LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

MESTRADO

MATEMÁTICA FINANCEIRA

TRABALHO FINAL DE MESTRADO

DISSERTAÇÃO

AMERICAN PUT OPTION PRICING – A COMPARISON BETWEEN NEURAL NETWORKS AND LEAST-SQUARE MONTE CARLO METHOD

BERNARDO PINTO MACHADO PORTUGAL SEQUEIRA

OUTUBRO - 2019

LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

MESTRADO EM
MATEMÁTICA FINANCEIRA

TRABALHO FINAL DE MESTRADO
DISSERTAÇÃO

AMERICAN PUT OPTION PRICING – A COMPARISON BETWEEN NEURAL NETWORKS AND LEAST-SQUARE MONTE CARLO METHOD

BERNARDO PINTO MACHADO PORTUGAL SEQUEIRA

ORIENTAÇÃO:

RAQUEL MEDEIROS GASPAR
SARA BÁRBARA DUTRA LOPES

OUTUBRO - 2019

Acknowledgments

Agradeço a Deus por todas as oportunidades que me deu na vida, por todos os caminhos que tive a sorte de percorrer e por me dar força nos momentos mais difíceis. Só Deus sabe o meu receio na escolha deste mestrado e das dificuldades que tive ao longo deste percurso todo por ter arriscado mudar de área.

A vida universitária foi uma caminhada longa e árdua, cheia de obstáculos mas também cheia de benesses, e não há ninguém melhor que a minha família para saber a minha devoção em busca de mais conhecimento. Quero agradecer aos meus pais, José Diogo Portugal de Sequeira e Mercês Maria Milheiro de Menezes Pinto Machado Portugal de Sequeira por terem dedicado a sua vida a proporcionar uma educação de excelência aos filhos e por terem planeado a nossa vida em detrimento de prazeres e tentações imediatas sem significado. Tudo o que sou hoje e o que serei no futuro é o espelho da educação privilegiada que me foi oferecida.

Um especial agradecimento aos meus irmãos. Ao meu irmão Rodrigo por ser a minha fonte de inspiração constante, onde desde o primeiro momento de estudo universitário tive a oportunidade de ter um exemplo contagiante de força, perseverança e dedicação. Que não haja dúvidas quanto à admiração que tenho por ele, e quanto eu acredito que o seu nome será recordado durante várias gerações, assim Deus o queira. À minha irmã Catarina que, na minha infância, foi um exemplo de alguém sem medo de mudanças na vida. Por muito risco que essas mudanças pudessem envolver, foi sempre capaz de saltar de país em país em busca de melhor.

Agradeço ao meu amigo Kevin Fernandes por toda a ajuda que me deu com este tema. Meu mentor desde o estágio que fizemos juntos em 2018 na James, o Kevin

é a grande razão para este meu interesse em Machine Learning. Caso esta amizade não se tivesse proporcionado, muito provavelmente esta tese, com este tema, não teria existido. Sem a sua preciosa orientação e conhecimento na área de Machine Learning, este estudo não teria a qualidade e profundidade que tem. Obrigado, Kevin.

Por último, mas não menos importante, um agradecimento especial às professoras que me acompanharam nesta tese. À professora Sara Lopes por desde o início do ano letivo me ter mostrado várias opções de temas para a tese, e, no final de tudo, ter aceite o meu desafio, mesmo sendo um tema de uma área que não pertence. Dedicou-se à minha tese como se da sua se tratasse, mesmo não tendo tempo livre nenhum, e por isso estarei sempre extremamente grato. Obrigado por estar sempre a uma mensagem de distância, disponível para ver e rever o que fosse preciso, a qualquer hora do dia. À professora Raquel Gaspar por ter aceite este tema desafiante e por ter um método de acompanhamento organizado e, simplesmente, impecável. Agradeço-lhe a forma direta e aberta com que lidou comigo, tendo tido a flexibilidade de continuar a acompanhar esta tese enquanto me encontrava fora do país. Sem as professoras, esta tese não seria possível, foram incansáveis nestes últimos meses, fui certamente abençoado nesta escolha.

