Neural Network pricing of American put options

Mahdi Salahshour and Navid Fazly

[1]Sharif University of Technology

Abstract

This report investigates the use of neural networks (NN) in pricing American put options, comparing them to the widely used Least-Square Monte Carlo Method (LSM). The study focuses on four large US companies (Bank of America Corp, General Motors, Coca-Cola Company, and Procter and Gamble Company) and analyzes option prices of them.

The Neural Network models and LSM both demonstrate good accuracy, but the neural networks show superior performance in terms of execution time and Root Mean Square Error (RMSE) once calibrated. While both NN models generally outperform the LSM, the simpler NN model 1 performs comparably. On the other hand, the more complex NN model 2 outperforms almost all other models.

Key Terms: Neural Networks, American Put Options, Least-Square Monte Carlo Method, Pricing, Root Mean Square Error.

Introduction

This study aims to conduct a comprehensive comparative analysis of two distinct methodologies used in pricing American put options. In financial markets, call options provide the holder with the right, although not the obligation, to purchase an underlying asset at a predetermined price. On the other hand, put options confer the right to sell the asset. It is worth noting that European style options can only be exercised on a fixed maturity date, whereas American style options allow for exercise at any point until maturity. Consequently, the inclusion of American style options introduces an optimal stopping time problem that needs to be addressed.

The pricing of American put options is a complex problem that has attracted extensive research attention since its first exploration by Brennan and Schwartz in 1977. Over the years, numerous scholars have made significant contributions to this field, further advancing our understanding of the issue. Researchers such as Parkinson (1977), Geske and Johnson (1984), Kim (1990), Carr et al. (1992), Kuske and Keller (1998), Sullivan (2000), Bunch and Johnson (2000), Rogers (2002), Zhu (2006), Chen et al. (2008), and Cremers and Weinbaum (2010) have all played crucial roles in advancing the understanding of pricing American put options. Their research has provided valuable insights and methodologies for addressing the challenges associated with this problem.

For a comprehensive review of the various approximation methods used in pricing American put options, researchers and practitioners can refer to the survey conducted by Zhao in 2018. This survey systematically reviews and compares the different approximation methods proposed in the literature, thus providing a valuable resource for further research and application in this area.

The present study focuses on comparing two specific approaches: the Least-Square Monte Carlo Method, a simulation-based technique initially introduced by Longstaff and Schwartz (2001), and a machine learning method known as Neural Networks (NN). Neural Networks have gained substantial attention in the field of pricing European style options, as evidenced by notable studies conducted by Hutchinson et al. (1994), Yao et al. (2000), Garcia and Gencay (2000), Bennell and Sutcliffe (2004), Gradojevic et al. (2009), and Liu et al. (2019). However, limited research has been undertaken on the application of Neural Networks to the valuation of American put options. Notably, Jang and Lee (2019) recently examined SP 100 American put options, representing a novel contribution in this area.

By focusing on the pricing of American put options for individual stocks, our study distinguishes itself as one of the pioneering efforts in utilizing Neural Networks. Furthermore, we differentiate our work by employing a larger dataset consisting of generated observations by heston formula.

The organization of this article is as follows: Section 'Methodologies' presents an explanation of the two methodological approaches employed, with a particular focus on the architecture and logic of Neural Networks. Section 'Data' describes the process for selecting and analyzing the data, including descriptive statistics, as well as implementation details for both methodologies. Section 'Result' presents and discusses the results. Finally, Section 'Conclusion' concludes the study and suggests avenues for further research and development.

Methodologies

Least-Square Monte Carlo Method

In the context of option pricing, Longstaff and Schwartz (2001) introduced the least-squares method (LSM) in Monte Carlo simulations. The main idea behind LSM is to estimate the expected payoff of an American option by using least squares regression. Instead of using nested Monte Carlo simulations or complex numerical methods, LSM takes a simpler approach. It involves a two-step procedure:

1. Backward Induction: Starting from the final time step, the method assigns a value to each state of the option at every time step. At each point in the option's cash flow matrix, the exercise value (if immediate exercise is optimal) is known, but the continuation value (if holding the option is optimal) needs to be estimated. LSM uses regression techniques to estimate this continuation value. It considers the in-the-money states of the option and measurable functions of the underlying asset at each time step and state.
2. Option Valuation: Once all the states are valued for every time step, the option's value can be calculated by moving through the time steps and states. At each step, an optimal decision on whether to exercise the option is made based on a particular price path. The value of the resulting payoff is considered.

The advantage of the LSM approach is its simplicity in implementation. It relies on straightforward regression methods and the simulation of the underlying asset's process. However, it's important to note that there may be some approximation error introduced due to the least squares regression used in the estimation process. In this particular study, a 5-degree polynomial is employed to estimate the continuation function as $\mathrm{E}(Y \mid X) = \sum_{i=0}^{5} \alpha_i X^i + \varepsilon$

In the equation above, Y represents the dependent variable and X represents the independent variable. The coefficients of the polynomial terms are represented by α_i.

The residual error is denoted by ε.

Neural Networks

Neural Networks are computational models inspired by the functioning of the human brain. They are widely used in various fields, including finance, for tasks such as option pricing. In this explanation, we will focus on the backpropagation method, which is an algorithm used to train neural networks.

A neural network consists of multiple interconnected artificial neurons organized in layers. The input layer receives the input data, which is then passed through one or more hidden layers. Finally, the output layer provides the network's prediction or decision.

The Mean Squared Error (MSE) is a popular cost function used in various fields, including statistics, machine learning, and optimization. It is a measure of the average squared difference between the predicted values of a model and the actual values. The MSE is commonly used to evaluate the performance of regression models, where the goal is to minimize the difference between the predicted and observed values.

Mathematically, the MSE is computed as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where n is the number of samples, y_i is the actual value of the target variable, and $\hat{y}_i$ is the predicted value of the target variable by our model.

The MSE measures the quality of an estimator by taking into account both the bias (how far off the average estimated value is from the true value) and the variance (how widely spread the estimates are from one sample to another). In other words, the MSE compares how close the predicted values are to the actual values on average.

One advantage of using MSE as a cost function is its mathematical properties. It is always a positive value, and it decreases as the error (difference between predicted and actual values) approaches zero. This makes it a useful tool for assessing the accuracy of a model.

However, the MSE also has limitations. It heavily weights outliers due to the squaring operation, which can make it sensitive to extreme values. Additionally, the MSE is not in the same unit as the original target variable, making it less interpretable.

Despite its limitations, the MSE is widely used in practice due to its simplicity, mathematical properties, and the availability of efficient optimization techniques for minimizing it.

