REM WORKING PAPER SERIES

Neural Network pricing of American put options

Raquel Gaspar, Sara D. Lopes, Bernardo Sequeira

REM Working Paper 0122-2020

April 2020

REM – Research in Economics and Mathematics
Rua Miguel Lúpi 20,
1249-078 Lisboa,
Portugal

ISSN 2184-108X

Neural Network pricing of American put options

Raquel M. Gaspar [*][†] Sara D. Lopes [†] Bernardo Sequeira

March 2020

Abstract

In this paper we use neural networks (NN), a machine learning method, to price American put options. We propose two distinct NN models – a simple one and a more complex one. The performance of two NN models is compared to the popular Least-Square Monte Carlo Method (LSM).

This study relies on market American put option prices, with four large US companies as underlying – Bank of America Corp (BAC), General Motors (GM), Coca-Cola Company (KO) and Procter and Gamble Company (PG). Our dataset includes all options traded from December 2018 to March 2019.

All methods show a good accuracy, however, once calibrated, NNs do better in terms of execution time and Root Mean Square Error (RMSE). Although on average both NN models perform better than LSM, the simpler model (NN model 1) performs quite close to LSM. On the other hand our NN model 2 substantially outperforms the other models, having a RMSE ca. 40% lower than that of the LSM. The lower RMSE is consistent across all companies, strike levels and maturities.

KEYWORDS: Machine learning, Neural networks, American put options, Least-square Monte Carlo

JEL CODES: C45, C63, G13, G17

1 Introduction

This study compares two different methods to price American put options. While call options give the right, but not the obligation, to buy some underlying asset at a pre-specified price, put options give the right to sell it. European style options can only be exercised at a pre-defined fixed date, the maturity. American style options, instead, can be exercised at any moment until maturity leading to an optimal stopping time problem.

The case of American put options is particularly hard to solve, and there are no closed form solutions.

This problem was first studied by Brennan and Schwartz (1977) and it has been recurrent in the literature ever since. Importante references on this matter are Parkinson (1977); Geske and Johnson

*Corresponding author. E-mail: Rmgaspar@iseg.ulisboa.pt

†ISEG, Universidade de Lisboa and CEMAPRE/REM Research Center. Rua do Quelhas 6, 1200-078 Lisboa, Portugal.
R.M. Gaspar and S.D. Lopes were partially supported by the Project CEMAPRE/REM - UIDB/05069/2020 financed by FCT/MCTES through national funds.

(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); Cremers and Weinbaum (2010), for instance. For a recent overview of different approximation methods to price American put options see Zhao (2018).

Here we focus on the comparison of two methods: the Least-Square Monte Carlo Method, a simulation method first presented by Longstaff and Schwartz (2001), and a machine learning method, Neural Networks (NN).

Most studies on NN explore mainly European style options. That is the case of Hutchinson et al. (1994), the first article where NN are trained to price options, but also of Yao et al. (2000); Garcia and Gençay (2000); Bennell and Sutcliffe (2004); Gradojevic et al. (2009) and Liu et al. (2019). A very recent exception is Jang and Lee (2019) that looks into S&P 100 American put options.

This study is, thus, one of the first to use NN to address the American put option problem. As opposed to Jang and Lee (2019) it looks into American put options for individual stocks. It is also based on a larger real market dataset, when compared to other NN studies that either use less data, or simulated data.

The rest of the text is organized as follows. Section 2 presents a brief literature overview. Section 3 explains the two methodological approaches used, with particular emphasis on NN architecture and logic. Section 4 explains the data selection process, its descriptive statistics, and specifies both methods implementation details. Section 5 presents and discusses the results. Finally, Section 6 concludes and suggests further developments.

2 Literature Overview

The pricing of American-style options is a classical problem in financial mathematics. This type of problem is associated with a moving boundary that is related with the optimal time to exercise the option. Various approaches have been proposed and developed for the valuation problem of American options. Methods based on solving partial differential equations include finite differences schemes as introduced by Brennan and Schwartz (1977) and the finite element method as presented by Allegretto et al. (2001).

In the class of simulation and tree approaches the most popular techniques include the binomial and trinomial tree methods and the Least Squares Monte Carlo (LSM) method proposed in Longstaff and Schwartz (2001).

Zhao (2018) studied the pricing of put and call options by comparing eight different valuation methods on a one-dimensional scale. He compared numerical methods as the binomial and trinomial tree methods, explicit and implicit finite difference and concluded that the binomial method has the best overall performance in terms of both time and accuracy. In the simulation methods class he analysed the simulated tree from Broadie et al. (1997), the bundling technique from Tilley (1993) and the LSM method, with LSM showing the best results also in terms of both accuracy and execution time.

The fundamental idea of the LSM approach is that the optimal exercise strategy is determined by the conditional expectation of the payoff from continuing to keep the option alive. Moreover, this conditional expectation can be estimated from the cross-sectional information in the simulation by using least squares allowing to accurately value American options by simulation. It should be noted that simulation methods tend to be more attractive when compared to other numerical schemes when the complexity of the option grows or the number of the underlying assets increases.

A most recent approach is the use of neural networks and deep learning methods. Artificial neural networks are a biologically-inspired programming paradigm which enables a computer to learn from observational data and were firstly mathematically formalized in McCulloch and Pitts (1943). The introduction of the error backpropagation learning algorithm by Rumelhart et al. (1985) has since contributed to the increase in popularity of neural networks in many research fields. Neural networks and deep learning currently provide the most efficient solution to many problems in image recognition, speech recognition, and natural language processing.

NNs have also been used to predict or classify economic and financial variables such as the gross domestic product as in Tkacz (2001), the unemployment rate in Moshiri and Brown (2004) , inflation in Choudhary and Haider (2012), exchange rates in Pacelli et al. (2011), or even to predict financial crashes as in Rotundo (2004).

In particular, for pricing financial derivatives, many studies have been pointing to the advantages of using neural networks as the main or complementary tool. For instance, Hutchinson et al. (1994) proposed the use of learning networks for estimating the price of European options and argued that learning networks can recover the Black-Scholes formula from a two-year training set of daily options prices, and that the resulting network formula could be used successfully to both price and delta-hedge options out-of-sample. Garcia and Gençay (2000) estimate a generalized option pricing formula that has a functional shape similar to the usual Black-Scholes formula by a feedforward neural network model and obtain delta-hedging errors that are small relative to the hedging performance of the Black-Scholes model. Yao et al. (2000) use backpropagation neural networks to forecast option prices of Nikkei 225 index futures and shown that for volatile markets a neural network option pricing model can outperform the traditional Black Scholes model. However, they point out that the Black Scholes model is still good for pricing at-the-money options and suggest to partition the data according to moneyness when applying neural networks. In Bennell and Sutcliffe (2004) a comparison between the performance of Black-Scholes with an artificial neural network (ANN) is carried out for pricing European call options on the FTSE 100 index and they concluded that for out-of-the-money options the NN is clearly superior to Black-Scholes and comparable for other cases excluding the deep in-the-money and long maturity options.

Gradojevic et al. (2009) propose a modular neural network (MNN) model to price the S&P-500 European call options. The modules based on time to maturity and moneyness of the options improve the out-of-sample performance when compared with standard feedforward neural network option pricing models. While most studies on NN focus on European options, the empirical study of Jang and Lee (2019) targets S&P 100 American put options prices and also points out the better performance of neural networks when compared with the classical financial option models.

3 Methodology

3.1 Least-Square Monte Carlo Method

Longstaff and Schwartz (2001) propose least-squares method (LSM) in Monte Carlo which uses least squares to estimate the conditional expected payoff from continuation (i.e. the value of not immediately exercising an American option).

Although this reduces computational time drastically when compared with nested Monte Carlo simulation or numerical methods, it also introduces approximation error from the least squares regression.

LSM works in a two step procedure:

- First, a backward induction process is performed in which a value is recursively assigned to every state at every time step. At each point of the option cash-flow matrix the *exercise value* is known, but the continuation value needs to be estimated. LSM uses regressions, considering the in-the-money states of the option cash-flow matrix at each time step and measurable functions of underlying at the same time step and state, to estimate the *continuation value* of the option. In this study we consider a 5 degree polynomial, so the continuation function is estimated as

$$E[Y|X] = \beta_0 + \sum_{n=1}^{5} \beta_n X^n + \epsilon \tag{1}$$

  where Y is the continuation value, X is the value of the stock in the simulation, and ϵ the residual error.

- Secondly, when all states are valued for every time step, the value of the option is calculated by moving through the time steps and states by making an optimal decision on option exercise at every step for a particular price path and the value of the payoff that would result in.

The LSM approach is easily implemented since it relies on simple regression methods and the simulation of the underlying process.

3.2 Neural Networks

Using the definition in Haykin (1994):

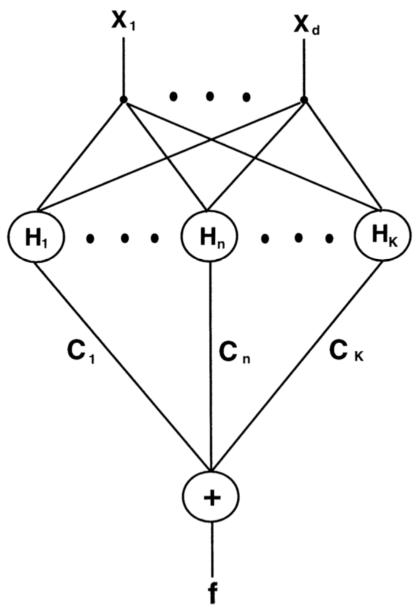
> "A neural network (NN) is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use."

NN are designed to learn a particular task given former experiences. They are used, for instance, in pattern recognition for spam e-mail management. In this case spam e-mail inbox has a learning algorithm that is able to continuously learn from the interaction between users and their spam inbox content.