Bernardo Pinto Machado Portugal Sequeira

Abstract

This thesis compares two methods to evaluate the price of American put options. The methods are the Least-Square Monte Carlo Method (LSM) and Neural Networks, a machine learning method. Two different models for Neural Networks were developed, a simple one, Model 1, and a more complex model, Model 2.

It relies on market option prices on 4 large US companies, from December 2018 to March 2019.

All methods show a good accuracy, however, once calibrated, Neural Networks show a much better execution time, than the LSM. Both Neural Network end up with a lower Root Mean Square Error (RMSE) than the LSM for options of different levels of maturity and strike.

Model 2 substantially outperforms the other models, having a RMSE ca. 40% lower than that of LSM. The lower RMSE is consistent across all companies, strike levels and maturities.

Keywords: Option Pricing, Neural Networks, Pricing Methods, American Options, Machine Learning

Resumo

Esta tese compara dois métodos de pricing de opções de venda Americanas. Os métodos estudados são redes neurais (NN), um método de Machine Learning, e Least-Square Monte Carlo Method (LSM). Em termos de redes neurais foram desenvolvidos dois modelos diferentes, um modelo mais simples, Model 1, e um modelo mais complexo, Model 2.

O estudo depende dos preços das opões de 4 gigantes empresas norte-americanas, de Dezembro de 2018 a Março de 2019.

Todos os métodos mostram uma precisão elevada, no entanto, uma vez calibradas, as redes neuronais mostram um tempo de execução muito inferior ao LSM. Ambos os modelos de redes neurais têm uma raiz quadrada do erro quadrático médio (RMSE) menor que o LSM para opções de diferentes maturidades e preço de exercício.

O Modelo 2 supera substancialmente os outros modelos, tendo um RMSE ca. 40 % inferior ao do LSM. O menor RMSE é consistente em todas as empresas, níveis de preço de exercício e maturidade.

List of Tables

List of Figures

Contents

1 Introduction

The purpose of this study is to compare two different methods to price American Put options. These kind of options give the owner of the contract the right, but not the obligation, to exercise the option at any time, as opposed to European options that only give the right to exercise the option at a pre-defined fixed date. That is, we have an optimal stopping time problem, and a closed form solution to price them does not exist.

This problem was first studied by Brennan and Schwartz [4] and is recovered on the literature ever since. See, for instance an overview of different approximation methods to price American Put options in Zhao [23].

We focus on the comparison of two methods: the Least-Square Monte Carlo Method, a simulation method first presented by Longstaff and Schwartz [15], and a machine learning method, Neural Networks.

Recent studies on Neural Networks explore mainly European options. That is the case for Hutchinson *et al.* [10], the first article where Neural Networks are trained to price options, and Yao *et al.* [22].

This study focuses on American options instead and is based on a larger market dataset when compared to the existent Neural Networks' recent literature.

The rest of the text is organized as follows. Section 2 presents the state of the art on the topic, Section 3 explains the data selection process and its descriptive statistics, Section 4 explains the methodology and also architecture of the Neural Networks, Section 5 presents and discusses the results. Finally, Section 6 concludes and suggests further developments.

2 Literature Review

This dissertation focuses its study on American put options, but an introductory review of European options is done in this section as these types of options are the core of options pricing. As Black and Scholes [2] derived in their article, a closed form valuation formula exists for both European call and put options. We recall the put option price formula:

$$P(x,t) = -xN(-d_1) + k\exp^{-rT}N(-d_2) \qquad , \tag{1}$$

where $P(x,t)$ is the price of a call option, at time t given the underlying price x, k is the strike price, r is the risk-free interest rate, T is the time to maturity and $N(.)$ is the standard normal distribution.

The derivation of this formula is based on some assumptions that are not necessarily true in the stock market, such as no transaction costs nor taxes, constant interest rate, constant volatility and lognormal returns. Many authors tested the empirical validity on these assumptions. For example, Schoutens [18] shows, on multiple data, the empirical distribution of asset returns exhibits fat tails and negative skewness. The author also states that volatility is not constant, changing stochastically over time. Cont [7] also explores the same issues and presents a set of statistical properties of asset returns, as is the case, for example, for absence of autocorrelations and heavy tails.

