges b_i and β_1 in this network are called the biases whose inputs are unchangeable 1s, and their weights are additional parameters as means of adjusting the output of the next layer. In other words, they help the network fit best for the given data.

The most famous activation function in deep learning and neural networks is sigmoid.The two main reasons for the broad application of the sigmoid function are the straight-forward calculations of its derivatives and its bounded value. The sigmoid function is defined as follows, and its values are in $[0, 1]$:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}. \tag{13}$$

These features make it a perfect candidate for problems that produce probabilities and for the Black–Scholes model. The values of options are nonnegative, and the effect of other parameters will not increase the calculations as they are multiplied by smaller values produced by sigmoid. On the contrary, the derivative of this well-known activator, its slope, is easily calculable between any two arbitrary points:

$$\text{sigmoid}'(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2}. \tag{14}$$

Although linear functions such as identity are not favorable for hidden layers as they take away the chance to generalize and adapt from the network, it is possible to use them as the activator of the output layer; hence, the hidden layers are present and are directly interacting with inputs.

3.3. Fine-Tuning. One of the key factors in the present study is the possibility of applying the fine-tuning methods during the training process. Fine-tuning is employing a previously trained neural network to find the solution of a new similar task. This process is normally applied to datasets related to images and voices. However, following the same approach, it is possible to increase the accuracy and speed of the network in this work.

Building and validating an ANN from scratch can be a huge task in its own right, depending on what data being trained on it, and many parameters such as the number of layers and the number of nodes in hidden layers, the proper activation functions, and learning rate should be found through trial and error. If a trained model that already does one task well exist and that task is similar to ours in at least some remote way, then everything the model has already learned can be taken advantage of and applied to the new specific task. If the task is completely similar, like what we are facing when solving the problem at different time steps, the exact weights can be used as the initial values. If the models are somewhat similar, still some knowledge exists on the previous network which is notable for speeding up the process of building/modifying and training the network for the new task. Then, the only job remains for the network is learning the new features and properties that were not available in the former task. Here, once the network is trained for the first time steps, the obtained parameters, weights, and biases can be effectively reemployed for training the data fed to the network in other time steps. The approximated solution and its convergence rate are compared in Section 5.

3.4. Domain Mapping. Considering the vulnerability of neural networks due to the bounded domain of their activation functions in calculations on infinite domains, the domain mapping approach is utilized to shift the problem from its semi-infinite domain to a finite interval. This helps to prevent the error caused by common solutions such as truncation of the domain.

On finite domains, $S = x$ will be considered, but in semi-infinite domains, transformation formulas should be used for shifting the problem to a desired finite one. Here, $x(S) = (2/\pi)\arctan(S/L)$ is used to shift the problem's domain

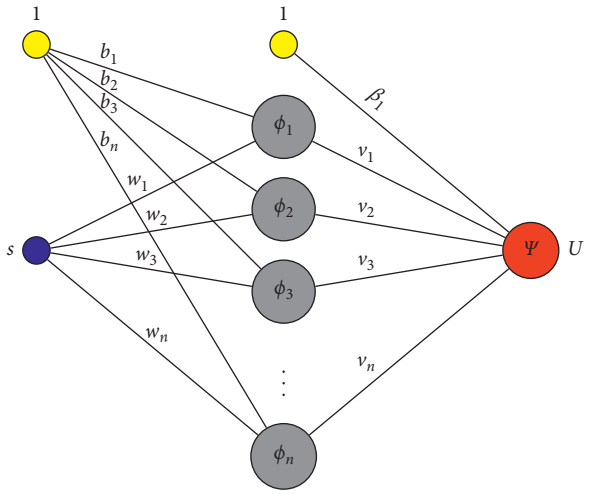

FIGURE 1: The topology of the network used for solving the Black–Scholes model.

which is $[0, \infty)$ to $[0, 1]$, in which L is the characteristic length of the mapping Boyd [51].

Here, L is chosen in a way that 60% of all training points stand before the mapped strike price because the price significantly differs from zero in $[0, K]$ (Rad et al. [52]). It means that, by defining l as an indicator for 0.6, in the view of the fact that these points are equidistant, L is computed as follows:

$$L = \frac{K}{\tan((\pi/2)l)}. \tag{15}$$

First, let us introduce the following notations:

$$\begin{aligned} U(S,t) &= \tilde{U}(x,t), \\ S &= L\tan\left(\frac{\pi}{2}x\right), \\ \Upsilon &\triangleq \frac{\partial S}{\partial x} = \frac{L\pi}{2\cos^2((\pi/2)x)}, \\ \Theta &\triangleq \frac{\partial \Upsilon^{-1}}{\partial x} = -\frac{2\cos((\pi/2)x)\sin((\pi/2)x)}{L}. \end{aligned} \tag{16}$$

Hence, the derivatives needed for the calculations according to Section 2 are

$$\begin{aligned} \frac{\partial U(S,t)}{\partial S} &= \frac{\partial \tilde{U}(x,t)}{\partial x}\frac{\partial x}{\partial S} = \frac{1}{\Upsilon}\frac{\partial \tilde{U}(x,t)}{\partial x}, \\ \frac{\partial^2 U(S,t)}{\partial S^2} &= \frac{\partial}{\partial S}\left(\frac{\partial U(S,t)}{\partial S}\right) = \frac{1}{\Upsilon^2}\frac{\partial^2 \hat{U}(x,t)}{\partial x^2} + \frac{\Theta}{\Upsilon}\frac{\partial \hat{U}(x,t)}{\partial x}. \end{aligned} \tag{17}$$

By substituting equation (17) in equations (3) and (2), the domain of the obtained Black–Scholes model is $[0, 1]$.

Since the transformation is applied to the whole problem, the payoff function and boundary conditions of equation (4) should be changed as well.

3.4.1. Discussion. Using domain mapping helps to approximate the answer on the whole interval of the problems. On the contrary, the number of training data points needed for solving the model on smaller intervals is significantly less as can be seen in Section 5. However, it can decrease the accuracy since the whole domain is being compacted into one small domain and loss of information may occur, meaning that truncating the domain causes a perfect approximation on a subdomain of problems but with acceptance of a bit of loss in accuracy, the whole domain can be covered. So, there lays a trade-off between these two methods.

3.5. Adaptive Moment Estimation Learning. Regression modeling is used to determine coefficients of mathematical functions based on empirical data. The method of least squares determines the coefficients such that the sum of the squares of the deviations between the data and the curve fit is minimized. Finding a satisfactory solution to nonlinear least square problems is one of the famous topics among scientists who work on nonlinear systems of equations. For minimizing a vector function, $\|\Lambda(x)\|$, that is, matrix A is defined as $\Lambda: \mathbb{R}^n \longrightarrow \mathbb{R}^m$, and $m \geq n$ with respect to a predefined $x = (x_1, x_2, \ldots, x_n)$, that is to say, $x^* \in \mathbb{R}^n$ is found in a way that

$$\Upsilon(x) = \frac{1}{2}\sum_{i=1}^{m}(\Lambda_i(x))^2 = \frac{1}{2}\|\Lambda(x)\|^2 = \frac{1}{2}f^T(x)\cdot f(x), \tag{18}$$

in which

$$x^* = \min_x\{\Upsilon(x)\}. \tag{19}$$

Several methods have been introduced for solving this nonlinear least square model so far. As we can see the same vector can be constructed when solving differential equations, similarly, for the Black–Scholes models, in each time step, the final goal is finding the proper network weights for solving the following system of equations:

$$\begin{aligned} \text{eq}_1 &= U(S,t_i) - \hat{U}(S,t_i) = 0, \quad i = 0,\ldots,M, \\ \text{eq}_2 &= \hat{U}(0,t_i) - f(S) = 0, \quad i = 0,\ldots,M, \\ \text{eq}_3 &= \lim_{x\longrightarrow\text{boundary}}\left(\hat{U}(S,t_i) - g(S)\right) = 0, \quad i = 0,\ldots,M. \end{aligned} \tag{20}$$