There are multiple types of NN. The types typically used for option pricing are: backpropagation networks, as in Hutchinson et al. (1994) and Yao et al. (2000), or feedforward networks, as are Tang and Fishwick (1993) and Garcia and Gençay (2000).

This study uses the backpropagation approach, with a structure as represented in Figure 1.

Figure 1: Neural Network Structure – Multilayer Perception



Source: Hutchinson et al. (1994).

The first layer, $(X_1$ to $X_d)$, represents the input layer, where d is the number of input variables. The second layer is the hidden layer, $(H_1$ to $H_k)$, where k is the number of nodes. Finally, f represents the output variable. Each node in the input layer is connected to each node of the set of hidden layers by a weight defined by the model after it is trained. Note that the architecture of the NN allows for more than one hidden layer. A structure with more than one hidden layer adds more connections and weights between the hidden layers only. For exposition simplicity, in this section, we assume a NN with just one hidden layer, although in our application to options pricing, we use both a model with just one hidden layer and another with several.

We can think of NN as an artificial system built to mimic the dynamics of a human brain. Figure 1 can be seen as the interaction between neurons in the human brain, where the input layer are the signals received by the human brain. These signals are then processed by, and with the help of, other neurons that activate different sensors. We can think of these as the hidden nodes (of hidden layers) that will help determine the output variable.

The nodes in the same layer are not connected to each other, but each node is connected to each node from the neighbouring layer. This connection is given by the weighted sum of the values of the previous nodes. Starting with the connection between the input layer and the hidden layer, let X_i represent a input node, and H_k the k-th node from the hidden layer, then each hidden node is obtained as

$$H_k = \phi(\sum_{i=1}^{N_i} w_{k,i} X_i + b_k) \ , \tag{2}$$

where N_i is the number of input variables, $w_{k,i}$ is the weight of the input layer i with respect to the hidden node k, b_k is the *bias node*, and ϕ is a so called *activation function*.

The *bias node* is an extra node added to each layer (except the output layer). It works as an extra argument to the activation function and is always equal to one. While the nodes are connected with the input layer through the weights, the bias node is not connected to the input layer. The argument of the activation function depends on the input nodes and the respective weights and is then used to complete the connection from the input nodes to each hidden node.

It is the *activation function* that scales the argument to a different range, introducing non-linearity and making the model susceptible to non-linear combinations between the input variables.

As in the hidden layer, the output node also depends on the activation function, with the weighted sum of the hidden nodes as the argument. Once we have a value for each H_k, the output of the function is given by

$$f = \phi(\sum_{k=1}^{N_k} v_k H_k + b) \ , \tag{3}$$

where f is the output value, N_k is the number of nodes in the hidden layer, v_k is the weight of the node H_k, and b is the bias.

Several activation functions have been used in the NN literature. The most commonly used is the sigmoid function that can be seen on the left of Figure 2. However, Krizhevsky et al. (2012) empirically compared it to a nonlinear function called Rectified Linear Units (ReLU). The conclusions of the authors are that the ReLUs consistently improve the NN training, decreasing its error. Also Xu et al. (2015) investigated the use of variants of ReLU, such as leaky ReLU. Although the authors recognise the need for rigorous theoretical treatment of their empirical results, they showed leaky ReLU consistently do better results than the original ReLU.

The general equation for both classical ReLUs and leaky ReLU is given by

$$f(x) = \begin{cases} x, & \text{if } x > 0. \\ ax, & \text{otherwise} \ . \end{cases} \tag{4}$$

The classical ReLU and an instance of a leaky ReLU are also represented on the right of Figure 2. For our option pricing NN, we opted for the leaky ReLU (with $a = 0.1$), instead of the traditional sigmoid function or standard ReLU, following the most recent literature.

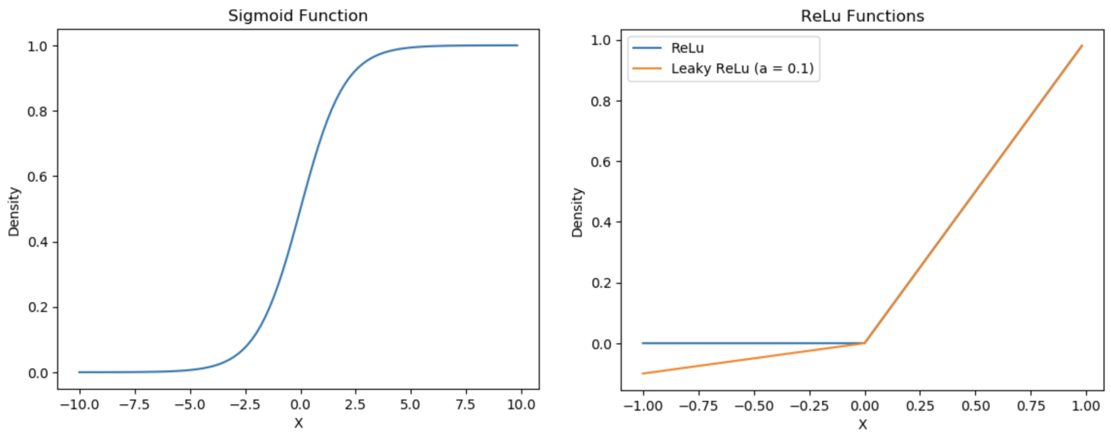
The learning algorithm for NN was first introduced by Rumelhart et al. (1985). In order to get the NN to learn, one should select an algorithm that allows the reduction of estimation error with regards to the correct observations. In the learning process of a multilayer perceptron, the weights of the nodes are adjusted by finding a global, or local, minimum of the so-called *cost function*. The cost function rates how good the NN does in the whole set of observations. Taking mean squared error (MSE) as an example of a cost function, we get

$$C(f_j(\theta)) = \frac{1}{n}\sum_{j=1}^{n} e_j^2, \qquad \text{for} \qquad e_j = f_j(\theta) - f_j(\theta)^* \ \ , \tag{5}$$

where n is the number of observations, f_j is the NN output value and θ, and f_j^* is the real observed value.

Backpropagation adjusts the connection weights to compensate for each error found during the learning process. The error amount is effectively divided among the connections. Technically, back-

Figure 2: Activation Functions



On the l.h.s. the most commonly used sigmoid activation function. On the r.h.s. ReLU and leaky ReLU functions as in (4).The difference between the ReLU functions is on negative part of the function, where, in the case of Leaky ReLU (red line), a can take a small value (0.01), while for ReLU, a is zero.

propagation calculates the gradient of the cost function associated with a given state with respect to the weights. The weight updates is commonly done via stochastic gradient descent (SGD),

$$\theta(t) = \theta(t-1) - \alpha \frac{1}{n} \sum_{j=1}^{n} \nabla f_{t-1}(\theta(t-1)) \ , \tag{6}$$

where α is the value of the *learning rate*. The choice of α should be made carefully as values near 1 would cause the algorithm to oscillate a lot and miss a global/local minimum in one iteration, whereas values near zero can converge to a non-optimal solution, and also slow the convergence to a solution. See LeCun et al. (2012) for an overview of learning rate issues.

The algorithm starts with random or pre-selected weights. Then for each observation, weights are updated at step j as follows,

$$\Delta w_{k,i}^{j+1} = -\alpha \nabla w_{k,i}^{j}, \qquad \text{with} \quad \nabla w_{k,i} = \frac{\partial C(f(\theta))}{\partial w_{k,i}} \qquad , \tag{7}$$

where $\nabla w_{k,i}$ represents the gradient of the cost function with respect to $w_{k,i}$. This value is not known, and we need to know how much a change in $w_{k,i}$ will affect the error. Applying the chain rule we get,

$$\frac{\partial C(f(\theta))}{\partial w_{k,i}} = \frac{\partial C(f(\theta))}{\partial out_n} \times \frac{\partial out_n}{\partial arg_n} \times \frac{\partial arg_n}{\partial w_{k,i}} \qquad , \tag{8}$$

where out_n is the output of the node and arg_n is the input of the node, or also the argument of the activation function of the node. The first partial derivative corresponds to the partial derivative of the error function with respect to the output of observation n, the second corresponds to the partial derivative of the activation function with respect to the argument of the function and the last corresponds to the partial derivative of the argument of the activation function with respect to $w_{k,i}$. We can, thus, use this chain rule to update the weights in (7). Following this logic for every observation, the total error should decrease.

In our option pricing application, we consider a variation of the classical SGD proposed by Bottou (2010) to deal with large-scale problems. In this variation, instead of updating the parameters after iterating once the full training set, i.e. after one *epoch*, we set a size for a sample, called *batch size*, and update the parameters on each random sample within the same *epoch*. This process accelerates the computation time, and is more efficient to use in large data sets, as shown in LeCun et al. (2012) and Bottou (2010). So, when calibrating a NN, one needs to set the number of epochs (iterations) and the batch (sample) size that conditions the number of parameter updates per epoch.

Finally, in machine learning methods it is common to use data scaled to a specific range (0 to 1 for example) in order to increase the accuracy of the models, and allowing the loss function to find a global, or local, minimum. The transformation we use is given by

$$y_i = \frac{\mathrm{x_i} - X_{min}}{\mathrm{X_{max}} - X_{min}} \ , \tag{9}$$

where x_i is the value to normalize, X_{min} and X_{max} are, respectively, the minimum and maximum values of the range. This transformation is also done due to the fact that the *input variables* have different scales. Output variables may be scaled or not, as they have a unique scale.

The learning algorithm from the multilayer perceptron could be affected by the different scales of the variables, and the loss function can fail to converge to the local, or global, minimum. This area is currently an area of research in machine learning. The problem with finding a local or global minimum was first presented by Rumelhart et al. (1985), where the authors conclude that although the learning algorithm finds a solution in almost every practical try, it does not guarantee that a solution can be found. In terms of global minimum, Choromanska et al. (2015) focus their study on the loss function non-convexity that leads the learning algorithms to find a local minimum instead of a global minimum. The authors show that, although there is a lack of theoretical support for optimization algorithms in NN, the global minimum is not relevant in practice because it may lead to overfitting the model.