To what regards American options, Merton *et al.* [16] show that an American call option with no dividends "is always worth more 'alive' than 'dead'.", as the price of the option is always higher than the payout of that option. Even if this conclusion

does not hold for American call options with dividends (because dividends can cause a positive probability of exercising the option before the maturity date), the truly hard question has always been that of valuing put options. It is for put options that the quest for the optimal stopping time is more relevant and there are no closed-form solutions for this pricing problem.

Merton *et al.* [16] show American put options always have a positive probability of prematurely exercising the option (even in a no dividends scenario), and conclude that the valuation of an American put is a complicated analytical task. With no closed-form solutions available, one must rely on either numerical or simulation methods.

Zhao [23] compares eight different valuation methods on a one-dimensional scale. The author studied both call and put options. The numerical methods considered are the binomial tree method, trinomial tree method, quadratic approximation, explicit finite difference and implicit finite difference, and the binomial method has the best overall performance in terms of both time and accuracy. With regards to the Monte Carlo methods, simulated tree from Broadie *et al.* [5], the bundling technique from Tilley [20] and the Least-Square Monte Carlo Method from Longstaff and Schwartz [15] are compared, with LSM showing the best results also in terms of both accuracy and execution time. Although simulation methods tend to be very slow when compared to numerical methods, they are widely used when the complexity of the option grows.

Here we focus on the Least-Square Monte Carlo Method and compare it against a Neural Network based machine learning technique. The Longstaff and Schwartz [15]

Least-Square Monte Carlo Method approach approximates the value of continuation using Ordinary Least Squares to recursively determine the conditional expectation for each time step along a simulated path. The advantages of this method are its accuracy and the ability to adapt to multi-dimensional pricing, while the disadvantage is its execution time.

As previously mentioned, we propose the usage of Neural Networks for pricing American put options. Kelly *et al.* [11] and Kohler *et al.* [12] both studied this theme. When contrasting to this study, Kelly *et al.* [11] used less than 10% of our total observations, while Kohler *et al.* [12] simulated the observations and studied higher dimension options.

Using the definition in Haykin [9]:

> "A neural network 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."

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

There are multiple types of Neural Networks, and the types used for option pricing are: backpropagation networks, Hutchinson *et al.* [10] and Yao *et al.* [22], and feedforward networks, as are Tang and Fishwick [19] and Garcia and Gençay [8]. Following Hutchinson *et al.* [10] and Yao *et al.* [22], approach, we build Neural

Networks with a structure as represented in Figure 1. For option pricing models it is a good fit since we need to train our model with the input variables via a loss function that will try to learn the relationship between the parameters.

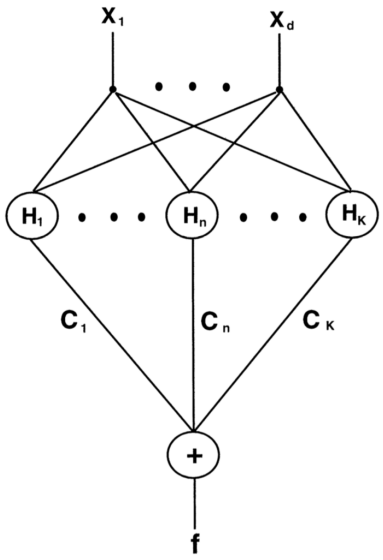

Figure 1: Multilayer Perceptron (from Hutchinson et al. *[10])*

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. Most Neural Networks present a structure with several hidden layers. 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.

The particular structure of the Neural Networks of this study and a more detailed discussion on other characteristics, can be found in the Methodology Section 4.

In the literature there is mixed evidence on whether or not Neural Networks can outperform other methods. Even in European options the remarks are not conclusive. For American options, Kelly *et al.* [11] states Neural Networks can

outperform approximation methods, however, both Hutchinson *et al.* [10] and Yao *et al.* [22] have more conservative conclusions. For European options the authors conclude that the empirical evidence lacks theoretical background. Particularly Yao *et al.* [22] tested different types of Neural Networks that yielded different results when compared to the Black Scholes model.