In our application to options pricing, we employ two models: one with a single hidden layer and another with multiple hidden layers.

The backpropagation algorithm is used to adjust the weights and biases of the neural network to minimize the prediction error. It consists of two phases: the forward pass and the backward pass.

1. Forward Pass: During the forward pass, the input data is propagated through the network, one layer at a time, until the output is generated. Each neuron in the network computes a weighted sum of its inputs, applies an activation

function to this sum, and passes the result to the neurons in the next layer.
The activation function introduces nonlinearity into the network, allowing it to model complex relationships between inputs and outputs. Common activation functions include the sigmoid function, hyperbolic tangent function, and rectified linear unit (ReLU) function. We will discuss them later in this part.

2. Backward Pass: In the backward pass, the error between the predicted output and the desired output is calculated. This error is then backpropagated through the network to update the weights and biases. The goal is to minimize the error by adjusting the parameters to improve the network's prediction accuracy.
The backpropagation algorithm uses the chain rule of calculus to compute the gradients of the error with respect to the weights and biases. These gradients indicate the direction and magnitude of the adjustments needed for each parameter. The adjustments are made using an optimization algorithm, such as gradient descent, which updates the parameters in the opposite direction of the gradients.
Mathematical Expressions:
Let's define some variables:
   - Input data: X
   - Weights of the neural network: W
   - Biases of the neural network: b
   - Activation function: f
   - Predicted output: Y_{pred}
   - Desired output: Y_{true}
   - Error: E

   During the forward pass:
   (a) Initialize the weights and biases of the neural network.
   (b) Shuffle the training dataset randomly.
   (c) For each training sample: - Compute the weighted sum:

$$Z = XW + b$$

   - Apply the activation function:

$$A = f(Z)$$

   - Generate the predicted output:

$$Y_{pred} = A$$

   - Compute the error:

$$E = Y_{true} - Y_{pred}$$

   - Calculate the gradients of the error with respect to the weights using the chain rule:

$$\frac{\partial E}{\partial W} = -\frac{\partial E}{\partial A} \cdot \frac{\partial A}{\partial Z} \cdot X$$

   - Update the weights and biases:

$$W_{new} = W_{old} - \eta \cdot \frac{\partial E}{\partial W}$$

$$b_{new} = b_{old} - \eta \cdot \frac{\partial E}{\partial A} \cdot \frac{\partial A}{\partial Z}$$

   - Repeat these steps for each training sample.
   (d) Repeat steps 3 for a specified number of epochs or until convergence is reached.

   It's important to note that in stochastic gradient descent, we update the weights and biases for each training sample individually, which introduces more randomness and can result in faster convergence due to the more frequent updates. However, this randomness can also lead to fluctuations in the training process. To address this, techniques such as mini-batch gradient descent or adaptive learning rates can be used.

Activation functions play a crucial role in artificial neural networks by introducing non-linearity to the network's output. The two commonly used activation functions are Rectified Linear Unit (ReLU) and Sigmoid.

1. The Leaky Rectified Linear Unit (Leaky ReLU) is an activation function commonly used in neural networks. It is a variant of the Rectified Linear Unit (ReLU) function, which is defined as:

$$LeakyReLU(x) = \{\, x\,, \quad if x > 0\ \alpha x, \quad otherwise$$

   While the ReLU function has been widely adopted due to its simplicity and effectiveness in deep learning models, it suffers from a drawback called the "dying ReLU" problem. This occurs when the activation function permanently outputs zero for negative inputs, leading to dead neurons and neuron saturation.
   To address this issue, the Leaky ReLU introduces a small slope for negative values, rather than assigning zero as the output. The mathematical expression for the Leaky ReLU function is as follows:

$$LeakyReLU(x) = \{\, x\,, \quad if x > 0\ \alpha x, \quad otherwise$$

   where α is a small positive value (usually a constant less than 1) known as the leak rate. The purpose of this leak rate is to prevent the complete saturation of neurons and to ensure that even negative inputs contribute a small gradient to the backpropagation process.
   Mathematically, the Leaky ReLU function can be represented as follows:

$$LeakyReLU(x) = \max(\alpha x, x)$$

2. Sigmoid Activation Function: The sigmoid activation function is defined as:

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

The sigmoid function maps the input to a range between 0 and 1, making it suitable for binary classification tasks. It squashes the input values into a probability-like output. The sigmoid function has a smooth gradient, which enables optimization techniques like backpropagation to work effectively. However, sigmoid functions suffer from the vanishing gradient problem, especially for large positive or negative inputs, where the gradient approaches zero, leading to slow learning.

Both ReLU and sigmoid activations have their advantages and disadvantages. ReLU is efficient and prevents saturation, but may suffer from the dying ReLU problem. Sigmoid provides smooth gradients but has a vanishing gradient issue. Hence, the choice of activation function depends on the specific task and network architecture.

In our study, we implemented a neural network for option pricing. To enhance the performance and accuracy of our model, we made a deliberate choice to utilize the leaky ReLU activation function with a specific parameter, a = 0.005. This decision was based on the latest advancements in the field, as supported by recent literature. By opting for the leaky ReLU instead of the conventional sigmoid function or standard ReLU, we aimed to leverage the benefits offered by this particular activation function in our option pricing neural network.

This selection is in line with the current research trends, where the leaky ReLU has been found to exhibit significant improvements in model convergence and performance compared to other activation functions. The leaky ReLU introduces a small negative slope (0.005 in our case) to the otherwise zero gradient for negative inputs, allowing for a more flexible and effective representation of complex data patterns

In our application for option pricing, we adopt Adam Optimizer. The Adam optimizer is widely used in the field of deep learning, especially for large-scale data. It is an adaptive learning rate optimization algorithm that combines the benefits of both the AdaGrad and RMSProp optimizers. Adam stands for Adaptive Moment Estimation.

The Adam optimizer is a stochastic gradient-based optimization algorithm that combines the benefits of Adaptive Gradient Algorithm (AdaGrad) and RMSProp. It dynamically adjusts the learning rate for each parameter based on their past gradients. This adaptive behavior allows Adam to perform better in scenarios with sparse gradients and non-stationary objectives.

Additionally, the Adam optimizer exhibits efficient memory usage by approximating the second-order momentum and incorporating it in a first-order optimization algorithm. This approximation avoids the need to explicitly compute and store the Hessian matrix, which can be computationally demanding for large-scale data.

In conclusion, the Adam optimizer stands out as a preferred choice for large-scale data in neural networks, offering adaptive learning rate, momentum, efficient memory usage, robustness to hyperparameters, and a wide range of applications. Its capabilities make it an effective and efficient optimization technique for parameter tuning in neural network training.