4 Data

4.1 Description, treatment and statistics

We have used Bloomberg collected data on 37952 American put options, traded from December 2018 to March 2019. The individual stocks under analysis are Bank of America Corp (BAC), Procter and Gamble Company (PG), General Motors (GM) and Coca-Cola Company (KO), selected because of their high market capitalization and their large option trading volume.

For each option, besides daily close prices on the option itself, we collected underlying stock prices, strike prices, maturities, volumes and implied volatilities[1]. We have also retrieved the quarterly dividend paid per share from each company throughout the studied period. In terms of risk-free interest rate, the US treasury rate for different maturities is used. For each option, we select the rate with the closest maturity to that option[2].

We applied a liquidity filter – minimum amount of 20 trades per trading day – to ensure our analysis is based upon reliable data. Also, due to Bloomberg gathering information on trades during the last day of trading of an option, some trades presented zero maturity, so we had to remove those observations. Finally we eliminated all missing values. From the original 37 952 observations we ended up with 21 111, which is still a much larger number than what can be found in the literature. Compare, for instance, with Kelly et al. (1994), Yao et al. (2000) or Kohler et al. (2010) who use 1 369, 17 790 and 2 000, respectively.

Table 1 presents basic statistics of our sample *input variables*, Figure 3 the associated histograms and Figure 4 box plots.

Table 1: Statistics on Input Variables

	Implied Volatility	Moneyness	Maturity	Dividend Yield	Interest Rate
Mean	0.296	1.056	113.2	0.034	0.024
Std Dev	0.105	0.162	171.9	0.008	0.001
Min	0.030	0.565	1	0.020	0.023
25%	0.235	0.976	15	0.025	0.024
50%	0.288	1.025	38	0.038	0.024
75%	0.337	1.096	134	0.040	0.025
Max	2.862	2.268	779	0.048	0.028

Input variables: Bloomberg's *implied volatility*, *moneyness* (ratio between stock price and strike price), *maturity* (time to maturity, here reported in days), *dividend yields* (also known as relative dividends) and the *interest rate* (FED risk-free rates).

Besides implied volatility, that varies mostly across the 4 different underlying companies, the variables that most impact the *output variable* (put prices) are moneyness and maturity. Figures 5 represents the put prices histogram and a moneyness versus maturity heat graph on put prices. In terms of interest rate, there are no big variations in the term structure throughout the 3 months under consideration. In addition, the short end of the term structure was quasi-flat.

[1]For implied volatility, we used the value determined by Bloombergs' quantitative analytics department "Equity Implied Volatility Surface".

[2]The data source is the Board of Governors of the Federal Reserve System (US) and was retrieved from FED, Federal Reserve Bank of St. Louis.

Figure 3: Input variables histogram

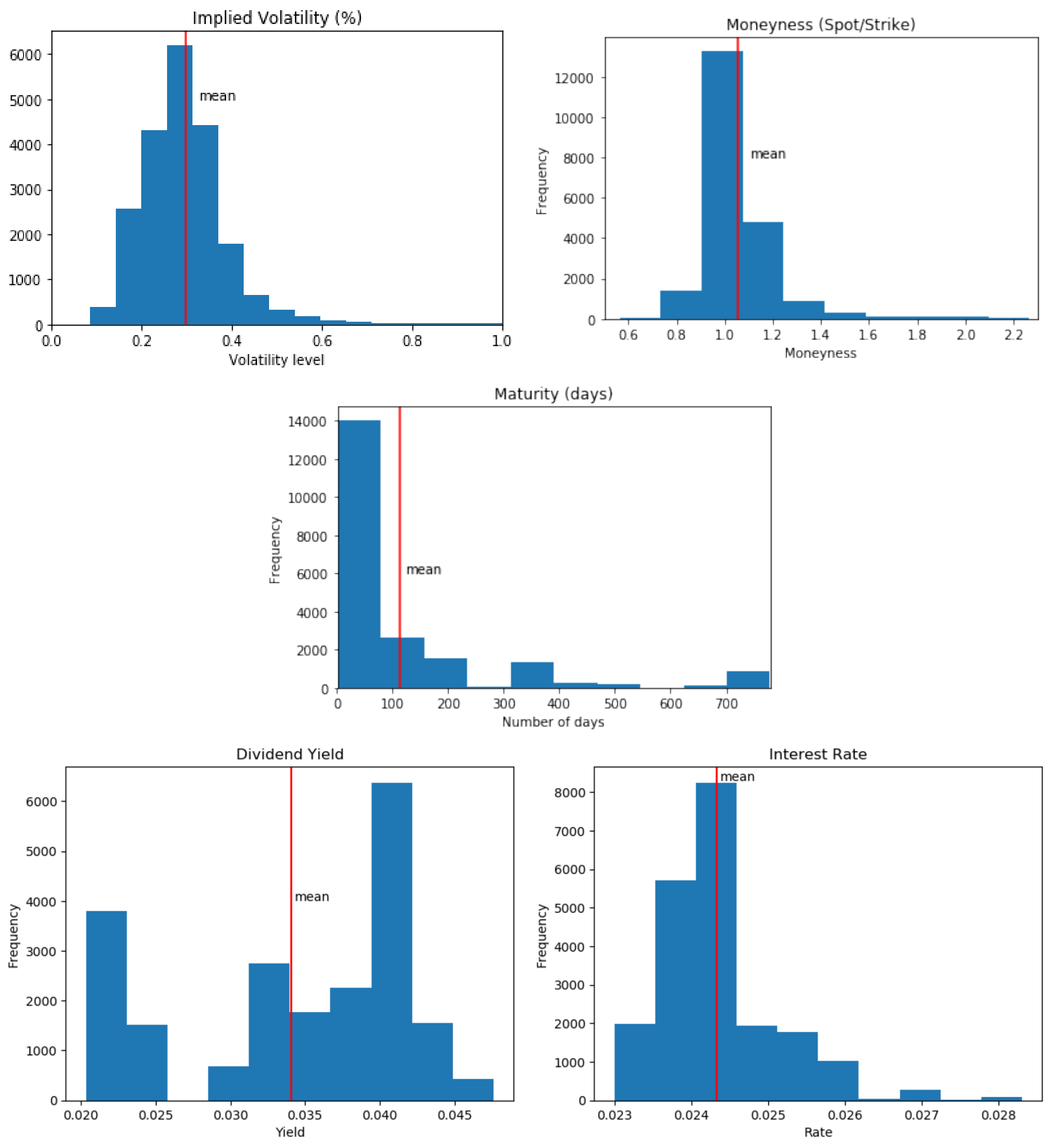


Figure 4: Boxplots of input variables

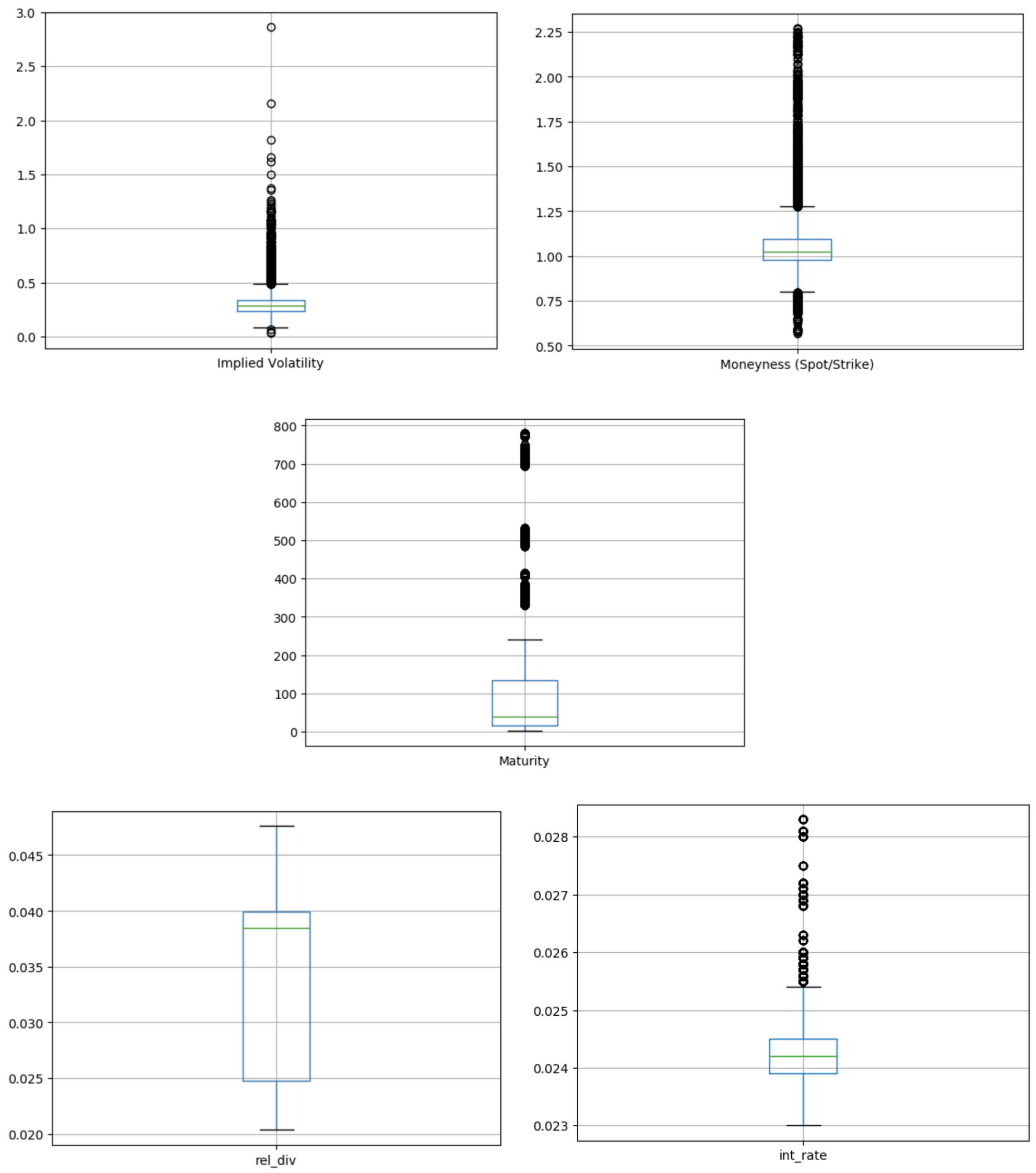
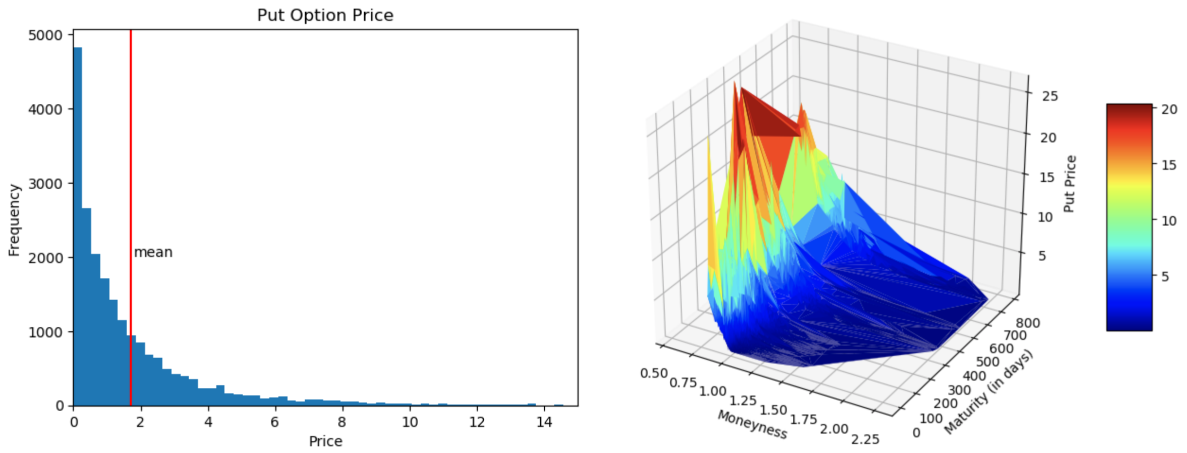