The advantages on using Neural Networks for option pricing is that this method can be adapted to a multiple underlying option as well as more exotic options, and does not rely on the usual Black Scholes assumptions. Neural Networks learn through data, so they do not make any assumptions on taxes, transaction costs, volatility or asset return distribution.

To the best of our knowledge this is the first study to compare simulation methods with Neural Networks when pricing American options. This study contributes to the literature by using a bigger data set than former papers and by introducing optimization methods for the Neural Networks structures.

3 Data

3.1 Data Description

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 the high market capitalization and large trading volume. A company with a high market capitalization has, on average, a higher trading volume of options than a company with a low market capitalization and the trading volume of the options has an impact on the true value of the options on various maturity and strike levels.

For each option, besides daily close prices on the option itself, we collect daily underlying stock prices, strike prices, maturities, volumes and implied volatilities[1]. Besides these metrics, we also need dividends and interest rates. To what regards the dividends, we retrieve the quarterly dividend paid per share from each company throughout the period studied. 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].

[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 FRED, Federal Reserve Bank of St. Louis.

3.2 Data Treatment and Descriptive Statistics

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 eliminate those observations, as maturity cannot be equal to zero. Finally we eliminated all missing values. From the original 37952 observations we ended up with 21111, which is still a much larger number than what can be found in the literature.

Table 1 presents basic statistics of our input variables - implied volatility, moneyness (equal to the ratio between stock price and strike price), maturity (time to maturity), relative dividends (also known as dividend yield) and the interest rate (risk-free rate).

	Implied Volatility	**Moneyness**	**Maturity**	**Relative Dividends**	**Interest Rate**
Mean	0.296	1.056	113.201	0.034	0.024
Std Dev	0.105	0.162	171.882	0.008	0.001
Min	0.030	0.565	1.000	0.020	0.023
25%	0.235	0.976	15.000	0.025	0.024
50%	0.288	1.025	38.000	0.038	0.024
75%	0.337	1.096	134.000	0.040	0.025
Max	2.862	2.268	779.000	0.048	0.028

Table 1: Statistics of Variables

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. To sum up, below the

threshold are ITM options, and above the threshold are out-of-the-money (OTM) options. Figure 2 shows the frequency of moneyness, and Table 2 shows the percentage of each category in each company. The distribution is heavily centered around approximately 1.

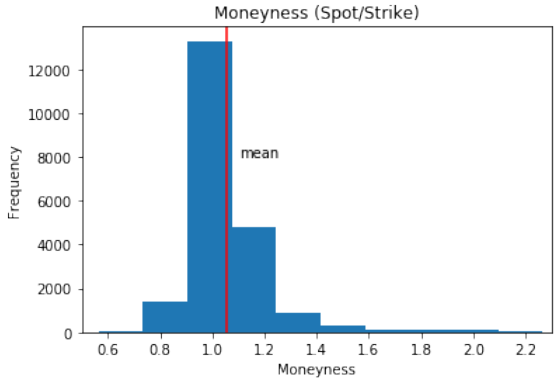

Figure 2: Moneyness Frequency

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

Table 2: Moneyness by Company

We also have many extreme values in OTM options that influence our Machine Learning model. The box plot showing these values can be seen in the appendix, Figure A2. We can observe in Figure 3 that, as expected[3], our maturity is heavily skewed and is almost fully concentrated in maturities below 1 year.

Table 3 shows the percentages that help understand this issue that can influence our Machine Learning model precision when learning with extreme values. The box plot for the maturity can also be seen in the appendix, Figure A3, and emphasizes

[3]Short-term and ATM maturities are, by far, the most traded options in the market.