Furthermore, it is customary in machine learning methodologies to normalize the input data within a specific range, typically between 0 and 1. This scaling technique aims to enhance the accuracy of the models by facilitating the optimization process, allowing the loss function to converge towards a global or local minimum more effectively.

The learning algorithm of the multilayer perceptron can be affected by the different scales of variables, leading to convergence issues of the loss function to either local or global minimum. This aspect is currently an active area of research in the field of machine learning. The challenge of finding a local or global minimum was initially introduced by Rumelhart et al. in 1985. They concluded that while the learning algorithm can find a solution in most practical cases, it does not guarantee the availability of a solution.

In terms of global minimum, Choromanska et al. conducted a study in 2015 specifically focusing on the non-convexity of the loss function. This non-convexity often leads the learning algorithms to converge towards a local minimum rather than a global minimum. The study demonstrated that although there is a lack of theoretical support for optimization algorithms in neural networks, the pursuit of a global minimum is not practically relevant as it may result in overfitting of the model.

Data

Description, treatment and statistics

For our data problem, we used the Heston model to generate data, which consists of a total of 27,157 data points. According to the article, it includes derivative securities of 4 stocks: BAC, KO, GM, and PG, which are among the largest companies in the United States.

For each company, in addition to simulating their stock prices, we have also simulated option prices, trading volume, implied volatility, and maturity.

In generating the data, we adjusted the boundaries of the algorithm in such a way that we do not have data with maturity equal to zero, just like in the article.

In addition to implied volatility, which has a significant impact on option prices, money-ness and maturity also have significant effects on price movements.

	Stock_Prices	Implied_Volatility	Strike_Prices	Dividend_Yield	Maturity	Moneyness	Interest_Rates
count	27156.000000	27156.000000	27156.000000	27156.000000	27156.000000	27156.000000	27156.000000
mean	86.593600	0.206603	96.907810	0.037701	120.524672	0.984318	0.159447
std	43.410543	0.328585	58.619252	0.010119	82.286496	0.306236	0.113775
min	27.670265	0.010003	20.000000	0.001180	3.000000	0.457169	0.004683
25%	48.246381	0.046215	47.000000	0.030927	50.000000	0.850538	0.033189
50%	60.253868	0.087433	100.000000	0.037996	108.000000	0.982730	0.209478
75%	128.887018	0.168927	132.500000	0.044784	180.000000	1.046826	0.257838
max	152.790077	5.999996	250.000000	0.072586	350.000000	2.779635	0.392522

Figure 1: *Description of Data*

Figure 1 presents basic statistics of our sample input variables, Figure2 the associated histograms and Figure 3 box plots. After all Figure 4 represents the put prices histogram and a moneyness versus maturity heat graph on put prices

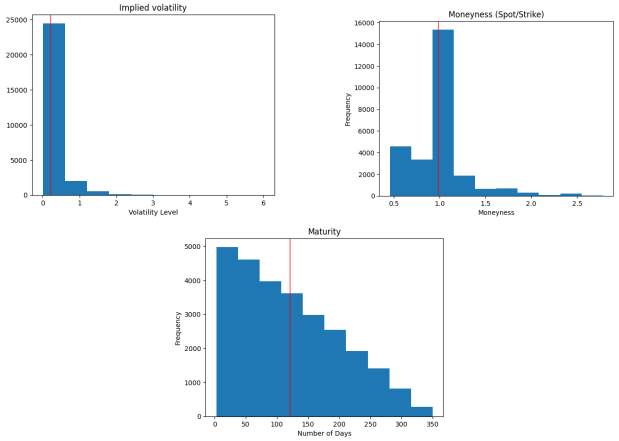
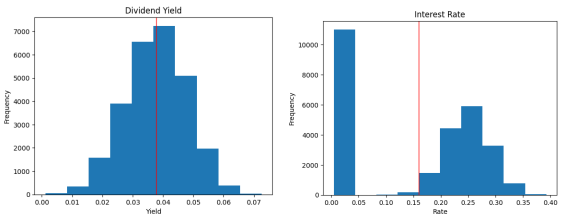


Figure 2: *Input variables histogram*



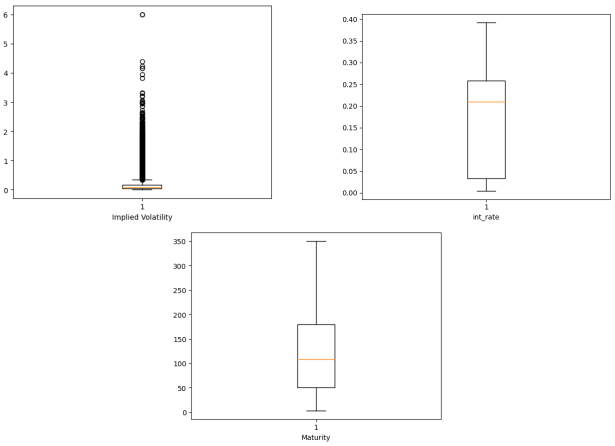
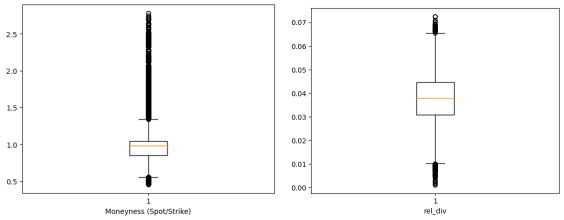


Figure 3: *Boxplots of input variables*



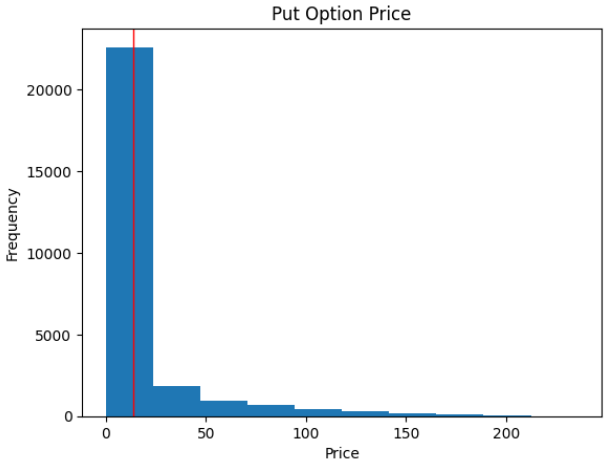
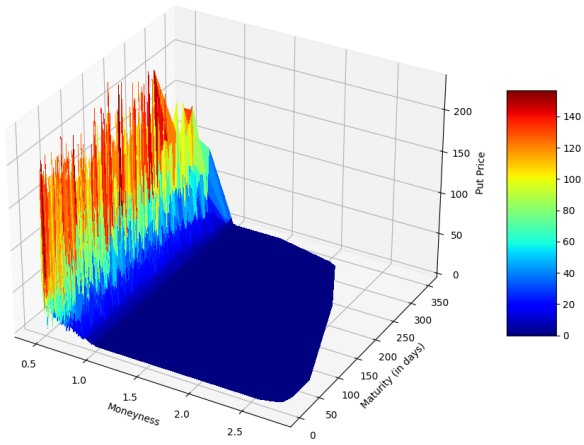