Figure 5: Put Prices – output variable



We have considered as at-the-money (ATM) those options with 5 percent deviation from the current stock price. In put options, an option is in-the-money (ITM) if the stock price is below the strike price, which means that the moneyness, defined before as stock price divided by strike price, is below 1. So, below the 0.95 threshold are ITM options, and above the 1.05 threshold are out-of-the-money (OTM) options. Table 2 shows the percentage of each category for each underlying company. From Figure 3 we can also see the moneyness distribution is heavily centred around 1, with some extreme values in OTM options. In our sample only 16% are ITM options. We can also observe in Figure 3 that, as expected, our maturity is heavily skewed and is almost fully concentrated in maturities below 1 year (less than 365 days). Table 2 emphasizes that only about 18% of our sample has maturities larger than 6 months.

Although both moneyness and maturity "biases" are expectable as short-term maturities and ATM/ /OTM are, by far, the most traded options in the market, it is an issue that may influence the NN models precision, when learning about ITM and/or larger maturity options.

Table 2: Sample Moneyness and Maturity

	ITM	ATM	OTM
BAC	18%	33%	49%
GM	18%	43%	39%
KO	11%	58%	31%
PG	5%	71%	24%
Total	16%	45%	39%

(a) Moneyness

	<1 Month	1-6 Months	>6 Months
BAC	35.18%	43.78%	21.05%
GM	46.73%	35.15%	18.12%
KO	38.40%	43.25%	18.36%
PG	48.14%	41.69%	10.17%
Total	42.72%	39.16%	18.12%

(b) Maturity

Descriptive statistics per company can also be found in Table 3. Each underlying company has different characteristics which positively impacts machine learning process.

Table 3: Descriptive Statistics per Company

	Put price	Imp.Vol.	Spot	Strike	Moneyness	Maturity	Div.Yield	Int.Rate
				BAC				
Mean	1.258	0.306	27.618	26.021	1.090	132.873	0.022	0.024
Median	0.620	0.279	28.306	26.500	1.047	49.000	0.021	0.024
Std.Dev.	1.764	0.119	1.644	4.142	0.207	181.865	0.001	0.001
Min	0.010	0.116	24.246	13.000	0.565	1.000	0.020	0.024
25%	0.170	0.242	25.629	24.000	0.974	21.000	0.021	0.024
50%	0.620	0.279	28.306	26.500	1.047	49.000	0.021	0.024
75%	1.660	0.332	28.932	29.000	1.153	162.000	0.023	0.025
Max	19.400	2.862	29.480	45.000	2.268	743.000	0.025	0.026
				GM				
	Put price	Imp.Vol.	Spot	Strike	Moneyness	Maturity	Div.Yield	Int.Rate
Mean	1.902	0.336	37.31	36.175	1.050	109.743	0.041	0.024
Median	1.080	0.319	38.050	36.500	1.024	36.000	0.040	0.024
Std.Dev.	2.404	0.087	1.648	4.663	0.162	177.520	0.002	0.001
Min	0.010	0.140	31.933	18.000	0.578	1.000	0.038	0.023
25%	0.400	0.287	36.378	34.000	0.968	15.000	0.039	0.024
50%	1.080	0.318	38.050	36.500	1.024	36.000	0.040	0.024
75%	2.480	0.363	38.547	39.000	1.093	105.000	0.042	0.024
Max	26.450	1.168	39.557	65.000	2.198	779.000	0.048	0.028
				KO				
	Put price	Imp.Vol.	Spot	Strike	Moneyness	Maturity	Div.Yield	Int.Rate
Mean	1.364	0.196	47.016	45.827	1.034	115.196	0.034	0.024
Median	0.780	0.182	46.960	46.000	1.019	43.000	0.034	0.024
Std.Dev.	1.890	0.059	1.371	3.995	0.099	165.670	0.001	0.001
Min	0.010	0.030	44.442	25.000	0.684	1.000	0.032	0.024
25%	0.260	0.160	46.236	44.000	0.982	17.000	0.033	0.024
50%	0.780	0.182	46.960	46.000	1.019	43.000	0.034	0.024
75%	1.750	0.213	47.852	48.000	1.067	133.000	0.035	0.025
Max	20.100	0.761	49.359	65.000	1.849	750.000	0.036	0.026
				PG				
	Put price	Imp.Vol.	Spot	Strike	Moneyness	Maturity	Div.Yield	Int.Rate
Mean	2.354	0.221	92.891	90.32	1.033	77.076	0.031	0.024
Median	1.650	0.217	91.397	90.050	1.014	35.000	0.031	0.024
Std.Dev.	2.439	0.056	3.139	6.302	0.077	105.230	0.001	0.001
Min	0.010	0.090	89.929	60.000	0.790	1.000	0.029	0.024
25%	0.710	0.186	90.454	87.500	0.992	14.000	0.030	0.024
50%	1.650	0.217	91.397	90.500	1.014	35.000	0.031	0.024
75%	3.080	0.250	94.840	94.000	1.049	100.000	0.032	0.025
Max	25.000	0.567	99.289	120.000	1.523	744.000	0.032	0.026

Statistics on output and input variables based upon 21 111 American put options, traded from December 2018 to March 2019: 5 302 (25.11%) are options on Bank of America Corp (BAC), 10 609 (50.25%) on General Motors (GM) 3 154 (14.95%) on Coca-Cola Company (KO) and 2 046 (9.69%) on Procter and Gamble Company (PG).

4.2 Training and test datasets

In order to train/develop and then test our NN models, we need to have our data divided into a *training dataset* and a *test dataset*. The training dataset is a set of examples used to train the learning process parameters (weights, etc.) and to fine tune (calibrate) the model. The test dataset is a dataset used to provide an unbiased evaluation of final model(s) fit to the training dataset.

As is common in machine learning techniques, here the division of both data sets is made randomly, with the training set consisting on 80% of the full data set. It is, thus, over the remaining 20% – still more than 4 000 options – that we test the performance of both our NN models and the LSM method.

Despite its random selection, it matters to test if the two datasets are in fact representative of the same reality. Using Kolmogorov-Smirnov tests we have checked the distributions of test and training sets are similar, feature by feature[3]. In Table 4 we observe that all features have a high p-value, which allows us not to reject the null-hypothesis that both sets are drawn from the same distribution. Figure 6 presents the distribution of each variable, for both sets.

Table 4: Kolmogorov-Smirnov by feature

	Put Price	Implied Vol.	Spot	Strike	Maturity	Div. Yield	Int. Rate
p-value	0.191	0.828	0.572	0.401	0.317	0.1121	0.88576
statistic	0.02	0.01	0.01	0.02	0.02	0.02	0.01

4.3 LSM specification

We assume the stock price S follows a GBM whose dynamics, under the risk neutral measure, are given by

$$dS_t = (r - q)\, S_t\, dt + \sigma S_t\, dW_t \ , \tag{10}$$

were W_t is a Wiener process, r is the risk-free interest rate, and q and σ are the associate stock dividend yield and volatility, respectively.

Although all the variables in the above Equation (10) – r, q and σ – are assumed constant, it is important to recall that in our test sample all options are different from one another.