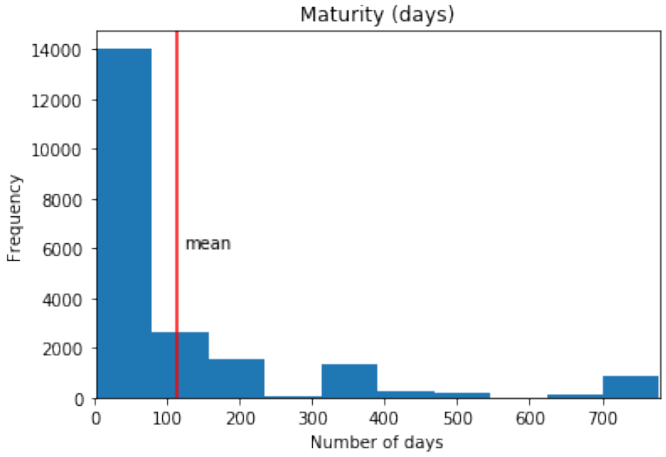

Figure 3: Maturity Frequency

this fact.

<1 Month	1-6 Months	>6 Months
42.72%	39.16%	18.12%

Table 3: Maturity Percentages

In Figure 4 we can observe the surface of the put price with respect to both moneyness and maturity. In the money options are with no surprise the highest valued options, regardless of maturity.

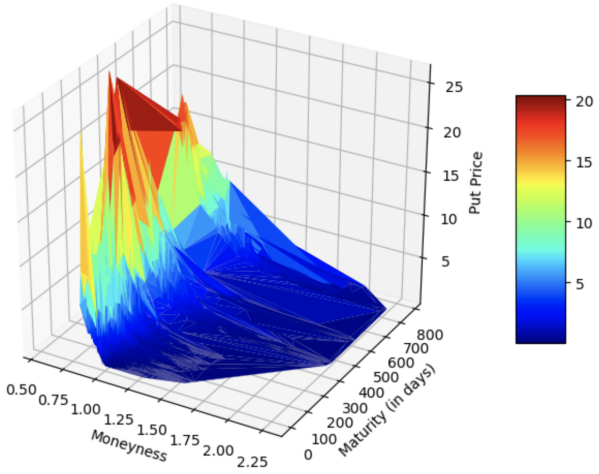

Figure 4: Put Price vs Maturity & Moneyness

In terms of interest rate, we retrieved the daily values for each maturity from the

US Federal Reserve System. Since our sample consists of only three months, there are no big variations in the term structure throughout the 3 months. Moreover, we can see in Figure 5, term structure is quasi-flat.

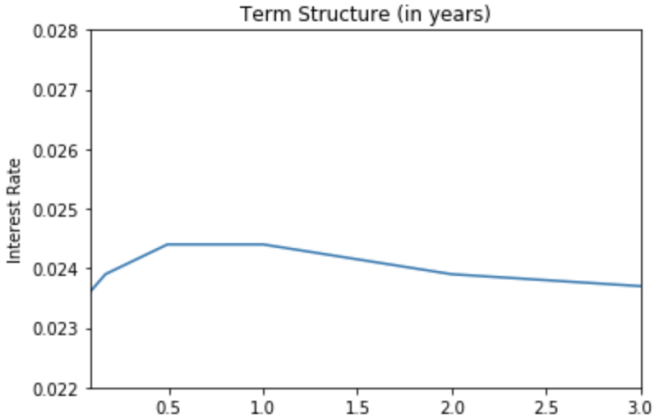

Figure 5: Interest Rate Term Structure at 17_{th} March 2019

Dividend yield values can be seen in Table 4, where the Dividend Yield column shows the average yield in the 3 month sample for each company.

Table 4 shows the mean of each variable per company. As we can observe in our data, each company has different characteristics on each variable. For example, during the period of the data, GM was facing a higher volatility in the stock market, that should be felt at the same time in the option market, thus having the highest volatility between the 4 companies. The difference in each company will positively impact our Machine Learning model, since it won't overfit to the same pattern of data, in the case our observations were from a single company.

Also refer to Table 5 to observe the median values. The differences between median and mean values emphasizes the fact that our data represents mainly ITM options with a low maturity but still contains extreme values that influence the mean of the variables.