Figure 4: *Put Prices – output variable*



We have considered options with a 5 percent deviation from the current stock price as at-the-money (ATM) options. In put options, an option is considered in-the-money (ITM) if the stock price is below the strike price, which means that the moneyness (stock price divided by strike price) is below 1. On the other hand, options below the 0.95 threshold are ITM options, and options above the 1.05 threshold are out-of-the-money (OTM) options.

Figure 5 presents the percentage of each category for each underlying company. From Figure2, we can observe that the moneyness distribution is heavily centered around 1, with some extreme values in OTM options. In our sample, only 24% of the options are ITM. Additionally, Figure 2 shows that our maturity is heavily skewed and mostly concentrated in maturities below 1 year (less than 365 days). Table 2 also highlights that only about 24% of our sample has maturities larger than 6 months.

Although it is expected to have biases in moneyness and maturity due to the dominance of short-term maturities and ATM/OTM options in the market, these biases may potentially influence the precision of neural network models when learning about ITM and/or longer maturity options.

Sample Moneyness

	ITM	ATM	OTM
BAC	0.3	0.34	0.36
GM	0.32	0.4	0.28
KO	0.29	0.24	0.47
PG	0.18	0.49	0.33
Total	0.24	0.4	0.35

Sample Maturity

	<1 Month	1-6 Months	>6 Months
BAC	0.11	0.62	0.27
GM	0.14	0.62	0.24
KO	0.15	0.62	0.23
PG	0.13	0.63	0.24
Total	0.13	0.62	0.24

Figure 5: *Sample Moneyness and Maturity*

	Unnamed: 0	Put_Prices	Stock_Prices	Implied_Volatility	Strike_Prices	Dividend_Yield	Maturity	Moneyness	Interest_Rates
count	5111.000000	5111.000000	5111.000000	5111.000000	5111.000000	5111.000000	5111.000000	5111.000000	5111.000000
mean	2555.000000	4.038919	45.399145	0.242026	45.140481	0.027992	119.799843	1.111349	0.035032
std	1475.562943	7.912262	4.288045	0.318936	12.877058	0.008068	82.318966	0.426969	0.001994
min	0.000000	0.000000	37.833660	0.010003	20.000000	0.001592	3.000000	0.540481	0.027596
25%	1277.500000	0.000000	41.043706	0.011452	39.000000	0.022671	51.000000	0.943921	0.033716
50%	2555.000000	0.000000	46.822612	0.022909	46.000000	0.027993	105.000000	1.001712	0.035029
75%	3832.500000	2.751260	48.351750	0.370052	50.000000	0.033425	180.000000	1.076852	0.036381
max	5110.000000	51.479703	55.592694	2.307936	70.000000	0.057518	330.000000	2.779635	0.042490

	Unnamed: 0	Put_Prices	Stock_Prices	Implied_Volatility	Strike_Prices	Dividend_Yield	Maturity	Moneyness	Interest_Rates
count	13055.000000	13055.000000	13055.000000	13055.000000	13055.00000	13055.000000	13055.000000	13055.000000	13055.000000
mean	6527.000000	20.403943	130.855324	0.100631	147.77365	0.039995	120.585216	0.939439	0.249706
std	3768.798217	38.967360	7.366832	0.116402	41.34198	0.007992	83.036436	0.205217	0.039942
min	0.000000	0.000000	119.551293	0.053333	100.00000	0.010346	5.000000	0.478205	0.082726
25%	3263.500000	0.000000	126.619014	0.076841	125.00000	0.034673	50.000000	0.866198	0.222800
50%	6527.000000	2.018397	129.137342	0.087433	132.50000	0.039919	105.000000	0.980609	0.250000
75%	9790.500000	15.852181	131.487411	0.099934	150.00000	0.045414	180.000000	1.033625	0.276890
max	13054.000000	235.917863	152.790077	4.167674	250.00000	0.072586	350.000000	1.527901	0.392522

	Unnamed: 0	Put_Prices	Stock_Prices	Implied_Volatility	Strike_Prices	Dividend_Yield	Maturity	Moneyness	Interest_Rates
count	5901.000000	5901.000000	5901.000000	5901.000000	5901.000000	5901.000000	5901.00000	5901.000000	5901.000000
mean	2950.000000	12.814883	53.483206	0.299413	62.810117	0.045045	116.10422	0.966166	0.020043
std	1703.616301	19.098485	3.435941	0.457697	22.151807	0.007952	80.75526	0.362289	0.003994
min	0.000000	0.000000	48.002718	0.010004	30.000000	0.015804	3.00000	0.457169	0.004683
25%	1475.000000	0.000000	51.333513	0.010005	47.500000	0.039628	48.00000	0.686792	0.017387
50%	2950.000000	1.924224	52.650969	0.076742	55.000000	0.045065	102.00000	0.958646	0.020045
75%	4425.000000	22.004544	55.122172	0.414270	75.000000	0.050567	174.00000	1.065014	0.022711
max	5900.000000	116.415324	64.083071	5.999996	105.000000	0.072467	330.00000	2.136102	0.035724

	Unnamed: 0	Put_Prices	Stock_Prices	Implied_Volatility	Strike_Prices	Dividend_Yield	Maturity	Moneyness	Interest_Rates
count	3089.000000	3089.000000	3089.000000	3089.000000	3089.000000	3089.000000	3089.000000	3089.000000	3089.000000
mean	1544.000000	3.444147	30.942093	0.418560	32.725154	0.030038	129.912593	0.998480	0.250150
std	891.861817	6.565921	2.230799	0.461120	7.978104	0.007867	81.211432	0.241137	0.040305
min	0.000000	0.000000	27.670265	0.013802	20.000000	0.001180	5.000000	0.553405	0.115867
25%	772.000000	0.000000	28.882140	0.018203	28.000000	0.024799	60.000000	0.922281	0.223268
50%	1544.000000	0.281621	31.062819	0.394051	31.000000	0.030018	120.000000	0.988645	0.249416
75%	2316.000000	2.696000	32.780220	0.658270	36.000000	0.035338	190.000000	1.068136	0.276883
max	3088.000000	41.208272	36.170843	3.048356	50.000000	0.061898	330.000000	1.808542	0.376795

Figure 6: *Descriptive Statistics per Company*

Training and test datasets

To ensure the effective training and evaluation of our neural network (NN) models, we employ a division of our data into distinct training and test datasets. The training dataset encompasses examples used to train the model by adjusting parameters such as weights and fine-tuning. Meanwhile, the test dataset serves to objectively evaluate the performance of the final trained models and the LSM (Local Sensitivity Matrix) method.