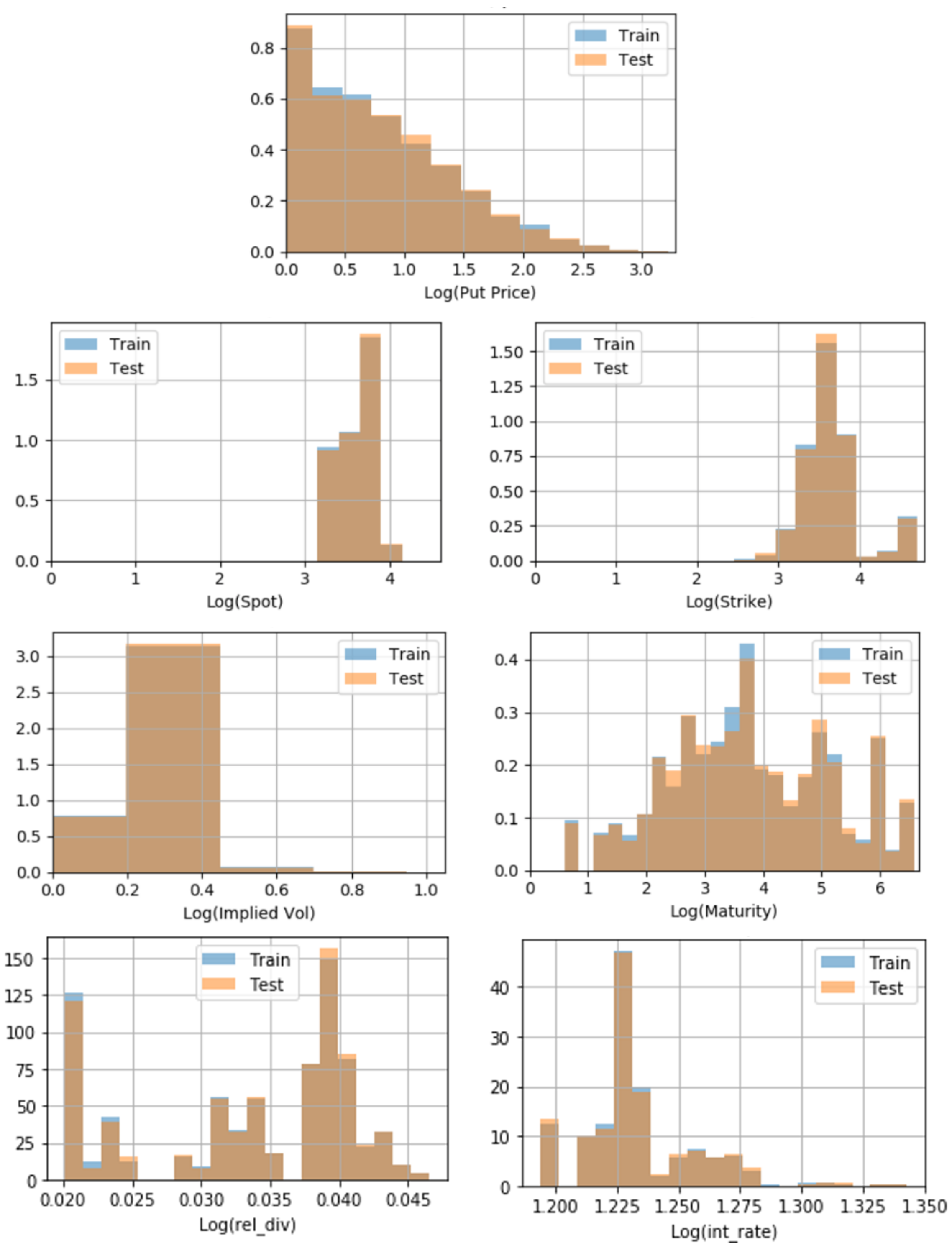
Besides four different underlying stocks, we also have different trading dates (different t), with different initial stock values, interest rate values, dividend yields, etc. In addition, we have also opted to use Bloomberg implied volatilities for σ, and implied volatilities are option specific.

In practical terms this means that for the case of each option of our test set – 4 222 options – we had to simulate both the underlying asset and to use LSM on each step of the cash-flow option matrix (as explained in Section 3.1) to obtain the output – its put price. In the underlying simulation we used as inputs the stock price, interest rate, dividend yield and volatility. For the options part we used, in addition, the strike price and maturity.

For each simulation, we used a total of 1 000 paths and 50 time steps and considered the 5-degree polynomial OLS function in Equation (1) to estimate continuation values.

[3]It tests whether two data sets are drawn from the same distribution. If the Kolmogorov-Smirnov statistic is small or the p-value is high, we cannot reject the hypothesis that the distributions of the two data sets are the same. We only test the important variables, leaving interest rate and dividend yield off, since the values do not variate much.

Figure 6: Kolmogorov-Smirnov by feature



4.4 NN models callibration

In order to minimize the error of a NN model, we need to calibrate it within the training dataset. We used some beforehand made decisions and also optimized parameters.

For both our NN models, in order to start the learning process, we gave random weights, from the normal distribution with mean equal to zero and standard deviation equal to 0.5, to the hidden nodes. The starting values of the hidden nodes are a random choice, and immediately change after the first learning cycle ends. In both cases we have also considered a learning rate $\alpha = 0.005$ with decay per epoch of 1e-6. The learning rate defines the size of the corrective steps that the model takes to adjust for errors in each observation and lead to a trade-off: a high learning rate shortens the training time, at the cost of lower accuracy, while a lower learning rate takes longer, but is more accurate.

For our *NN model 1*, we opted to consider a NN as simple as possible with only 1 hidden layer, and a minimal number of inputs. In particular, given the small changes in the dividend yield and interest rates, we did not even consider them as inputs.

In our *NN model 2*, we considered an increased number of inputs, adding to all the regular option inputs, also dummy variables per company[4] and the average put price per company[5] .

In terms of hidden layers we compared NN with 2, 3 and 4 and have opted for 3 layers as the best performing model.

All other parameters have been optimized, i.e. the alternatives were compared through a cross-validation test, where we looked at the training dataset mean square errors (MSE) as the comparison metric

$$MSE = \frac{1}{n}\sum_{j=1}^{n} e_j^2, \qquad \text{for} \qquad e_j = y_i - y_i^* \ , \tag{11}$$

where e_j is the error of each prediction, with y_i representing the prediction of the model, and y_i^* the true value of the option.

In *NN model 1* and for the number of nodes for our 1 hidden layer we tested for 3 to 10 nodes. When comparing the values, see Figure 7, we note that the 7, 8 and 9 nodes are the ones with the lowest variance in MSE and also the lowest mean values. So, for NN model 1 we, thus, opted for 9 hidden nodes in our 1 hidden layer. Likewise, for *NN model 2* we also optimized the number of nodes for the chosen 3 layers, considering for each layer between 3 to 20 nodes. The optimization was performed simultaneously over the 3 layers. Considering all combinations, the best performing solution was 16 nodes for layer 1, 8 for layer 2, and 4 for layer 3.

After testing the three activation functions previously mentioned – sigmoid, ReLU and Leaky ReLU – a leaky ReLU with $\alpha = 0.1$ performed better in both models. Recall Figure 2.

An epoch is defined as the complete dataset training cycle. In terms of the updating rate, commonly referred to as *batch size*, we assumed a batch size equal to 64. For our *NN model 1* we used 400 epochs while for *NN model 2* we used 3 000.

Finally, while in *NN model 1* we scaled both the input and output variables, in *NN model 2* we did not normalize the output variable. For this reason the MSE values in Figure 8 have different ranges.

Table 5 summarizes all this information.

In terms of learning curves for our NN models, in Figure 8 we can see the MSE curve from NN model 1 going very fast to 0.001. NN model 2 also seems to decrease very fast but if we take a better

[4]A dummy variable takes only 0 or 1 as values. In our case, for example $Company\_KO$ would be 1 if the company is Coca-Cola Company and 0 otherwise. The introduction of such variables is an alternative to training a different model for each company.

[5]This variable represents the mean of (the training set) put price in the training set per company.

Figure 7: NN model 1 – number of nodes (one hidden layer)

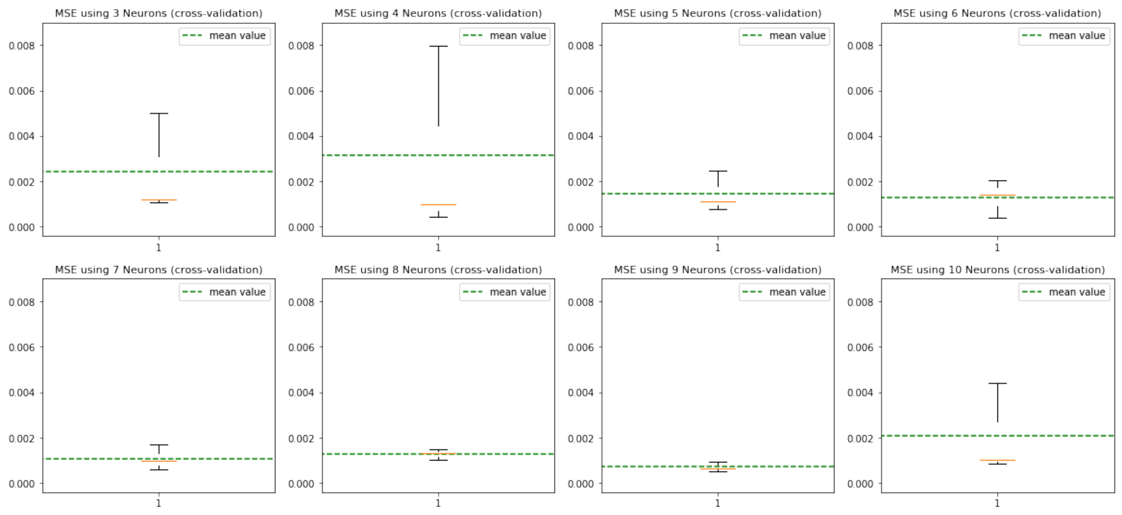
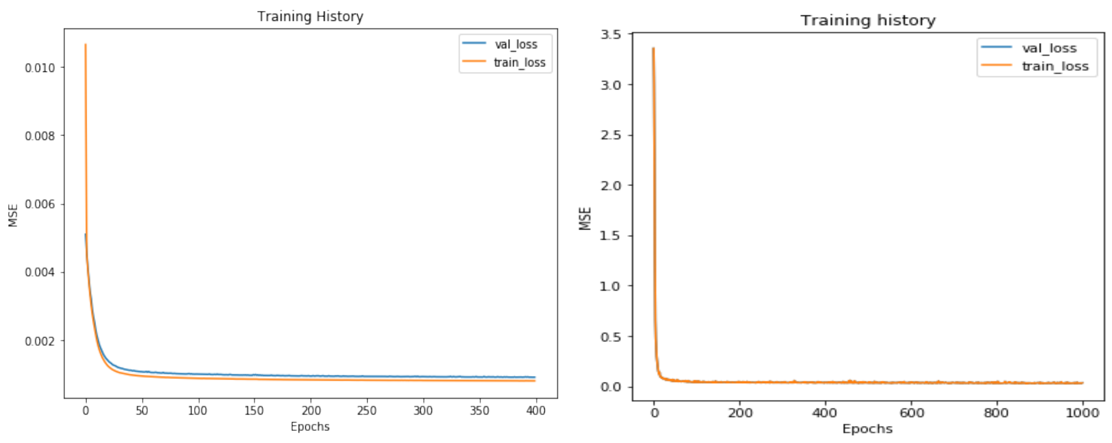