The best possible solution to this system of equations is calculated when the sum of the squares of these equations gets smaller. Therefore, the problem is converted into an optimization problem. By minimizing the following equation, the appropriate weights and biases for our network are found:

$$\text{objective}(\hat{U}(S,t)) = \min_{W,V,B_1,B_2}\text{eq}_1^2 + \text{eq}_2^2 + \text{eq}_3^2. \tag{21}$$

Gradient descent (GD) is the most famous iterative algorithm employed as a learning paradigm to solve regression problems. In GD, after initializing the weights, the

gradients, G, of the cost function is calculated. The cost function is the sum square error of the output based on the desired output for each member of the training dataset. Then, based on G, the weights become updated $W = W - \eta G$. This process is repeated by considering the new values as the initial ones until the cost function is desirably minimized. η is known as the learning parameter and is used to balance the rate of increase or decrease in each iteration. It should be noted that the only constraint on this value is $0 < \eta < 1$. Various methods and theorems are introduced to find the boundaries of this value according to the problem. Some suggest using a big value and decreasing its value as the result approaches the correct value. In contrast, some others suggest choosing a very small value and then increasing it exponentially in time when the correct direction is found. But generally, they all prefer making this parameter a function of time, while none of them propose a single formula to calculate the exact amount of it for the best approximation [46, 53, 54].

GD family has different optimizers such as Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), Root Mean Square Propagation (RMSprop), and Nesterov Accelerated Gradient (NAG) which are mostly used in deep learning because of their speed and strength according to various control parameters such as the size of the training datasets and the pattern in which the training data is scattered.

In GD for updating only one parameter, all available samples in the dataset should be visited; however, in SGD [55], minibatches, which are small subsets of the whole dataset, are used to update a single parameter. For relatively large datasets, this causes the algorithm to converge faster. GD is an actual optimizer trying to find the exact gradients while in SGD; as explained, the algorithm only approximates the gradients and not the precise value. Since SGD fluctuates a lot, due to frequent updates with high variance, it shows a paradoxical behavior. It can explore new and potential directions to find the minimum but, at the same time, this behavior puts the network in danger of completely missing the local or global minimum.

A solution was proposed by the father of propagation, Geoffrey Hinton Tieleman and Hinton [56]. This thorough study which has not been academically submitted or published gained a lot of attention. The proposed algorithm fights the possibilities of vanishing or exploding the gradients. In other words, RMSprop normalizes the gradient using a moving average of squared gradients. This normalization balances the step size, reducing the step size for large gradients to avoid exploding and increasing it for small ones to avoid vanishing. Since this approach uses the exponentially decaying average, it is related to the most recent gradients, so the past gradient would not play a great role in updating the parameters. This leads to slow changes in the learning rate; however, it is relatively faster than GD.

So far, it can be seen that RMSProp and SGD are the best options. Adam is an adaptive algorithm which is generally considered as the combination of these two paradigms with momentum. This methodology has been specifically created to be used by neural networks.

Like RMSprop, Adam employs squared gradients to modify the learning rate. Also, the first and second moments are utilized using the moving average of the gradient such as SGD. However, a specific learning rate is calculated for each network parameter (weights) using two hyperparameters. Here, a summary of how Adam optimization works for the present model is provided. For the complete explanation, readers are encouraged to study Kingma and Ba [57]. The convergence of the method has been described in several great papers. But finally, all of the studies confirm the convergence proof provided in the first papers [58].

All computations are done using autograd package of Python. In this package, Adam optimization is implemented as follows:

```
def adam (grad, w, callback = None, num_iters = 100,
step_size = 0.001, b1 = 0.9, b2 = 0.999, eps = 10** −8):
m = np · zeros (len (w))
v = np · zeros (len (w))
for i in range (num_iters):
g = grad (w, i)
if callback: callback (w, i, g)
m = (1 − b1) * g + b1 * m # First moment estimate.
v = (1 − b2) * (g * * 2) + b2 * v # Second moment estimate.
mhat = m/ (1 − b1 * * (i + 1)) # Bias correction.
vhat = v/ (1 − b2 * * (i + 1))
x = x − step_size * (mhat/ (np · sqrt (vhat) + eps)).
return x
```

As the name of the method describes, it is derived from adaptive moment estimation. nth moment of a random variable is defined as the expected value of that variable to the power of n:

$$m_n = E\left[w^n\right], \tag{22}$$

where m shows the nth moment and w is a random variable. The gradient of the cost function of the neural network can be considered a random variable since it usually evaluated on some small random batch of data. The first moment is mean, and the second moment is uncentered variance. To estimate the moments, Adam utilizes exponentially moving averages and computed on the gradient evaluated on a current minibatch:

$$\begin{aligned} m_i &= \beta_1 m_{i-1} + \left(1 - \beta_1\right) g_i, \\ v_i &= \beta_2 v_{i-1} + \left(1 - \beta_2\right) g_i^2, \end{aligned} \tag{23}$$

where m and v denote the moving averages and g is the gradient of the current data presented to the network. According to Kingma and Ba [57], which is also mentioned in the above snippet from autograd package, the values of hyperparameters β_1 and β_2 have two default values of 0.9 and 0.999, respectively. While the authors did not discuss the choosing process of these two variables, all studies reported very promising and, in most cases, perfect estimations using these two default values (see also [59]). The vectors of moving averages are initialized with zeros at the first iteration.

The remaining problem with these moments was being biased towards zero since m_i and v_i are initialized as vectors of 0's. In other words, especially during the initial epochs and when the decay rates are small (i.e., β_1 and β_2 are close to 1), the values of m_i and v_i will not change significantly or even at all. So, the authors proposed the following bias corrections in order to surmount this obstacle:

$$\widehat{m}_i = \frac{m}{1-\beta_1^i}, \qquad \widehat{v}_i = \frac{v}{1-\beta_2^i}. \tag{24}$$

Now, for each of the parameters (weights), a specific updating rule can be created:

$$w = w - \eta \frac{\widehat{m}}{\sqrt{\widehat{v}} + \epsilon}, \tag{25}$$

where ϵ is an control parameter preventing the fractional part from producing a division by zero error.

Different scientific studies have shown that Adam outperforms other methods. According to empirical practices, this method has better performance and accuracy. This is also discussed in Section 5. One problem that is stated by many studies is the convergence of the method. However, Kingma and Ba [57] provided the analysis for the convex problems; other papers argued the convergence of the method on a few nonconvex problems. And, with some modification, they finally agreed on its usability.

In [60], the full analysis of the convexity of the Black–Scholes model is proposed. Due to differences such as the failure of put-call parity in real markets instead of theory, this paper proves that, for all American options, they preserve their convexity in bubbled markets as well as nonbubbled ones. They showed that European options are convexity preserving only for bounded payoffs. Thus, in this respect, the prices of American options are more robust than their European counterparts. In the same study, it is shown that models for bubbles are convexity preserving for bounded contracts. More precisely, consider $(x,t) \in [0,\infty) \times [0,T]$, and let $u1(x,t)$ and $u2(x,t)$ be the option prices such that their corresponding volatilities are nonnegative α_1 and α_2 which satisfy $\alpha_1(x,t) \le \alpha_2(x,t)$.

Theorem 2. *Assume that g is concave. Then, $u(x,t)$ is concave in x for any $t \in [0,T]$. Moreover, the option price is decreasing in the volatility, that is, $u1(x,t) \ge u2(x,t)$ for all $(x,t) \in [0,\infty) \times [0,T]$. Similarly, if g is convex and bounded, then $u(x,t)$ is convex in x for any $t \in [0,T]$. Moreover, the option price is increasing in the volatility, that is, $u1(x,t) \le u2(x,t)$ for all $(x,t) \in [0,\infty) \times [0,T]$.*

The full proof is available in [60]. As they mentioned, the proof is valid under the assumption of the uniqueness of the result for such an option, which is proved in their thorough study on properties of Black–Scholes models in more realistic markets as well.