To achieve equitable evaluation, we utilize a random allocation process to divide both datasets. Specifically, the training set constitutes 80% of the entire dataset, leaving the remaining 20% for assessing the performance of our NN models and the LSM method.

While the random selection scheme ensures fairness, it is essential to verify whether both datasets accurately reflect the same underlying reality. To address this, we conduct Kolmogorov-Smirnov tests to compare the distributions of each feature between the test and training sets[3]. Our analysis, as presented in figure7, demonstrates that all features exhibit high p-values, indicating support for the null hypothesis that both sets are drawn from the same distribution. Furthermore, figure8 visually depicts the distribution of each variable for both sets.

By adopting this systematic approach, we can effectively train and evaluate our neural network models while accounting for potential variations between the training and test datasets.

Put_Prices: KstestResult(statistic=0.014326365653331452	**pvalue=0.3602499149752024**	**statistic_location=2.560223297748047** **statistic_sign=-1)**
Stock_Prices: KstestResult(statistic=0.01595432708258171	pvalue=0.2403122066894336 5	statistic_location=123.44821485269348	statistic_sign=-1)
Implied_Volatility: KstestResult(statistic=0.014123331266443884	pvalue=0.3775919960035397 6	statistic_location=0.061646815402149 5	statistic_sign=-1)
Strike_Prices: KstestResult(statistic=0.013873190190012319	pvalue=0.3996636598841 34	statistic_location=115.0	statistic_sign=-1)
Maturity: KstestResult(statistic=0.018664852566548706	pvalue=0.1105044422471 2042	statistic_location=201.0	statistic_sign=1)
Moneyness: KstestResult(statistic=0.012543451074846057	pvalue=0.5288265335120061	statistic_location=0.975115600890756	statistic_sign=1)
Interest Rates: KstestResult(statistic=0.017231514750294052	pvalue=0.1692582032185399	statistic_location=0.22410091893460 77	statistic_sign=-1)

Figure 7: *Kolmogorov-Smirnov by feature*

LSM specification

In our analysis, we consider the dynamics of the stock price S, assuming it follows a Geometric Brownian Motion (GBM). This GBM model, under the risk-neutral measure, is expressed as:

dSt = (r q) St dt + St dWt,

Here, St represents the stock price, r is the risk-free interest rate, q is the associated stock dividend yield, represents the volatility, dt is the time increment, and dWt is a random Wiener process.

It is important to note that although the variables in Equation above —r, q, and —are assumed to be constant, we need to consider the variability present in the options market. Within our test sample, we have different options with varying underlying stocks, trading dates (t), initial stock values, interest rates, dividend yields, and other variables. Additionally, we have chosen to use Bloomberg implied volatilities for , which are specific to each option.

To account for these variations, we have simulated the underlying asset and employed the Longstaff-Schwartz Method (LSM) at each step of the cash-flow option matrix. This approach allows us to estimate the put price of each option in our test set.

During the simulation process, we generated a total of 1,000 paths and divided the timeline into 50 time steps. To determine the continuation values, we employed a 5-degree polynomial Ordinary Least Squares (OLS)

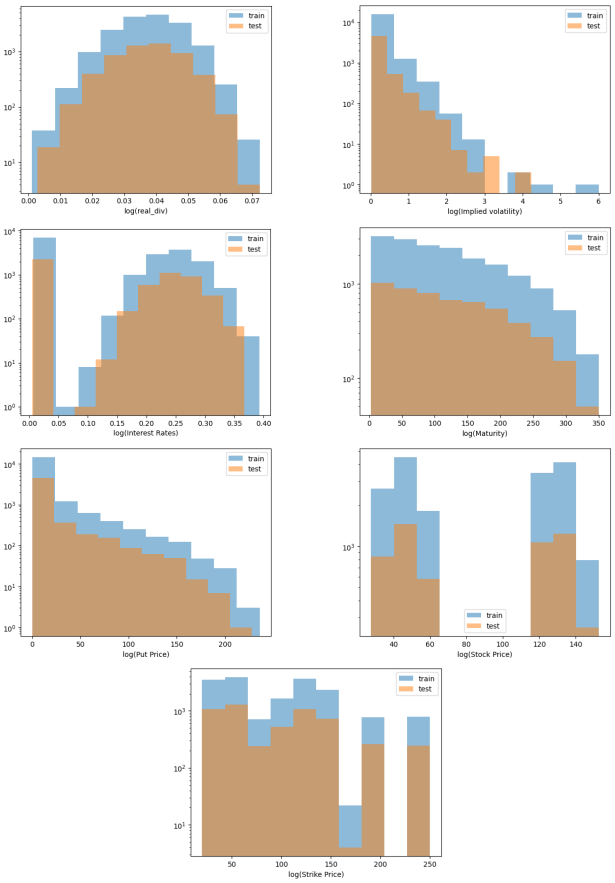


Figure 8: *Kolmogorov-Smirnov by feature*

NN models callibration

To ensure the accuracy and minimize the error of a neural network (NN) model, it is essential to calibrate it within the training dataset. In our approach, we made several decisions and optimized parameters.

To begin the learning process for both of our NN models, we assigned random weights to the hidden nodes. These weights were drawn from a normal distribution with a mean of zero and a standard deviation of 0.5. The initial values of the hidden nodes were randomly chosen and were updated after the completion of the first learning cycle. Additionally, we set a learning rate of = 0.005 with a decay per epoch of 1e-6. The learning rate determines the size of the corrective steps the model takes to adjust for errors in each observation. It presents a trade-off: a higher learning rate reduces training time at the cost of lower accuracy, while a lower learning rate takes longer but provides greater accuracy.

For our first NN model, we aimed to keep the neural network as simple as possible. We only included one hidden layer and a minimal number of inputs. Given the small changes in the dividend yield and interest rates, we chose not to consider them as inputs in this model.