Table 5: NN Models

	NN model 1	NN model 2
Output Variable	Put price (scaled)	Put price (unscaled)
Input Variables (all scaled)	Stock Price Strike Price Implied Volatility Maturity	Stock Price Strike Price Implied Volatility Maturity Dividend yield Interest Rate Dummy per company Average (train set) put value per company
Weights	Random: N(0,0.5)	Random: N(0,0.5)
Learning Rate	$\alpha = 0.005$ w/ decay per epoch of 1e-6	$\alpha = .005$ w/ decay per epoch of 1e-6
Hidden Layers	1	3
Hidden Nodes	9	16, 8, and 4
Activation Function	$a = 0.1$	$a = 0.1$
Epochs	400	3 000
Batch	64	64

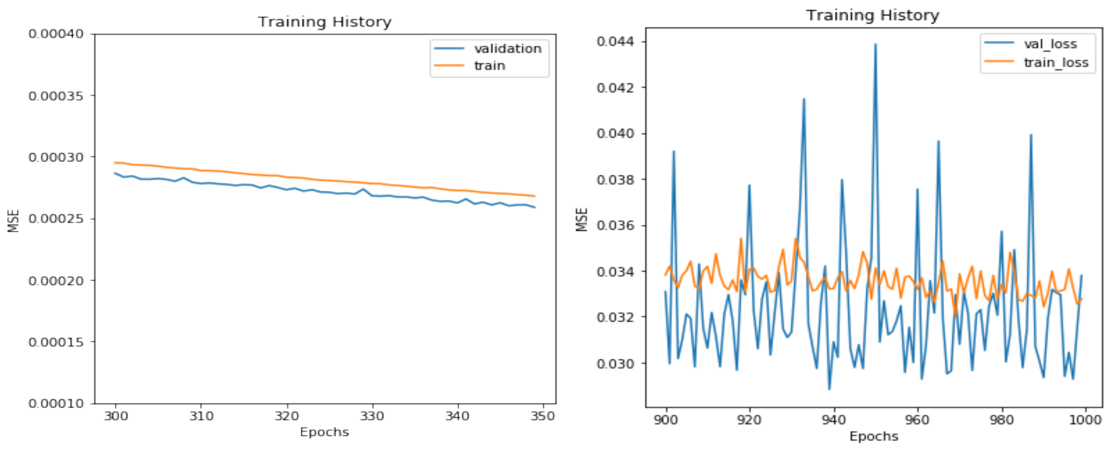
look at Figure 9 we see that there is still space for improvement. Also notable is the high variance when converging to the local optimum.

Figure 8: Learning curves of NN models



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.

Figure 9: NN models learning curves



We can also observe that for NN model 1 the learning curve for the validation set is below the the learning curve for the training set which can indicate that Model 1 is too simple to reproduce the pricing dynamics, as it presents signs of an under-fitted model. On the contrary, we observe a different behaviour for NN model 2 as we have included more layers and variables to reflect the non-linearity and complexity of the relationships between input and output variables.

5 Results

In this section we compare our Neural Network (NN) models results with those of the Least Square Monte Carlo (LSM) method, using the RMSE as the comparison measure for error, and execution time for comparison of the time spent in each method to price the options.

The methods were applied using the programming language Python, version 3.7.3. Experiments were run on a Macbook Pro 14.1 with an Intel Core i5 2.3 GHz processor with a memory of 2133 MHz and 8 GB of RAM, running macOS version 10.14.6.

First of all, the calibration time of NN model 2 is about 5 minutes due to Tensorflow[6] and Keras[7] modules from python that substantially increase the calibration time. NN model 1 takes 2 minutes to train the full set. Its higher speed, when compared to NN model 2, is explained by the simpler architecture, as well as the fewer number of input variables. After the calibration time, the prediction of both NN models per option is immediate. LSM takes about 0.38 seconds to price one option alone. In terms of time. If, in NN models, we include the calibration time in the pricing, for NN model 2, for each option we would have the average of 0.07 seconds, less than 20% of time when compared to LSM. In the case of NN model 1, the average time per option would be 0.03 seconds, 50% less than in NN Model 2.

To compare model results we use root mean square error (RMSE), given by

$$RMSE = MSE^{0.5} \ , \tag{12}$$

with the MSE as previously defined in Equation 11, but now considered only over the test dataset.

Table 6: Root Mean Square Error (RMSE) per model

	NN model 1	NN model 2	LSM
BAC	0.166	0.099	0.112
GM	0.215	0.163	0.268
KO	0.198	0.139	0.302
PG	0.457	0.273	0.400
Total	0.223	0.161	0.259

In terms of RMSE we can observe in Table 6 that overall NN model 2 is the best performing model.

The RSME can be interpreted as the average dollar deviation ($\$x$) from the true option value. Procter & Gamble is the company with the highest deviations, no matter the model. This is probably due to the fact that its underlying is the one with the highest spot values (recall Table 3). Comparing across models, NN model 2 is the one where this effect is less obvious.

In terms of total RMSE, NN model 2 deviates on average $\$0.16$ per option, while NN model 1 deviates $\$0.22$ per option, and the LSM has an average deviation of $\$0.26$ per option. This means that overall, for our randomly selected test data set, on average NN models did outperform the LSM, considered in the literature as one of the most accurate methods to price American options.

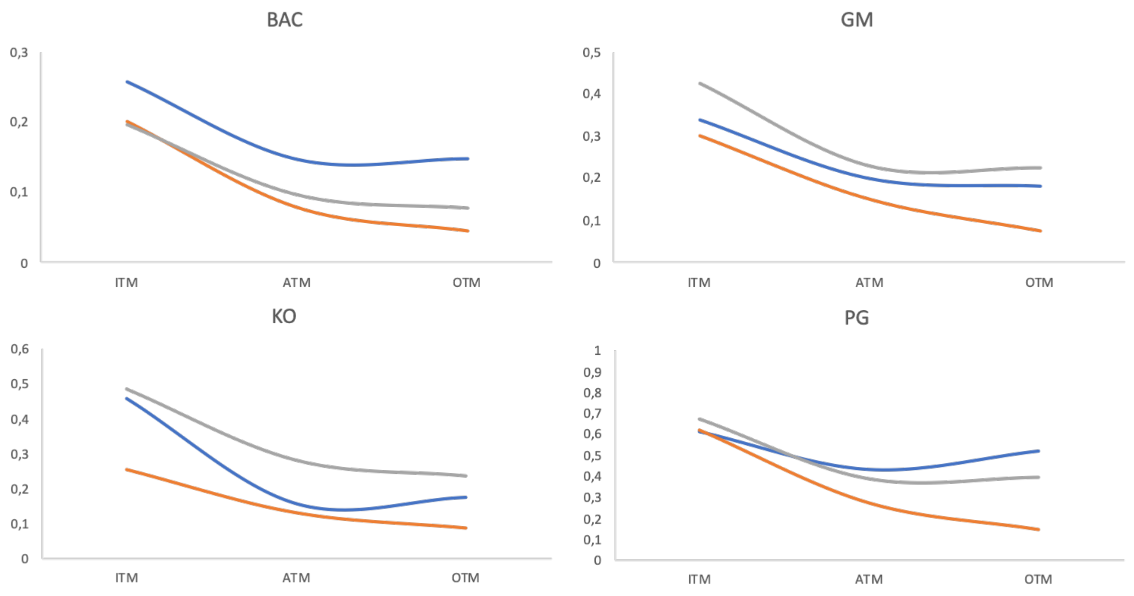
These are, of course, just general results. Below we analyse the performance of each model – both in absolute and relative RMSE terms – per moneyness and maturity classes.

[6]Tensorflow is an open-source library used in Machine Learning models, check Abadi et al. (2016) for a deep understanding on why Tensorflow greatly increases time spent in training any Neural Networks model.

[7]Keras is a NN library that is capable of running on top of TensorFlow and other libraries. It provides an API for building deep learning models quickly and efficiently.