4. Numerical Results and Discussion

In this section, three test examples with exact solutions are chosen according to the previous works for examining the accuracy and efficiency of the proposed ANN. According to what is mentioned in Section 3.5, this approach is applicable to other kinds of options such as barriers and American options. All computations are performed using Python.3.7 software on a 2.7 GHz Intel Core i7 CPU machine with 16 GB of memory. Only one hidden layer with 20 hidden neurons is used in all of the samples. The initial weights are scattered in $[-0.01, 0.01]$. The number of epochs for the first time stamp is 5000, and for the rest of the steps, thanks to fine-tuning, decreasing this number to 1200 provides very promising results. All values in $[2000, 7000]$ for the first time-step iterations and all values in $[600, 7000]$ do not lead to overfitting/underfitting, but the best results in our experiment achieved using the stated values. If the learning rate is low, then training is more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are small, and on the contrary, if the learning rate is high, then training may not converge or may even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse. In this research, best values for the learning rate are found using the genetic algorithm.

Example 1. Let us consider a European call option, in which its interest rate, volatility, and strike price are 0.05, 0.2, and 10, respectively. The governing equation is similar to equation (31), and the boundary conditions set is as follows:

$$\begin{cases} U(S,T) = \max(S-K,0), \\ U(0,t) = 0, \\ \lim\limits_{S\longrightarrow\infty} U(S,t) = S - K\,\exp(-rt). \end{cases} \tag{26}$$

With the maturity of 1, in years, the approximate price at $t = 0$ is shown in Figures 2(a) and 2(b). The exact solution of this call option can be obtained using the following analytical solution denoted by $U_{\text{exact}}(S,t)$:

$$d_1 = \frac{\ln(S/K) + \left(r + (1/2)\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, \qquad d_2 = d_1 - \sigma\sqrt{T-t}, \qquad N(S) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{S} \exp\left(-\frac{1}{2}y^2\right) \mathrm{d}y, \qquad U_{\text{exact}}(S,t) = SN(d_1) - K\,\exp(-r(T-t))N(d_2). \tag{27}$$

When truncating the domain at 15, the cardinal of the training dataset is 150 containing equidistant points scattered between $[0, 15]$ and $\eta = 0.03$. The number of time steps, N, is 20, and the average calculation time per each epoch on the abovementioned configuration is 0.098 s. Figure 2(a) demonstrates that when the problem domain is truncated, the approximate price is behaving fine until it reaches the truncation point which is 15 in these examples.

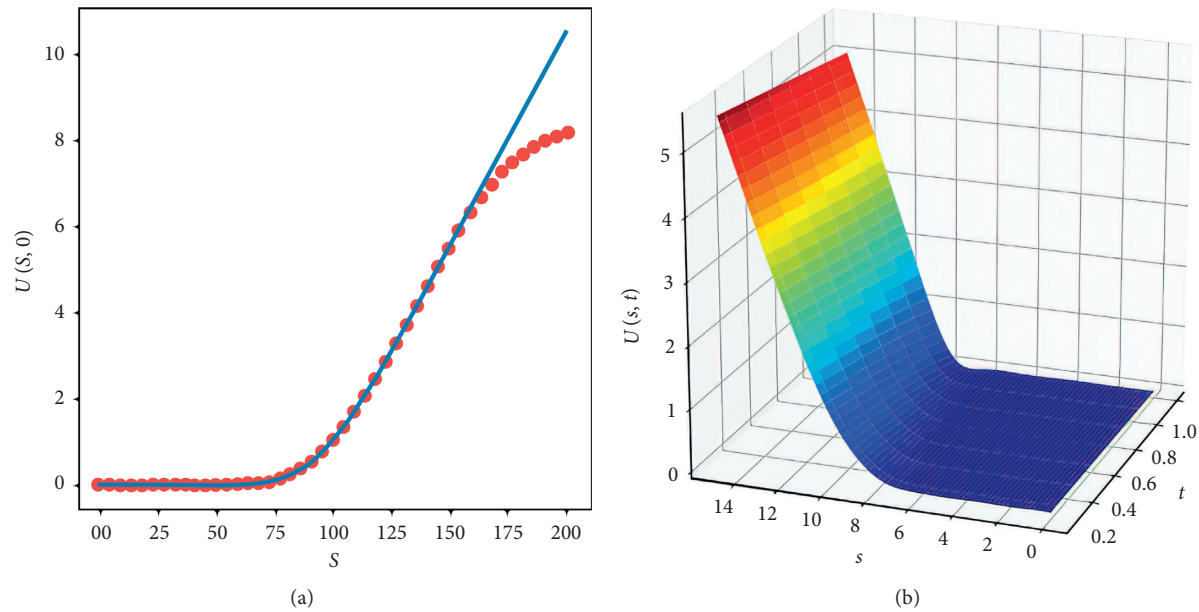

FIGURE 2: Plots of the approximated solutions of Example 1 when the number of hidden neurons is 20 and dt = 1/20. (a) $U(S,0)$. (b) $U(S,t)$, $t \in [0,T]$.

To solve this issue, the problems are mapped to $[0,1]$ using Section 3.4; then, $U(x,t)$ is computed using the proposed ANN and sigmoid functions as the activation functions; then, $U(x,t)$ is reverted to the original model's domain using the inverse mapping function so that $U(S,t)$ is calculated. Only 10 equidistant points are used as training points in this case, and the logarithmic absolute errors obtained from two approaches are compared in Figure 3. It should be noted that, after the truncation point, the error increases rapidly for the first approach, but when truncating the domain, the overall error is higher at the beginning of the interval but it remains steady and even falls at the end of the domain. Since the mapping function converges to infinity on $x = 1$, the numerical calculation on software such as Python will not be able to perform the calculations. So, these comparisons are done using a very big value for $x = 0.9999999$. Figure 4 confirms the fact that the Adam optimizer performs better than the other two optimizers, as it starts to converge and moves towards the answer in earlier epochs for the first time step. The average calculation time for SGD and RMSprop are, respectively, 0.45 s and 0.63 s per epoch. It is noteworthy that RMSprop crashed due to overflow encounters, and the depicted figure is just for comparing, with the learning rate of 0.01 instead of 0.03 which somehow might make the comparison unreliable. But the point is observed, and this method fails in comparison to the other methods for solving this type of the Black–Scholes model.

Figures 5(a) and 5(b) illustrate the incredible influence of fine-tuning on the objective convergence.

Example 2. Consider the following fractional model of a European option with homogeneous boundary conditions as follows:

$$D^{\alpha}U(s,t) = a\frac{\partial^2 U(s,t)}{\partial t^2} + b\frac{\partial U(s,t)}{\partial t} - cU(s,t) + f(S,t) = 0,$$
$$\begin{cases} U(0,t) = t, \\ U(1,t) = 0, \\ U(S,0) = S^2(S-1). \end{cases} \tag{28}$$

The interest rate and volatility are 0.05 and 0.25, respectively. Other variables are calculated based on these two variable, $a = (1/2)\text{sigma}^2$, $b = r - a$, and $c = r$. The fractional order of the equation is $\alpha = 0.7$. Also,

$$f(S,t) = \left(\frac{2t^{2-\alpha}}{\Gamma(3-\alpha)} + \frac{2t^{1-\alpha}}{\Gamma(2-\alpha)}\right)S^2(1-S) - (t+1)^2 \left[a(2-6S) + b\left(2S - 3S^2\right) - cS^2(1-S)\right]. \tag{29}$$

The exact solution for this equation is formulated as below:

$$U_{\text{exact}}(S,t) = (t+1)^2 S^2 (1-S). \tag{30}$$

The number of time steps, N, is 10, the number of hidden neurons in this example is 6, and the average calculation time per each epoch on the abovementioned configuration is 0.0012 s. Also, $\eta = 0.03$. The approximate solution calculated using the Adam neural network is plotted in Figure 6. However, smaller number of training data points, 20 and 40, still produced errors in $[0, 10^{-2}]$, and 60 equidistant points are used as training points in this case to reach the logarithmic absolute errors illustrated in Figure 7(a), giving us the freedom to increase the accuracy even more. Because the number of epochs in this example is very small, the absolute errors obtained from SGD, RMSprop, and Adam are