In our second NN model, we expanded the number

of inputs. In addition to the regular option inputs, we also included dummy variables per company and the average put price per company. When it came to hidden layers, we compared NN models with 2, 3, and 4 layers. After assessing their performance, we concluded that the model with 3 hidden layers yielded the best results.

We optimized all other parameters as well. To compare different alternatives, we conducted a cross-validation test using mean square errors (MSE) as the comparison metric. MSE represents the average squared difference between the predicted and true values. For NN model 1, we tested the number of nodes for our single hidden layer, ranging from 3 to 10 nodes. After comparing the values, we noticed that the nodes with 7, 8, and 9 had the lowest variance and mean MSE values. Therefore, we chose to use 9 hidden nodes in the first hidden layer. Similarly, for NN model 2, we optimized the number of nodes across the chosen 3 layers, considering between 3 to 20 nodes for each layer. The best-performing solution, determined through this optimization process, consisted of 16 nodes in layer 1, 8 nodes in layer 2, and 4 nodes in layer 3.

Regarding the activation functions, we tested three different options: sigmoid, ReLU, and Leaky ReLU. After comparing their performance, we found that a leaky ReLU function with = 0.1 yielded better results for both models.

An epoch in the context of NN training refers to a complete cycle of training using the entire dataset. The updating rate, typically referred to as the batch size, was set to 64. For NN model 1, we used 200 epochs, while for NN model 2, we used 3,00 epochs to ensure comprehensive training.

Normalization of variables played a role in our models. In NN model 1, we scaled both the input and output variables, while in NN model 2, we only normalized the input variables. This difference in normalization led to MSE values with different ranges, as depicted in Figure10.

In terms of learning curves for our NN models, Figure10showcases the MSE curve for NN model 1, which rapidly approaches 50. In summary, we took various measures to calibrate and optimize our NN models, including random weight initialization, selection of the number of hidden layers and nodes, choice of activation function, setting the learning rate and decay, determining the batch size and number of epochs, and normalizing variables. Through these steps, we aimed to minimize error and achieve accurate predictions within our neural network models.

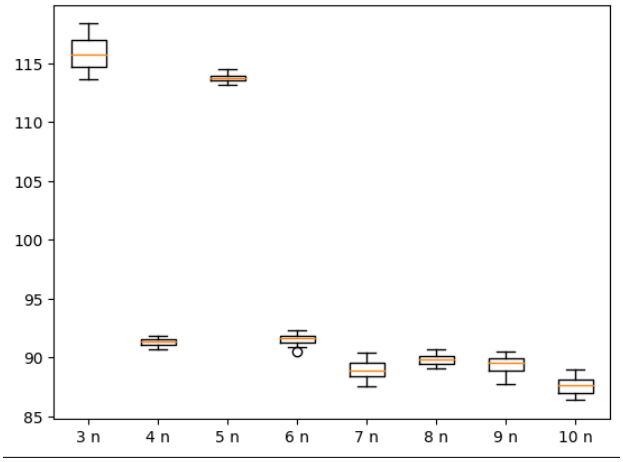


Figure 9: *NN model 1 – number of nodes (one hidden layer)*

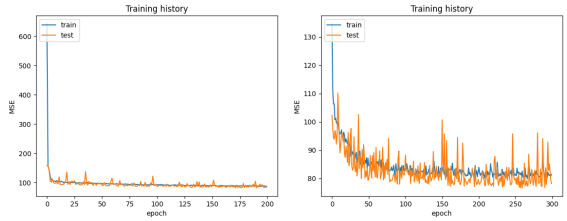


Figure 10: *Learning curve of both NN models: NN model 1 (on the left) and NN model 2 (on the right). As in NN model 1 output are scaled put prices and in NN model 2 unscaled put prices, the MSE values are not comparable*

Results

In this section, we aim to perform a comparative analysis between Neural Network (NN) models and the Least Square Monte Carlo (LSM) method. The objective is to evaluate the performance of these methodologies based on the Root Mean Square Error (RMSE) metric as a means of error comparison. Additionally, we will examine the execution time associated with each method, with a specific focus on pricing options.

Firstly, the calibration time of the neural network (NN) models will be examined. The calibration process for NN model 2 requires approximately 90 minutes due to the utilization of Tensorflow and Keras modules from the Python programming language. These additional modules contribute significantly to an increase in the time needed for calibration. Conversely, NN model 1 can be calibrated within 20 minutes for the full dataset. The shorter calibration duration of NN model 1 in comparison to NN model 2 can be attributed to its simpler architecture and a reduced number of input variables. After the completion of the calibration process, both NN models are capable of providing immediate predictions for each option.

Now, let us proceed with a comparative analysis

of the execution durations of the models under consideration. The Least Squares Monte Carlo (LSM) technique possesses a pricing time of approximately 0.53 seconds for a single option. However, if we incorporate the calibration period into the pricing process for Neural Network (NN) model 2, the average time per option would be reduced to 0.23 seconds. This indicates that the NN model 2 requires less than 0.19 of the time consumed by the LSM method. Furthermore, in the case of NN model 1, the average time per option would decrease to 0.11 seconds, signifying a 0.08 decrease compared to NN model 2.

To compare the results of the models, the evaluation metric employed will be the Root Mean Square Error (RMSE), which will be exclusively considered for the test dataset.

RMSE per model

	NN model 1	NN model 2	LSM
BAC	0.31	0.15	0.2
GM	0.28	0.15	0.47
KO	1.99	1.92	2.9
PG	13.25	13.11	13.03
Total	9.36	9.02	11.34

Figure 11: *RMSE per model*

When examining the RMSE values in Figure 11 . , it becomes evident that NN model 2 outperforms NN model 1 overall. The RMSE metric provides insight into the average dollar deviation ($x) from the actual option value. Notably, among the analyzed companies, Procter Gamble consistently displays the highest deviations, irrespective of the model utilized. This observation can be attributed to the company's underlying assets having the highest spot values. However, when comparing the impact of this factor across the different models, it becomes apparent that NN model 2 exhibits a mitigated influence.

In the context of root mean square error (RMSE) evaluation, NN model 2 displays an average deviation of $9.02 per option, while NN model 1 exhibits an average deviation of $9.36 per option, and the LSM method demonstrates an average deviation of $11.34 per option. This implies that, in reference to our randomly selected test dataset, the NN models surpass the LSM method in terms of performance. It is worth noting that the LSM method is widely acknowledged in academic literature as one of the most precise approaches for pricing American options.

These results, however, should be considered in a broader context. In the following analysis, we will examine the performance of each model in absolute and relative RMSE terms within different moneyness and maturity classes.