Figure 10: RMSE per Company – moneyness and maturity

(a) Moneyness



(b) Maturity

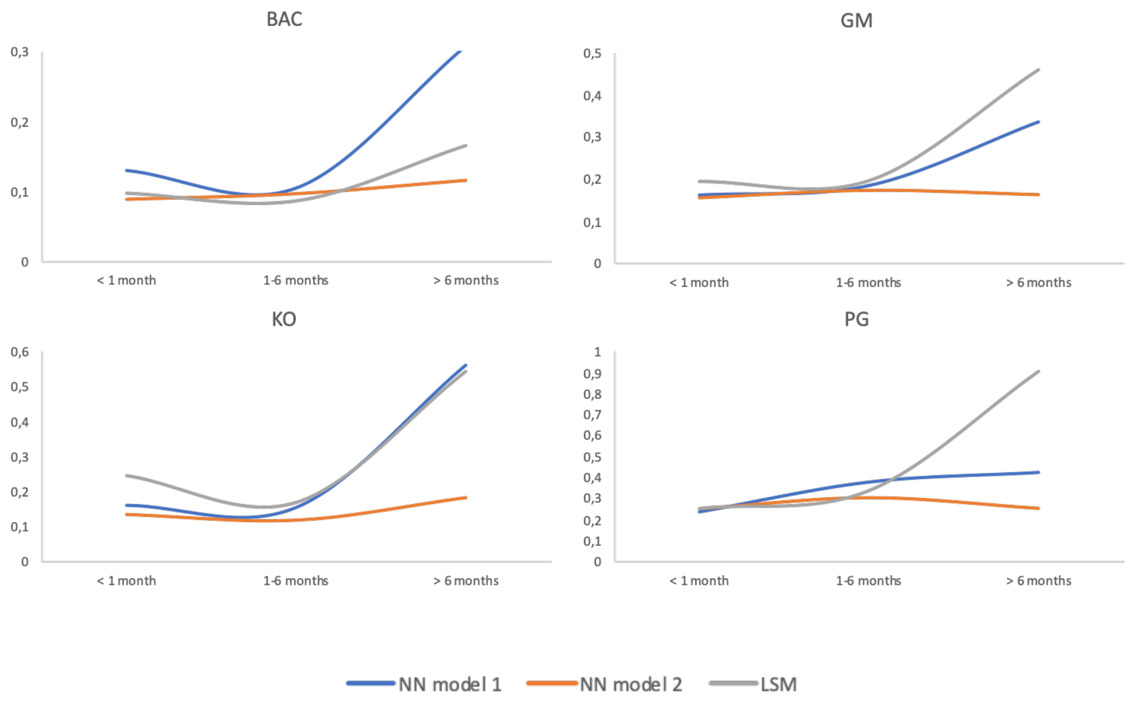


Table 7: RMSE per Company – moneyness and maturity

	ITM		
	NN model 1	NN model 2	LSM
BAC	0.257	0.200	0.196
GM	0.339	0.300	0.424
KO	0.458	0.254	0.487
PG	0.613	0.622	0.671
	ATM		
	NN model 1	NN model 2	LSM
BAC	0.147	0.079	0.096
GM	0.199	0.149	0.228
KO	0.157	0.130	0.283
PG	0.430	0.272	0.385
	OTM		
	NN model 1	NN model 2	LSM
BAC	0.148	0.046	0.077
GM	0.181	0.074	0.224
KO	0.175	0.086	0.238
PG	0.519	0.144	0.392

(a) Moneyness

	$<$ 1 Month		
	NN model 1	NN model 2	LSM
BAC	0.131	0.089	0.098
GM	0.162	0.155	0.195
KO	0.160	0.134	0.244
PG	0.238	0.248	0.253
	1-6 Months		
	NN model 1	NN model 2	LSM
BAC	0.106	0.097	0.087
GM	0.185	0.174	0.197
KO	0.154	0.118	0.167
PG	0.380	0.304	0.338
	$>$ 6 Months		
	NN model 1	NN model 2	LSM
BAC	0.308	0.116	0.166
GM	0.338	0.163	0.469
KO	0.561	0.182	0.543
PG	0.426	0.250	0.904

(b) Maturity

Table 7 presents results per company and for different moneyness and maturity. Figure 10 illustrate the same results.

Since these are absolute, i.e.$, results, naturally, most expensive ITM options tend to present higher deviations than ATM or OTM options. See Figure 10a. This bias – or "smirk" – is present on the graphs of the four companies.

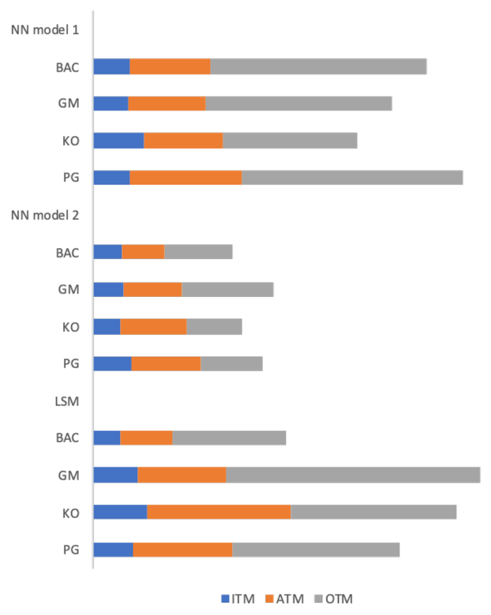
More importantly, at all instances, our NN model 2 (orange line) has the lowest errors. When comparing our naive NN model 1 (blue line) and LSM (grey line) the evidence is mixed. For the underlyings GM and KO, our NN model 1does better than LSM, no matter the moneyness class. When BAC stock is the underlying, LSM shows less errors than NN model 1, with error levels above, but close to NN model 2. Finally, for the PG stock, LSM and NN model 1 show very close results to one another, with NN model 1 performing slightly better for ITM options and LSM slightly better for ATM and OTM options.

In terms of maturities, the longer the maturity the more expensive is the option. This expected bias in model performance is evident for NN model 1 and LSM, but interestingly enough not for NN model 2, that tends to present a quasi-flat absolute errors term structure. See Figure 10b.

As before, our NN model 2 performs better than the other two models for all maturities and companies, performing fantastically well in the case of options with maturities larger than 6 months.

When dealing with a variety of options with various prices, *relative* RMSE gives us the average error in percentage of option prices. Figure 11 and Table 8 presents the relative results, presenting RMSE as

Figure 11: Relative RMSE per Company – moneyness and maturity



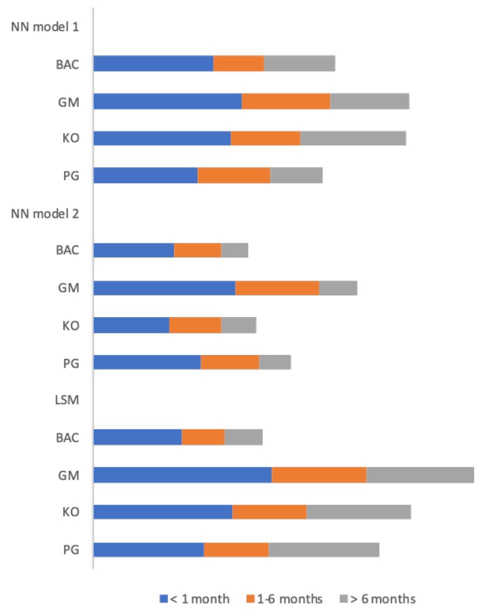


Table 8: Relative RMSE per Company – moneyness and maturity

	ITM		
	NN model 1	NN model 2	LSM
BAC	6.56%	5.10%	5.00%
GM	6.25%	5.53%	7.82%
KO	8.89%	4.93%	9.45%
PG	6.58%	6.68%	7.20%
	ATM		
	NN model 1	NN model 2	LSM
BAC	13.87%	7.45%	9.05%
GM	13.60%	10.18%	15.58%
KO	14.01%	11.60%	25.25%
PG	19.36%	12.25%	17.33%
	OTM		
	NN model 1	NN model 2	LSM
BAC	37.78%	11.74%	19.66%
GM	23.32%	9.53%	28.86%
KO	32.46%	15.95%	44.14%
PG	38.60%	10.71%	29.16%

(a) Moneyness

	$<$ 1 Month		
	NN model 1	NN model 2	LSM
BAC	21.00%	14.27%	15.71%
GM	25.97%	24.85%	31.26%
KO	24.14%	13.40%	24.40%
PG	18.23%	19.00%	19.38%
	1-6 Months		
	NN model 1	NN model 2	LSM
BAC	8.96%	8.20%	7.35%
GM	15.64%	14.71%	16.65%
KO	11.99%	9.19%	13.01%
PG	12.69%	10.16%	11.29%
	$>$ 6 Months		
	NN model 1	NN model 2	LSM
BAC	12.44%	4.69%	6.71%
GM	13.66%	6.59%	18.59%
KO	18.59%	6.03%	17.99%
PG	9.08%	5.33%	19.27%

(b) Maturity

percentage of put prices. The overall message is the same as with the absolute results – NN model 2 is the best model and our naive NN model 1 performs close to LSM, beating it at some instances.

The ranges of relative errors of NN model 2 are the smallest in all categories of moneyness and maturity.

OTM have the highest percentual errors for all models. We must, however, remember these tem to be relatively cheap options. Still, for the underlyings under consideration NN model 2 errors for OTM range from 9.53% to 15.95%, while for NN model 1 they range from 23.32% to 38.60% and for LSM from 19.66% to 44.14%.

ITM options are the ones presenting smaller ranges of relative errors, in between 4.93% and 9.45%, depending on the model and/or company under analysis.

The "winning" model – NN model 2 – presents errors as percentage of options prices considerably lower than the alternative models, around: 5% for ITM, 10% for ATM, 12% for OTM, 20% for less than on month, 12% for between 1 month and 6 months, and 6% higher than 6 months maturity.