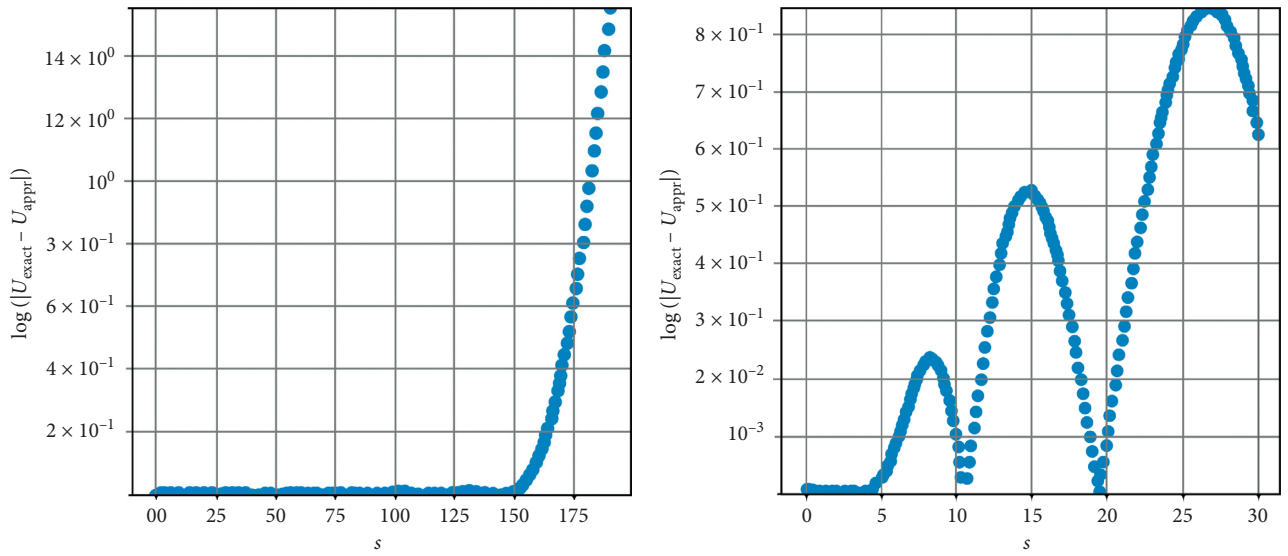

FIGURE 3: Plots of the logarithmic absolute error for the solution of Example 1 when the number of hidden neurons is 20 and dt = 1/20, the logarithmic error (a) using the truncating approach and (b) using the mapping function.

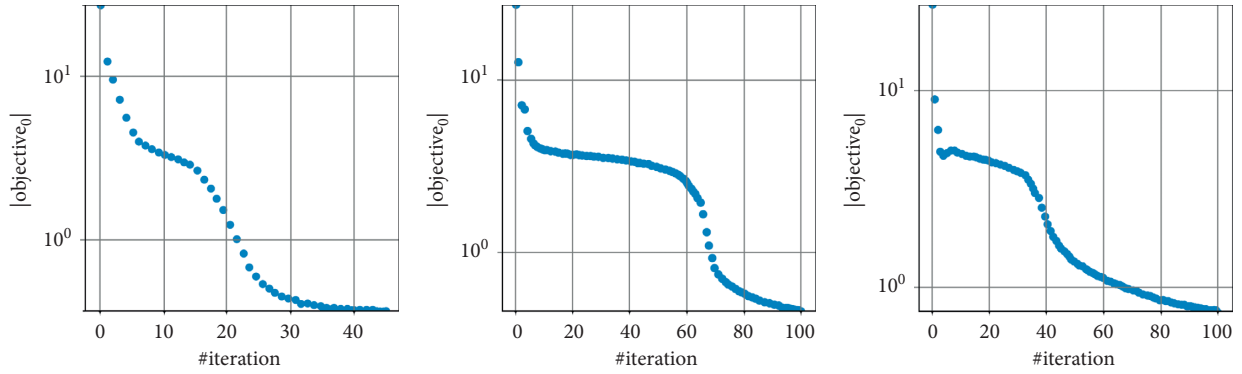

FIGURE 4: Comparison of networks' convergence toward the exact solution of Example 1 using (a) Adam optimizer, (b) SGD optimizer, and (c) RMSprop (η = 0.01)

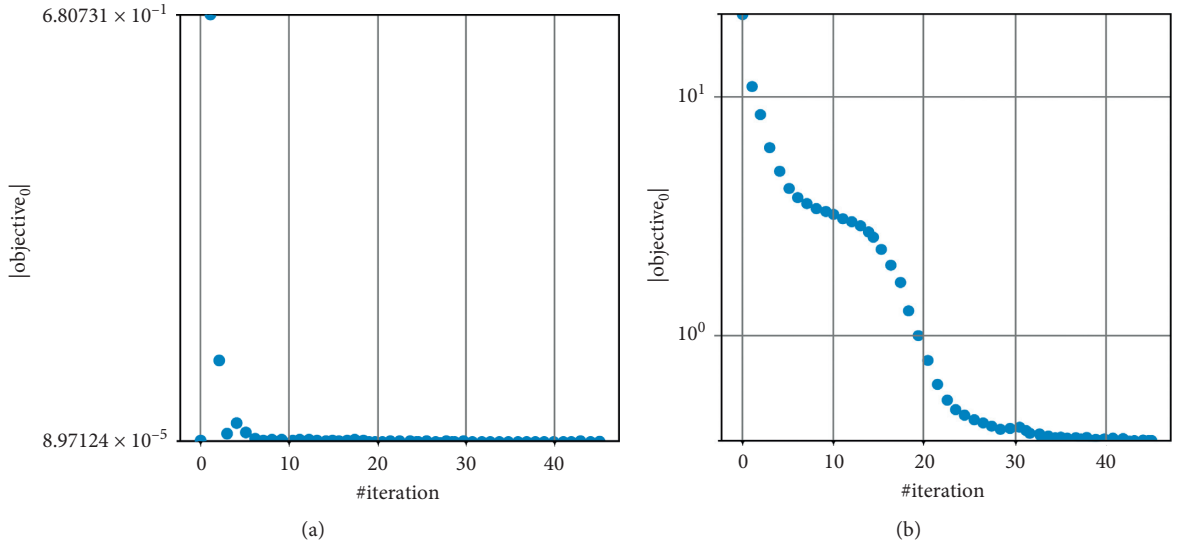

FIGURE 5: The effect of fine-tuning on the convergence rate of the method. Solving Example 1 with (a) fine-tuning and (b) without fine-tuning.