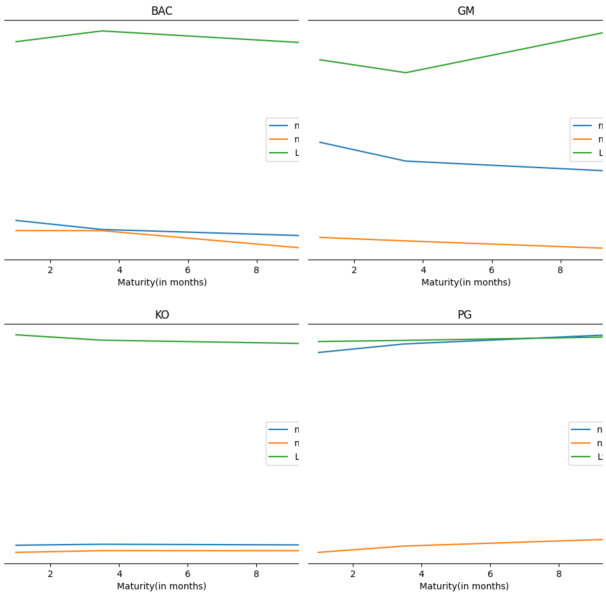


Figure 12: *RMSE per company- maturity*

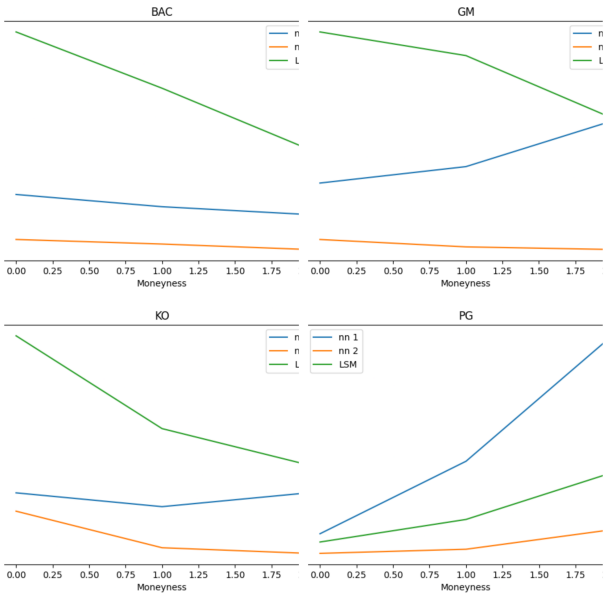


Figure 13: *RMSE per company-moneyness*

RMSE per company (ITM)

	NN model 1	NN model 2	LSM
BAC	0.17	0.03	0.29
GM	0.61	0.07	0.31
KO	0.48	0.3	0.52
PG	3.05	1.13	1.67

RMSE per company (<1 month)

	NN model 1	NN model 2	LSM
BAC	0.48	0.35	1.25
GM	0.5	0.18	0.8
KO	2.07	1.93	5.87
PG	12.72	8.4	13.27

RMSE per company (ATM)

	NN model 1	NN model 2	LSM
BAC	0.18	0.07	0.69
GM	0.18	0.05	0.89
KO	0.27	0.15	0.35
PG	0.87	0.68	0.79

RMSE per company (1-6 months)

	NN model 1	NN model 2	LSM
BAC	0.38	0.54	1.58
GM	0.36	0.19	0.4
KO	2.4	2.31	5.33
PG	12.99	8.61	12.79

RMSE per company (OTM)

	NN model 1	NN model 2	LSM
BAC	0.29	0.09	0.99
GM	0.3	0.12	0.74
KO	0.48	0.47	1.04
PG	0.88	0.87	0.91

RMSE per company (>6 months)

	NN model 1	NN model 2	LSM
BAC	0.36	0.21	1.24
GM	0.37	0.13	0.93
KO	2.08	1.99	5.55
PG	13.57	9.03	13.49

Figure 14: *RMSE per company - moneyness and maturity*

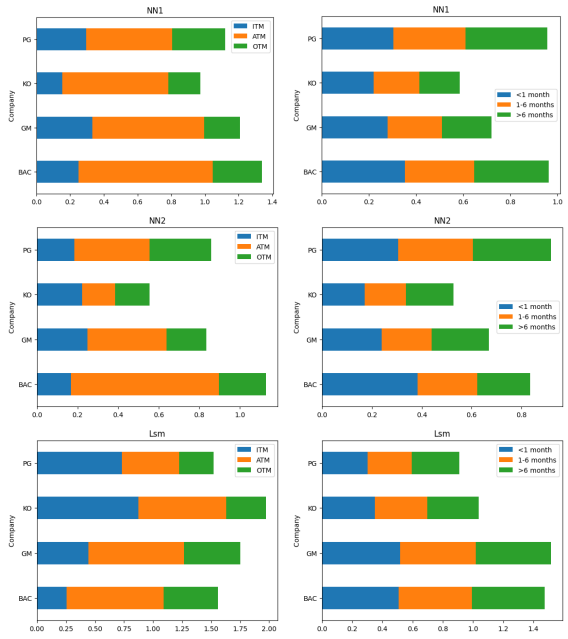


Figure 15: *Relative RMSE per company - moneyness and maturity*

RRMSE per company (ITM)

	NN model 1	NN model 2	LSM
BAC	0.03	0.03	0.17
GM	0.05	0.13	0.11
KO	0.08	0.07	0.11
PG	0.08	0.05	0.07

RRMSE per company (<1month)

	NN model 1	NN model 2	LSM
BAC	0.18	0.13	0.25
GM	0.14	0.08	0.26
KO	0.11	0.07	0.18
PG	0.15	0.1	0.15

RRMSE per company (ATM)

	NN model 1	NN model 2	LSM
BAC	0.27	0.12	0.28
GM	0.22	0.06	0.28
KO	0.21	0.05	0.38
PG	0.17	0.11	0.25

RRMSE per company (1-6 months)

	NN model 1	NN model 2	LSM
BAC	0.2	0.12	0.24
GM	0.15	0.1	0.25
KO	0.13	0.08	0.17
PG	0.2	0.15	0.15

RRMSE per company (OTM)

	NN model 1	NN model 2	LSM
BAC	0.29	0.23	0.47
GM	0.21	0.2	0.48
KO	0.19	0.17	0.34
PG	0.31	0.3	0.3

RRMSE per company (>6 months)

	NN model 1	NN model 2	LSM
BAC	0.21	0.11	0.24
GM	0.14	0.12	0.25
KO	0.12	0.1	0.34
PG	0.23	0.16	0.31