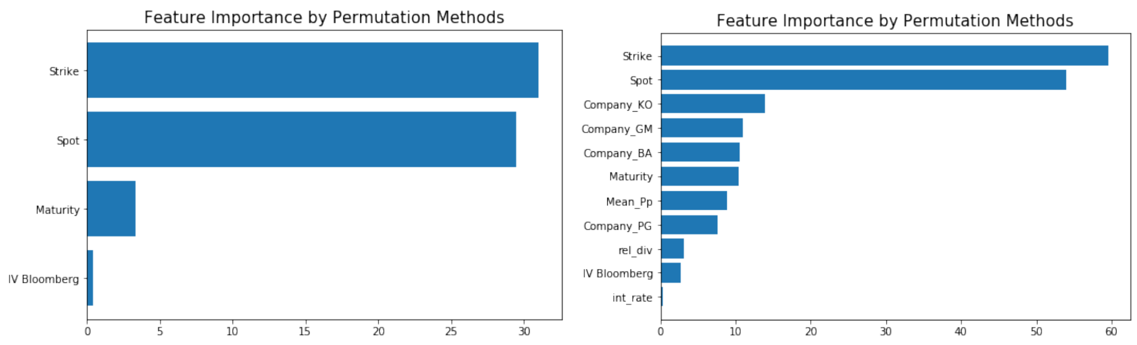
To conclude the analysis of our NN models, and get a better feeling on the relative importance of each input, we use a permutation method.

For each variable we sort the column of the variable randomly, *ceteris paribus*, and re-run the new predictions on both NN models, comparing the new RMSE with the original RMSE from the correct, in terms of percentual deviation.

Figure 12 shows the ordered feature importance. For instance, when in NN model 2 we randomize interest rate values, the total RMSE increases by 27%, which is not as substantial as the absence of a correct strike price, which increases the RMSE by approximately 6 000%. It is not surprising that getting as input the correct spot and/or strike prices is key in obtaining the correct option prices.

The graphs are more interesting when read in comparative terms. For instance, it is clear that for NN model 2 the dummy variables were important, with relevance similar to the option maturity and more relevant than implied volatility.

Figure 12: Inputs permutation results



Permutation method applied to NN model 1 (on the left) and NN model 2 (on the right). The x-axis shows the increase, in percentage, of RMSE that results from considering one of the inputs randomised instead of the calibrated values.

Table 9 shows a measure of relative importance of each input for both NN models. This relative importance has been computed using the permutation technique results, dividing each input RMSE percentual increase by the sum of all inputs RMSE percentual increase, for each model.

Table 9: Inputs relative importance

Input Variables	NN model 1	NN model 2
Stock Price	45.92%	29.66%
Strike Price	48.31%	32.75%
Implied Volatility	0.64%	1.51%
Maturity	5.14%	5.72%
Dividend yield		1.73%
Interest Rate		0.15%
Company_BAC		5.79%
Company_GM		6.03%
Company_KO		7.65%
Company_PG		4.16%
Average (train set) put value		4.84%

6 Conclusion

Neural Networks (NN) have many advantages when it comes to option pricing as explored throughout this work. Still, some disadvantages can be encountered, as the necessity to have a complete and large data set to train the modelling structure. This implies, for instance, that more exotic options, and, in particular, over-the-counter options, cannot be priced as fairly by the NN when compared with other derivatives more readily available in the markets and with a higher trading volume. Also, a note of advice is that, when using NN, as with other regression methods, the model is calibrated to past data, which means that any changes to the future composure of financial markets, for example, a big financial crisis, might modify the values of worldwide options, leading to miscalculations in prices.

In terms of limitations of the analysis presented in this study, we note that although the number of options used here is larger than most studies in the literature, is still scarce, especially when compared to the datasets used to calibrate NN models in other fields. That is the case, for instance in image recognition where NN are the main methodology applied.

Regarding execution time, while Least-Square Monte Carlo method (LSM) produced results in a reasonable amount of time to price a single option, if a more complicated underlying process, for example a jump process with stochastic volatility, was added into the LSM, the execution time might be compromised. On the contrary, once calibrated, the NN have an immediate execution time, and this method can beat traditional methods in terms of time and error performance.

This study focused on training two NN models to a data set from American put options with four companies' stocks as underlying. Each of the stocks from a different sector of activity. Despite using all traded options on those underlyings, we considered a relatively small time frame: from December 2018 until March 2019. It could be meaningful to analyse if NN trained for a specific sector would yield better results, and if a NN trained on a complete year would alter the outcome of the results.

Nonetheless, the results of this study are extremely promising in terms of NN applicability. Our simplest model, NN model 1, showed to perform at a level similar or better than LSM which is considered a reference in the literature. As to the proposed NN model 2, it outperformed by far both LSM and NN model 1. The results are robust across, underlying, maturities and levels of moneyness.

Should NN continue to gather good results, investment companies could start to use them, in order to price an option and complete a trade instantaneously.

References

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283.

Allegretto, W., Y. Lin, and H. Yang (2001). A fast and highly accurate numerical method for the evaluation of american options. *Dynamics of Continuous Discrete and Impulsive Systems Series B 8*, 127–138.

Bennell, J. and C. Sutcliffe (2004). Black–scholes versus artificial neural networks in pricing ftse 100 options. *Intelligent Systems in Accounting, Finance & Management: International Journal 12*(4), 243–260.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer.

Brennan, M. J. and E. S. Schwartz (1977). The valuation of american put options. *The Journal of Finance 32*(2), 449–462.

Broadie, M., P. Glasserman, and G. Jain (1997). Enhanced monte carlo estimates for american option prices. *Journal of Derivatives 5*, 25–44.

Bunch, D. S. and H. Johnson (2000). The american put option and its critical stock price. *The Journal of Finance 55*(5), 2333–2356.

Carr, P., R. Jarrow, and R. Myneni (1992). Alternative characterizations of american put options. *Mathematical Finance 2*(2), 87–106.

Chen, X., J. Chadam, L. Jiang, and W. Zheng (2008). Convexity of the exercise boundary of the american put option on a zero dividend asset. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics 18*(1), 185–197.

Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pp. 192–204.

Choudhary, M. A. and A. Haider (2012). Neural network models for inflation forecasting: an appraisal. *Applied Economics 44*(20), 2631–2635.

Cremers, M. and D. Weinbaum (2010). Deviations from put-call parity and stock return predictability. *Journal of Financial and Quantitative Analysis 45*(2), 335–367.

Garcia, R. and R. Gençay (2000). Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics 94*(1-2), 93–115.

Geske, R. and H. E. Johnson (1984). The american put option valued analytically. *The Journal of Finance 39*(5), 1511–1524.

Gradojevic, N., R. Gençay, and D. Kukolj (2009). Option pricing with modular neural networks. *IEEE transactions on neural networks 20*(4), 626–637.

Haykin, S. (1994). *Neural networks*, Volume 2. Prentice hall New York.

Hutchinson, J. M., A. W. Lo, and T. Poggio (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance 49*(3), 851–889.

Jang, H. and J. Lee (2019). Generative bayesian neural network model for risk-neutral pricing of american index options. *Quantitative Finance 19*(4), 587–603.

Kelly, D. L., J. Shorish, et al. (1994). Valuing and hedging american put options using neural networks. *Unpublished manuscript, Carnegie Mellon University*.

Kim, I. J. (1990). The analytic valuation of american options. *The Review of Financial Studies 3*(4), 547–572.

Kohler, M., A. Krzyżak, and N. Todorovic (2010). Pricing of high-dimensional american options by neural networks. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics 20*(3), 383–410.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Kuske, R. A. and J. B. Keller (1998). Optimal exercise boundary for an american put option. *Applied Mathematical Finance 5*(2), 107–116.

LeCun, Y. A., L. Bottou, G. B. Orr, and K.-R. Müller (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer.

Liu, S., C. Oosterlee, and S. Bohte (2019, Feb). Pricing options and computing implied volatilities using neural networks. *Risks 7*(1), 16.

Longstaff, F. A. and E. S. Schwartz (2001). Valuing american options by simulation: a simple least-squares approach. *The review of financial studies 14*(1), 113–147.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*(4), 115–133.

Moshiri, S. and L. Brown (2004). Unemployment variation over the business cycles: a comparison of forecasting models. *Journal of Forecasting 23*(7), 497–511.

Pacelli, V., V. Bevilacqua, M. Azzollini, et al. (2011). An artificial neural network model to forecast exchange rates. *Journal of Intelligent Learning Systems and Applications 3*(02), 57.

Parkinson, M. (1977). Option pricing: the american put. *The Journal of Business 50*(1), 21–36.

Rogers, L. C. (2002). Monte carlo valuation of american options. *Mathematical Finance 12*(3), 271–286.

Rotundo, G. (2004). Neural networks for large financial crashes forecast. *Physica A: Statistical Mechanics and its Applications 344*(1-2), 77–80.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Sullivan, M. A. (2000). Valuing american put options using gaussian quadrature. *The Review of Financial Studies 13*(1), 75–94.

Tang, Z. and P. A. Fishwick (1993). Feedforward neural nets as models for time series forecasting. *ORSA journal on computing 5*(4), 374–385.

Tilley, J. A. (1993). Valuing american options in a path simulation model. In *Transactions of the Society of Actuaries*. Citeseer.

Tkacz, G. (2001). Neural network forecasting of canadian gdp growth. *International Journal of Forecasting 17*(1), 57–69.

Xu, B., N. Wang, T. Chen, and M. Li (2015). Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.

Yao, J., Y. Li, and C. L. Tan (2000). Option price forecasting using neural networks. *Omega 28*(4), 455–466.

Zhao, J. (2018). American option valuation methods. *International Journal of Economics and Finance 10*(5), 1–13.

Zhu, S.-P. (2006). An exact and explicit solution for the valuation of american put options. *Quantitative Finance 6*(3), 229–242.