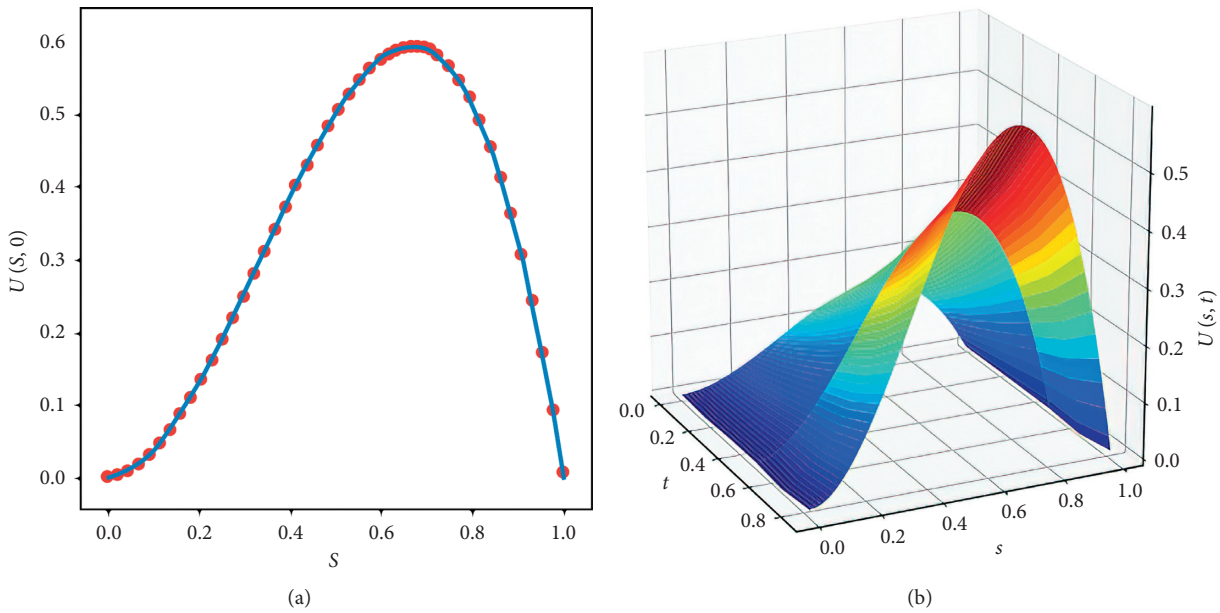

Figure 6: Plots of the approximated solutions of Example 2 when the number of hidden neurons is 60 and dt = 1/20. (a) $U(S,0)$ (the red dotted plot shows the approximate solution). (b) $U(S,t)$, $t \in [0,T]$.

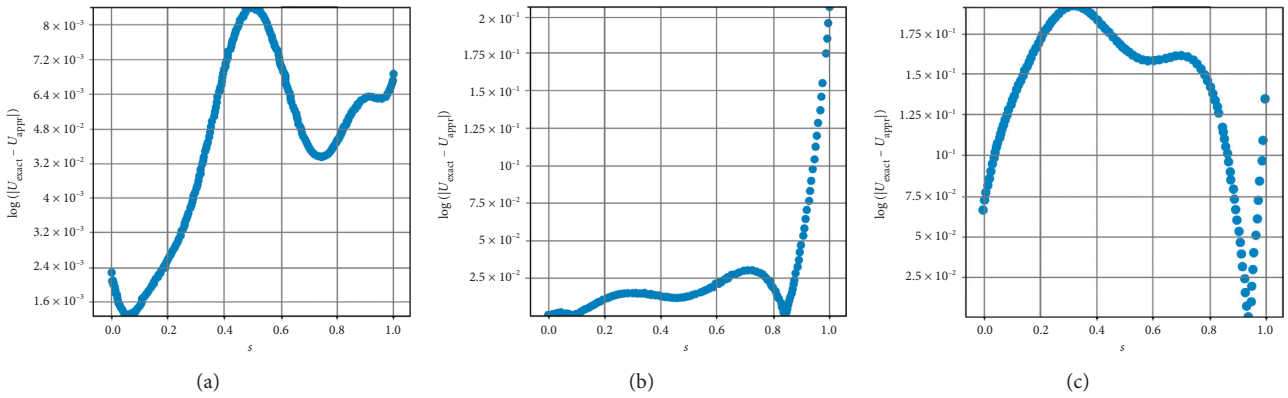

Figure 7: Comparison of networks' convergence toward the exact solution of Example 2 using (a) Adam optimizer, (b) SGD optimizer, and (c) RMSprop.

compared in Figure 7. Here, it can be seen that, with a small number of neurons and training points, the accuracy of the model is more promising than the other optimizers. The average calculation time for SGD and RMSprop are, respectively, 0.0059 s and 0.17 s per epoch.

Figure 8 illustrates the incredible influence of fine-tuning on the objective convergence. When fine-tuning is used, the objective function starts with a very small value, and hence, it converges rapidly even for very small values of the cost (10^{-4}).

Example 3. Let us consider a European put option, in which its interest rate, volatility, and strike price are 0.05, 0.2, and 10, respectively:

$$LU(S,t) = \frac{\partial U(S,t)}{\partial t} - \frac{\sigma^2}{2}S^2\frac{\partial^2 U(S,t)}{\partial S^2} - rS\frac{\partial U(S,t)}{\partial S} + rU(S,t), \tag{31}$$

$$\begin{cases} V(S,T) = \max(K-S,0), \\ V(0,t) = K\,\exp(-rt), \\ \lim\limits_{S\longrightarrow\infty} V(S,t) = 0. \end{cases} \tag{32}$$

With the maturity of 1, in years, the achieved result is shown in Figure 9. The exact solution of this put option can be obtained using the following analytical solution denoted by $U_{\text{exact}}(S,t)$:

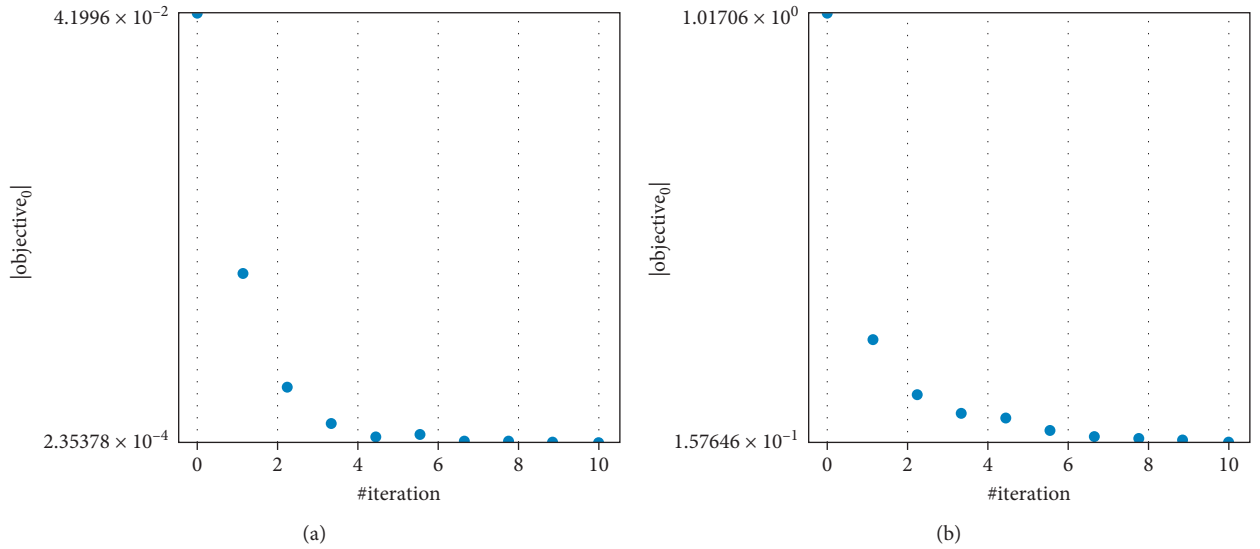

FIGURE 8: The effect of fine-tuning on the convergence rate of the method. Solving Example 2 with (a) fine-tuning and (b) without fine-tuning.