Figure 16: *Relative RMSE per company - moneyness and maturity*

In Figure 14 , the results for different moneyness and maturity are presented for each company. These results are further illustrated in Figure 13 and Figure 12. According to the analysis, there is a noteworthy pattern indicating that options classified as in-the-money (ITM) exhibit greater variability in comparison to at-the-money (ATM) or out-of-the-money (OTM) options. This particular observation is supported by the data presented in Figure 13 . It is noteworthy that this tendency is consistently observed across all four companies under investigation. Significantly, our second neural network (NN) model, depicted by the orange line, consistently demonstrates the most favorable performance in terms of error rates across all instances. When examining our initial neural network model (represented by the blue line) and the least squares Monte Carlo (LSM) method (illustrated by the greeen line), the findings are mixed. Regarding the BAC and KO stocks, our first NN model outperforms LSM regardless of the moneyness class. However, in the case of the PG stock, LSM showcases lower error rates relative to NN model 1, although the discrepancy between the error levels is minimal. As for the PG stock, LSM and NN model 1 yield comparable outcomes, with NN model 1 exhibiting slightly superior performance for in-the-money (OTM) options, and LSM slightly outperforming for at-the-money (ATM). In analyzing the aspect of maturities in options, a notable observation reveals that options with longer maturities tend to exhibit higher pricing. Overall, our NN model 2 consistently performs better than the other two models across all maturities and companies. In the context of evaluating alternative options with varying prices, the utilization of the relative root mean squared error (RMSE) offers significant implications in determining the average deviation, expressed in percentage form, of option prices. Within our comprehensive analysis, we meticulously present the relevant findings in Figure 17 and 15 , efficiently portraying the RMSE as a proportion relative to put prices.

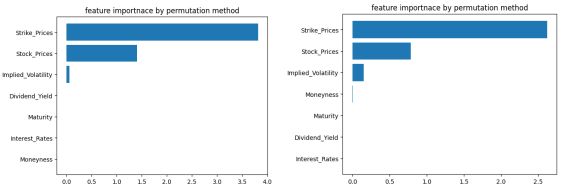


Figure 17: *Left: NN1 , Right: NN2*

The overarching conclusion aligns consistently with the absolute findings: NN model 2 distinguishes itself as the superior model, while our initial NN model 1 closely rivals the least squares Monte Carlo (LSM) approach and even surpasses it in certain instances. It is worth noting that NN model 2 demonstrates the narrowest ranges of relative errors across all moneyness and maturity classifications. Remarkably, it has been observed that out-of-the-money (OTM) options consistently manifest the most significant percentage errors across all models. It is worth emphasizing that these options typically possess relatively lower prices. Nonetheless, when evaluating the underlying assets in question, the relative errors for OTM options span from 17% to 30% for Neural Network (NN) model 2, comparing to 19% to 31% exhibited by NN model 1, and the more extensive range of 30% to 48% displayed by the Least Squares Monte Carlo (LSM) model. In contrast, in-the-money (ITM) options demonstrate smaller ranges of relative errors, ranging from 3% to 17%, depending on the specific model and/or company being analyzed. The "winning" model, NN model 2, mostly exhibits significantly lower errors as a percentage of option prices

compared to alternative models. The relative errors are approximately: 9% for ITM options, 23% for at-the-money (ATM) options, 27% for OTM options, 17% for options with less than one month to maturity, 12% for options with a maturity between one month and six months, and 19% for options with a maturity greater than six months. To ensure a comprehensive evaluation of our neural network (NN) models and obtain insights into the relative significance of each input, we employ a permutation method. This method entails random sorting of each variable's column while maintaining the integrity of all other variables. Subsequently, we rerun predictions on both NN models and compare the newly obtained root mean squared error (RMSE) with the original RMSE. This allows us to gauge the percentual deviation and assess the impact of each input variable. The analysis of the graphs yields additional insights, particularly with regard to NN model 2. The inclusion of dummy variables in this model demonstrates a noteworthy level of relevance, comparable to that of option maturity, and even surpasses the significance of implied volatility.

Conclusion

Neural Networks (NN) have emerged as a widely adopted approach for option pricing due to their numerous advantages, as extensively explored in this research. However, it is important to acknowledge certain limitations and disadvantages that can arise when implementing NN models for option pricing.

One significant challenge is the requirement for a comprehensive and sizable dataset to effectively train the modelling structure. Consequently, NN may face difficulties in accurately pricing more exotic options, particularly over-the-counter options, which are less readily available in the market and possess lower trading volumes. This limited access to data can result in NN models pricing such options less fairly compared to other, more liquid derivatives.

It is also essential to recognize that NN, like other regression methods, are calibrated based on historical data. As a consequence, any significant changes in the future state of financial markets, such as a major financial crisis, can potentially lead to miscalculations in option prices. Therefore, the predictive accuracy of NN models heavily relies on the stability and consistency of market conditions.

In terms of the analytical limitations present in this study, although the number of options utilized is relatively larger compared to previous literature, it remains modest when compared to datasets employed in other fields to calibrate NN models, such as image recognition. This indicates that there is still room for improvement in terms of the volume and variety of options data used for training NN models, potentially enhancing their accuracy and effectiveness. Considering the execution time, the Least-Square Monte Carlo method (LSM) demonstrated reasonable performance in pricing individual options. However, when dealing with more complex underlying processes, such as a jump process with stochastic volatility, the execution time of LSM may be compromised. On the contrary, once calibrated, NN models exhibit immediate execution time and can outperform traditional methods in terms of time efficiency and error performance. This specific study focused on training two NN models using a dataset consisting of American put options with underlying stocks from four different companies, each representing a distinct sector of activity. It is worth investigating whether training NN models specific to a particular sector would yield superior results, as well as how the outcomes might be affected by training a NN model using data spanning more than one year.

Nevertheless, the results of this study offer promising insights into the applicability of NN in option pricing. The simpler NN model, NN model 1, compares favorably to the widely recognized LSM in terms of performance. Moreover, the proposed NN model 2 significantly outperforms both LSM and NN model 1. These results demonstrate the robustness of the findings across various underlying assets, maturities, and levels of moneyness.

If NN models continue to exhibit such promising results, investment companies could potentially adopt them to instantly price options and facilitate swift trading execution. In summary, despite the advantages of NN models in option pricing, it is crucial to consider their limitations, such as the need for comprehensive datasets and the potential for miscalculations during periods of market turbulence. Nonetheless, the study's findings highlight the potential of NN models, particularly the outperformance of NN model 2, and suggest their potential future application in the financial industry. The utilization of NN models could enable investment companies to enhance their pricing capabilities and execute trades more efficiently.