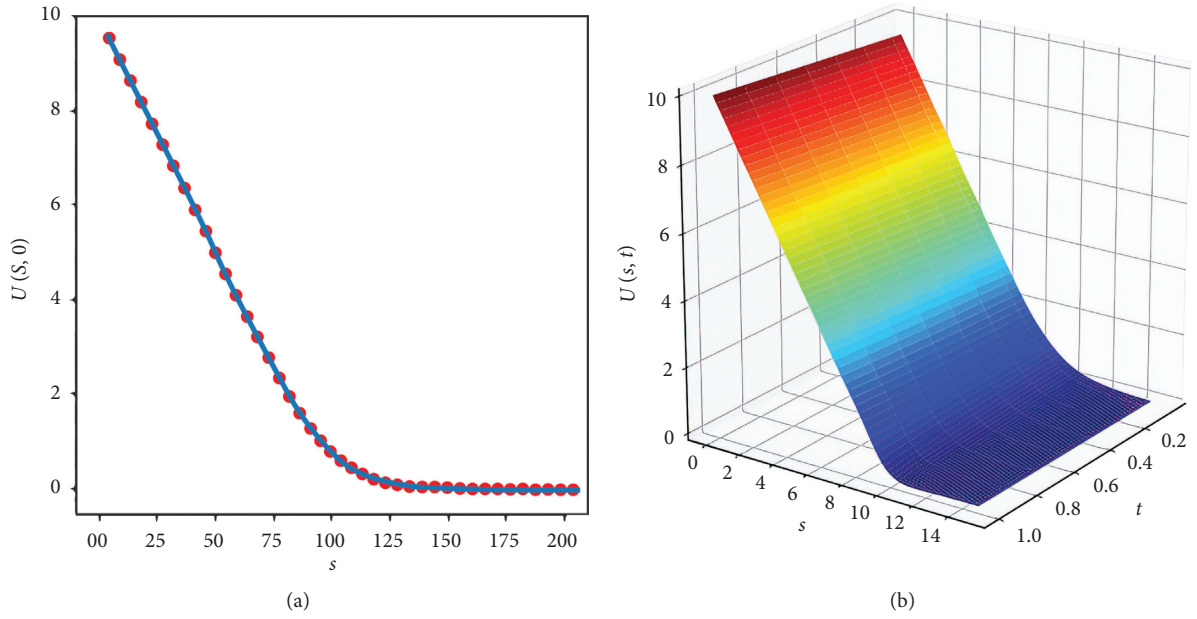

FIGURE 9: Plots of the approximated solutions of Example 3 when the number of hidden neurons is 60 and dt = 1/20. (a) $U(S,0)$ (the red dotted plot is the approximate solution). (b) $U(S,t)$, $t \in [0,T]$.

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

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

$$N(S) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{S} \exp\left(-\frac{1}{2}y^2\right)\mathrm{d}y,$$

$$U_{\text{exact}}(S,t) = -SN(-d_1) + K\,\exp(-r(T-t))N(-d_2). \tag{33}$$

Since the problem domain is unbounded according to the boundary conditions, when truncating the domain at 15, the cardinal of the training dataset is 110, containing the equidistant point scattered between $[0, 15]$ and $\eta = 0.2$. The number of time steps, N, is 10, and the average calculation time per each epoch on the abovementioned configuration is 0.032 s. Increasing the number of data points will increase the accuracy for this configuration slightly (other parameters might need to be adjusted as well); however, we preferred this size to reduce complexity and memory usage.

In Figure 9(a), it is shown that when the problem domain is truncated, unlike Example 3, the approximate price is

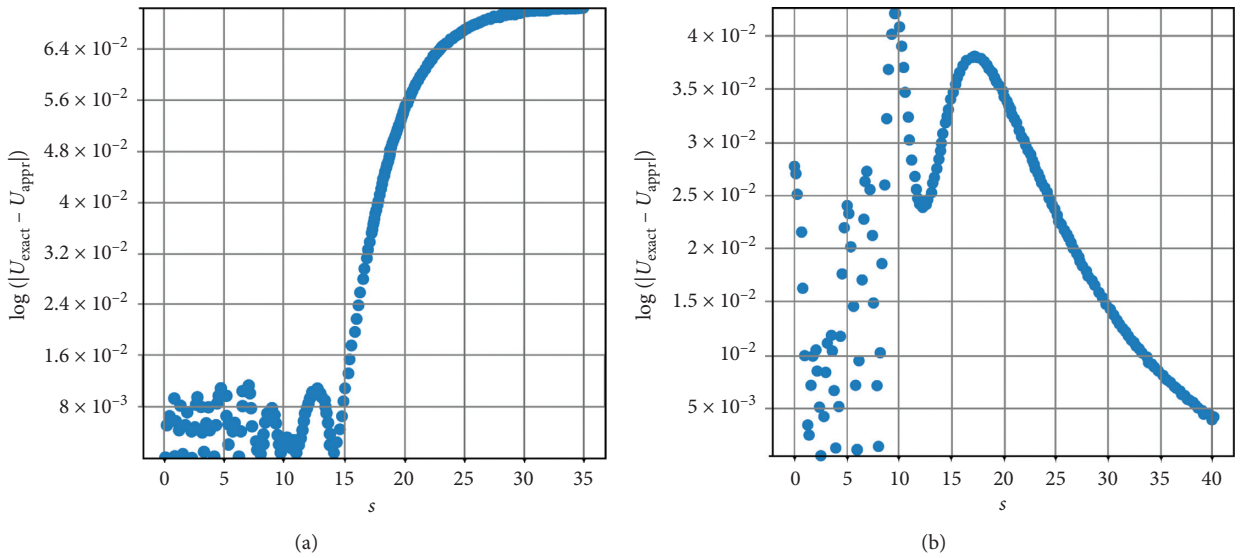

FIGURE 10: Plots of the convergence and logarithmic absolute error for the solution of Example 3 when the number of hidden neurons is 60 and $dt = 1/30$. (a) Cost function in each ANN iteration. (b) $|U_{\text{exact}}(S,T) - U_{\text{app}}(S,0)|$.

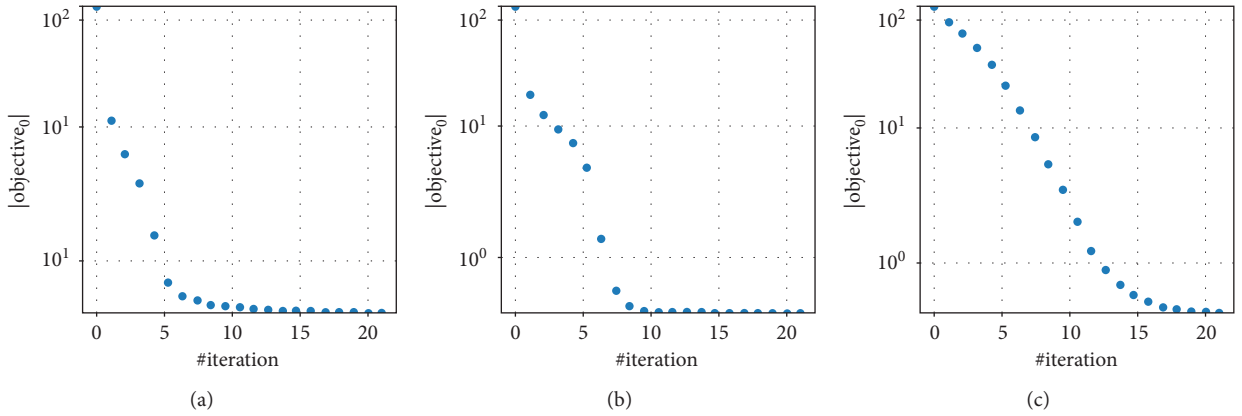

FIGURE 11: Comparison of networks' convergence toward the exact solution of Example 3 using (a) Adam optimizer, (b) SGD optimizer, and (c) RMSprop.

behaving fine throughout the whole unbounded interval. But this does not state that this behavior is the expected behavior of the option considering that the boundary condition makes the option price move towards zero. To make it clearer, the errors for truncated and mapped approximate solutions are compared in Figure 10. In Figure 10(a), the absolute error before the truncation point is relatively better than Figure 10(b). However, after the truncation point, it starts to increase and then again flattens out which is predictable according to the boundary condition of this specific function. In other words, this behavior cannot be generalized to other options as well because farther points are outside the training dataset and the network cannot learn their values. So, the preferred way is employing a mapped domain with lower accuracy but more stable behavior. Besides, only 10 equidistant points are used as training points in this case.

Figure 11 shows the superiority of the Adam optimizer as it starts to converge and moves towards the answer in earlier epochs for the first time step. The average calculation time for SGD and RMSprop are, respectively, 0.038 s and 0.045 s per epoch. Figures 12(a) and 12(b) illustrate the influence of fine-tuning on the objective convergence.

Remark for future work: this paper did not cover the application of the method on space fractional or time-space fractional equations [61]. As the sigmoid functions are not fractional, the end result will lack the proper accuracy;

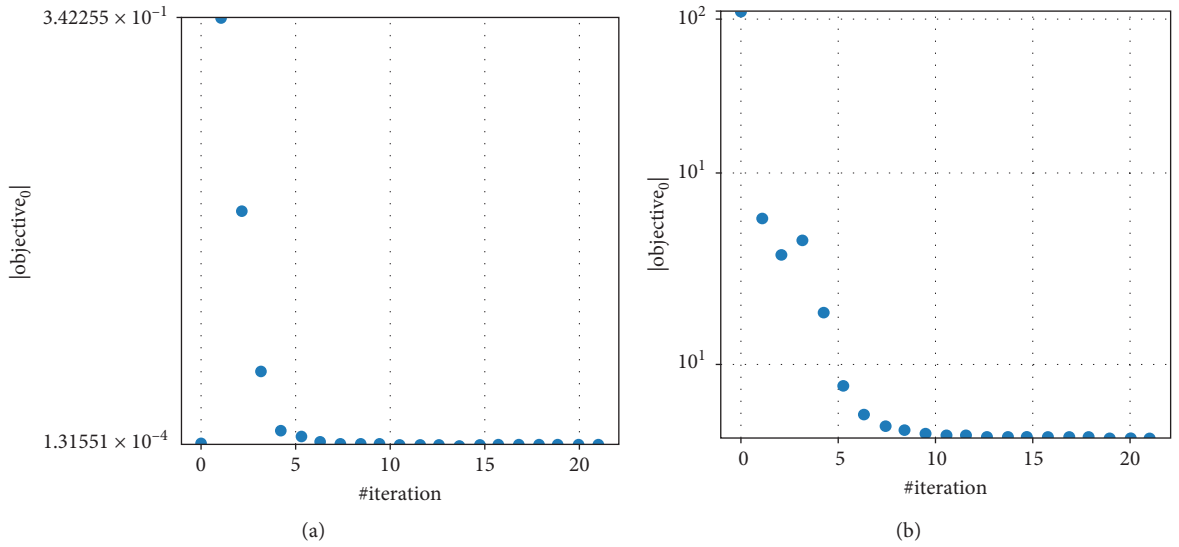

Figure 12: The effect of fine-tuning on the convergence rate of the method. Solving Example 3 with (a) fine-tuning and (b) without fine-tuning.

therefore, readers are encouraged to use other basis functions such as fractional Chebyshev functions to approximate space fractional models.

5. Conclusion

This study investigates neural networks with the famous Adam optimizer for solving financial Black–Scholes equations. Converting the PDE into a series of time-dependent ODEs using the backward Euler finite difference method and then solving each of these equations using the proposed model confirm the satisfactory result and fast calculation of the method. The speed of the method is caused by the parallel computations in the neural network for each independent neuron, the straightforward calculations of sigmoid activation functions that do not add to the complexity of the model, and also the small number of training points and hidden neurons for achieving very promising accuracy. The neural network outperforms other methodologies regarding the consistency and accuracy of the model in confrontation with machines or calculation mistakes because of its fault tolerance. Fine-tuning plays a significant role in this method by reducing the building, validation, and calculation time. It also helps the method converge faster by finding the appropriate direction for gradients as depicted in three examples in Section 4. Domain mapping, which has not been used in ANNs before to the best of found knowledge, is employed to make calculations possible on bigger or infinite problem domains. As a result of combining these approaches into one single ANN, reliable, fast, and accurate results were calculated. The methodology is applicable to other types of options priced by either ordinary or fractional models as well as other partial differential equations in any other field of study that can be solved using this network.

Data Availability

The data used to support the findings of the study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors thank Prof. John P. Boyd (University of Michigan) for his thorough guidance and answering our questions about the methods borrowed from his articles to apply on the problems of the present work.

References

[1] B. Mandelbrot, "The variation of certain speculative prices, The Journal of Business," *The Journal of Business*, vol. 3, pp. 394–413, 1963.

[2] E. Peters, "Fractal structure in the capital market," *Financial Analyst*, vol. 7, pp. 434–453, 1989.

[3] H. Q. Li and C. Q. Ma, "An empirical study of long-term memory of return and volatility in Chinese stock market," *Journal of Finance and Economics*, vol. 31, pp. 29–37, 2005.

[4] T. F. Huang, B. Y. Li, and J. X. Xiong, "Test on the chaotic characteristic of Chinese futures market," *Systems Engineering*, vol. 30, pp. 45–53, 2012.

[5] J. A. Rad, K. Parand, and L. V. Ballestra, "Pricing european and american options by radial basis point interpolation," *Applied Mathematics and Computation*, vol. 251, pp. 363–377, 2015.

[6] R. Farnoosh, A. Sobhani, H. Rezazadeh, and M. H. Beheshti, "Numerical method for discrete double barrier option pricing with time-dependent parameters," *Computers & Mathematics with Applications*, vol. 70, pp. 2006–2013, 2015.

[7] R. Farnoosh, H. Rezazadeh, A. Sobhani, and M. H. Beheshti, "A numerical method for discrete single barrier option pricing

with time-dependent parameters," *Computational Economics*, vol. 48, no. 1, pp. 131–145, 2016.

[8] R. Farnoosh, A. Sobhani, and M. H. Beheshti, "Efficient and fast numerical method for pricing discrete double barrier option by projection method," *Computers & Mathematics with Applications*, vol. 73, no. 7, pp. 1539–1545, 2017.

[9] A. Golbabai and E. Mohebianfar, "A new method for evaluating options based on multiquadric rbf-fd method," *Applied Mathematics and Computation*, vol. 308, pp. 130–141, 2017.

[10] A. Golbabai and E. Mohebianfar, "A new stable local radial basis function approach for option pricing," *Computational Economics*, vol. 49, pp. 271–288, 2017.

[11] J. Rashidinia and S. Jamalzadeh, "Collocation method based on modified cubic b-spline for option pricing models," *Mathematical Communications*, vol. 22, pp. 89–102, 2017.

[12] J. Rashidinia and S. Jamalzadeh, "Modified b-spline collocation approach for pricing american style asian options," *Mediterranean Journal of Mathematics*, vol. 14, p. 111, 2017.

[13] A. Sobhani and M. Milev, "A numerical method for pricing discrete double barrier option by legendre multiwavelet," *Journal of Computational and Applied Mathematics*, vol. 328, pp. 355–364, 2018.

[14] Á. Cartea and D. del-Castillo-Negrete, "Fractional diffusion models of option prices in markets with jumps," *Physica A: Statistical Mechanics and Its Applications*, vol. 374, no. 2, pp. 749–763, 2007.

[15] W. Wyss, "The fractional black–scholes equation," *Fractional Calculus and Applied Analysis*, vol. 3, pp. 51–61, 2000.

[16] T. Björk and H. Hult, "A note on wick products and the fractional black–scholes model," *Financial Stoch*, vol. 9, pp. 197–209, 2005.

[17] A. Meerschaert and M. M. Sikorskii, *Stochastic Models for Fractional Calculus*, Walter de Gruyter, Berlin, Germany, 2012.

[18] A. Golbabai, O. Nikan, and T. Nikazad, "Numerical analysis of time fractional black–scholes european option pricing model arising in financial market," *Computational and Applied Mathematics*, vol. 38, p. 173, 2019.

[19] W. Chen, X. Xu, and S.-P. Zhou, "Analytically pricing double barrier options based on a time-fractional black–scholes equation," *Quarterly of Applied Mathematics*, vol. 72, pp. 597–611, 2015a.

[20] S. Kumar, A. Yildirim, Y. Khan, H. Jafari, K. Sayewand, and L. Wei, "Analytical solution of fractional black–scholes european option pricing equation by using laplace transform," *Computational and Applied Mathematics*, vol. 2, pp. 1–9, 2012.

[21] G. Jumarie, "Derivation and solutions of some fractional black scholes equations in coarse-grained space and time. application to merton's optimal portfolio," *Computational and Applied Mathematics*, vol. 59, pp. 1142–1164, 2010.

[22] H. Q. Li and C. Q. Ma, "The numerical solution of fractional order equation in financial models and its application," Hangzhou University of Electronic Science and Technology, Hangzhou, China, 2009pp. 29–37, Master's thesis.

[23] J. Liang, J. Wang, W. Zhang, W. Qiu, and F. Ren, "Option pricing of a bi-fractional black-merton-scholes model with the hurst exponent h in [1/2 , 1]," *Applied Mathematics Letters*, vol. 23, pp. 859–863, 2010.

[24] G. Hariharan, S. Padma, and P. Prabhaharan, "An efficient wavelet based approximation method to time fractional black–scholes european option pricing problem arising in financial market," *Computational and Applied Mathematics*, vol. 7, pp. 3445–3456, 2013.

[25] W. Chen, "Numerical methods for fractional black–scholes equations and variational inequalities governing option pricing," Ph. D. thesis, University of Western Australia, Perth, Australia, 2014.

[26] A. A. Elbeleze, A. Kılıçman, and B. Taib, "Homotopy perturbation method for fractional black–scholes european option pricing equations using sumudu transform," *Mathematical Problems in Engineering*, vol. 41, pp. 697–704, 2013.

[27] S. Kumar, D. Kumar, and J. Singh, "Numerical computation of fractional blackeschole equation arising in financial market," *Egyptian Journal of Basic and Applied Sciences*, vol. 2, pp. 177–193, 2014.

[28] D. Kumar, J. Singh, and D. Baleanu, "Numerical computation of a fractional model of differential–difference equation," *Journal of Computational and Nonlinear Dynamics*, vol. 11, Article ID 061004, 2016.

[29] J. S. Duan, L. Lu, L. Chen, and Y. An, "Fractional model and solution for the black–scholes equation," *Mathematical Methods in the Applied Sciences*, vol. 41, pp. 697–704, 2018.

[30] O. Marom and E. Momoniat, "A comparison of numerical solutions of fractional diffusion models in finance," *Nonlinear Analysis Real World Applications*, vol. 10, pp. 3435–3442, 2009.

[31] L. Song and W. Wang, "Solution of the fractional black-scholes option pricing model by finite difference method," *Abstract and Applied Analysis*, vol. 2013, Article ID 194286, 2013.

[32] X. Zhang, S. Sun, and W. Lifei, "θ-difference numerical method for solving time-fractional black–scholes equation," *Chinese Academy of Science and Technology Papers*, vol. 7, pp. 1287–1295, 2014.

[33] M. N. Koleva and L. Vulkov, "Numerical solution of time-fractional black–scholes equation," *Computational and Applied Mathematics*, vol. 36, pp. 1699–1715, 2017.

[34] W. Chen, X. Xu, and S.-P. Zhu, "A predictor-corrector approach for pricing American options under the finite moment log-stable model," *Applied Numerical Mathematics*, vol. 97, pp. 15–29, 2015b.

[35] W. A. Khan and F. A. Ansari, "European option pricing of fractional black-scholes model using sumudu transform and its derivatives," *General Letters in Mathematics*, vol. 1, pp. 74–80, 2016.

[36] P. Pradip Roul, "A high accuracy numerical method and its convergence for time-fractional black–scholes equation governing european options," *Applied Numerical Mathematics*, vol. 151, pp. 472–493, 2020.

[37] P. Roul, "A high accuracy numerical method and its convergence for time-fractional black-scholes equation governing european options," *Applied Numerical Mathematics*, vol. 151, pp. 472–493, 2020.

[38] M. Malliaris and M. Salchenberger, "A neural network model for estimating option prices," *Applied Intelligence*, vol. 3, pp. 193–206, 1993.

[39] J. T. Yao, Y. L. Li, and C. L. Tan, "Option price forecasting using neural networks," *Omega-international Journal of Management Science*, vol. 28, pp. 455–466, 2000.

[40] S. Amornwattana, D. Enke, and C. H. Dagli, "A hybrid option pricing model using a neural network for estimating volatility," *International Journal of General Systems*, vol. 36, no. 5, pp. 558–573, 2007.

[41] P. C. Andreou, C. Charalambous, and S. H. Martzoukos, "Pricing and trading european options by combining artificial neural networks and parametric models with implied

parameters," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1415–1433, 2008.
[42] H. Jang and J. Lee, "Generative bayesian neural network model for risk-neutral pricing of american index options," *Quantitative Finance*, vol. 19, pp. 587–603, 2019.
[43] A. M. Ozbayoglu, M. U. Gudelek, and O. B. Sezer, "Deep learning for financial applications: a survey," *Applied Soft Computing*, vol. 93, Article ID 106384, 2020.
[44] J. A. González Cervera, "Solution of the black-scholes equation using artificial neural networks," *Journal of Physics: Conference Series*, vol. 1221, Article ID 012044, 2019.
[45] S. Eskiizmirliler, K. Günel, and R. Polat, "On the solution of the black–scholes equation using feed-forward neural networks," *Computational Economics*, vol. 41, pp. 697–704, 2020.
[46] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.
[47] W. Chen, X. Xu, and S.-P. Zhu, "Analytically pricing double barrier options based on a time-fractional Black-Scholes equation," *Computers & Mathematics with Applications*, vol. 69, no. 12, pp. 1407–1419, 2015.
[48] A. HadianRasanan, N. Bajalan, K. Parand, and J. Rad, "Simulation of nonlinear fractional dynamics arising in the modeling of cognitive decision making using a new fractional neural network," *Mathematical Methods in the Applied Sicences*, vol. 43, pp. 1437–1466, 2019.
[49] Z. Suna and X. Wub, "A fully discrete difference scheme for a diffusion-wave system," *Applied Numerical Mathematics*, vol. 56, pp. 193–209, 2006.
[50] B. C. Csáji, "Approximation with artificial neural networks," M. Sc. thesis, Eötvös Loránd University (ELTE), Budapest, Hungary, 2001.
[51] J. Boyd, *Chebyshev & Fourier Spectral Methods*, Springer-Verlag, Berlin, Germany, 1989.
[52] J. A. Rad, K. Parand, and S. Abbasbandy, "Local weak form meshless techniques based on the radial point interpolation (rpi) method and local boundary integral equation (lbie) method to evaluate european and american options," *Commun Nonlinear Sci Numer Simul*, vol. 22, pp. 1178–1200, 2015.
[53] L. N. Smith, "Cyclical learning rates for training neural networks," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pp. 464–472, Santa Rosa, CA, USA, March 2017.
[54] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search," in *Proceedings of the Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, pp. 3–12, Helsingoer, Denmark, September 1992.
[55] T. Schaul, I. Antonoglou, and D. Silver, "Unit tests for stochastic optimization," in *Proceedings of the International Conference on Learning Representations 2014*, Scottsdale, AZ, USA, May 2013.
[56] T. Tieleman and G. Hinton, "Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, 2012.
[57] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Proceedings of the 3rd International Conference for Learning Representations*, San Diego, CA, USA, May 2015.
[58] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 2018, http://arxiv.org/abs/1904.09237.
[59] S. Bock and M. Wei, "A proof of local convergence for the adam optimizer," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Budapest, Hungary, July 2019.
[60] E. Ekström and J. Tysk, "A fully discrete difference scheme for a diffusion-wave system," *The Annals of Applied Probability*, vol. 19, pp. 1369–1384, 2009.
[61] S. O. Edeki, R. M. Jena, S. Chakraverty, and D. Baleanu, "Coupled transform method for time-space fractional black-scholes option pricing model," *Alexandria Engineering Journal*, vol. 59, no. 5, pp. 3239–3246, 2020, https://www.sciencedirect.com/science/article/pii/S1110016820304087.