	Put Price	Implied Vol	Spot	Strike	Moneyness	Mat.	Div Yield	Int Rate
BAC	1.258	0.306	27.618	26.021	1.090	132.873	0.022	0.025
GM	1.902	0.336	37.319	36.175	1.050	109.743	0.041	0.024
KO	1.364	0.196	47.016	45.827	1.034	115.196	0.034	0.024
PG	2.354	0.221	92.891	90.32	1.033	77.076	0.031	0.024

Table 4: Mean of variables per company

	Put Price	Implied Vol	Spot	Strike	Moneyness	Mat.	Div Yield	Int Rate
BAC	0.620	0.279	28.306	26.500	1.047	49.000	0.021	0.024
GM	1.080	0.319	38.050	36.500	1.024	36.000	0.040	0.024
KO	0.780	0.182	46.960	46.000	1.019	43.000	0.034	0.024
PG	1.650	0.217	91.397	90.050	1.014	35.000	0.031	0.024

Table 5: Median of variables for each company

In order to train and test our Neural Networks we need to have a training set and a test set. As is common in machine learning techniques, the division of both data sets was made randomly, with the training set consisting on 80% of the full data set. Using Kolmogorov-Smirnov tests we have checked the distributions of test and training set are similar, feature by feature[4]. In Table 6 we observe that all features seem to have a high p-value, which allows us not to reject the hypothesis that both sets are drawn from, and represent, the same distribution.

	Put Price	Implied Volatility	Spot	Strike	Maturity
p-value	0.191	0.828	0.572	0.401	0.317
Statistic	0.02	0.01	0.01	0.02	0.02

Table 6: Kolmogorov-Smirnov by feature

In appendix, Figure A6, the distribution of each variable is drawn for both sets.

[4]It tests whether 2 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.

4 Methodology

4.1 Least-Square Monte Carlo Method

The Least-Square Monte Carlo Method is an algorithm that combines standard Monte Carlo methods, by simulating n random paths, with a least-square approach in order to obtain the optimal stopping times for American put options. It is a backwards algorithm that calculates the optimal stopping time comparing the exercise value of the option and its continuation value[5].

Although American options can be exercised continuously, we considered a discrete number k of time steps. The algorithm starts at expiration date t_k and goes to t_0 in order to obtain a cash-flow matrix with all steps for all paths.

In the following we describe the LSM using an illustrative example. We consider 5 paths and 5 time steps.

Spot	**Strike**	**Maturity (in years)**	**Interest Rate**	**Dividends**	**Volatility**
100	105	1	0.06	0.06	0.2

Table 7: Parameters of Put Option

The first step (*Step 1*) consists in simulating the paths of the underlying stock price, as shown in Table 9a.

Step 2 of the method is to compute the cash-flow matrix for time t_k. For t_k, the exercise cash-flow of each path is just the payoff of exercising, or not, the option:

$$CF_{t_k} = MAX(0, K - S_{t_k}) \tag{2}$$

[5]At each point in the time discretization grid the continuation value is the estimated value of holding the option at that point.

Where K is the strike price of the option and S_{t_k} is the stock price at t_k. For each t_k, we will compare the exercise value to the continuation value. Table 9b shows the cash flows at time 5.

The cash flow matrix at time t_k, the exercise value is known, but the continuation value has to be estimated through a regression of the cash-flow at time t_k and measurable functions of the stock price at time t_k. In this study we use a 5 degree polynomial to estimate the continuation value of the option:

$$E[Y|X] = \beta_0 + \sum_{n=1}^{5} \beta_n X^n + \epsilon \tag{3}$$

Where Y is the continuation value, X is the value of the stock in the simulation, and ϵ the residual error. *Step 3* is to estimate the continuation value. In Table 9c we have the values for time $t = 4$. Note that, as in Longstaff and Schwartz [15], we only take the in the money paths.

Regressing for time 4, we have the values for all the β in our regression, and can now obtain the estimated continuation value for each path.

In *Step 4* we compare the estimated value and the exercise value at time 4, as presented in Table 9d, and take the maximum between both values. Note that the exercise value has to be discounted back to the correct time, so we apply a discount rate derived from the interest rate and the number of time steps defined.

The cash flow matrix at time 4 has a particular detail. If in a single path, the option was exercised at time 4, all subsequent cash flows must be equal to zero, since the option is no longer in the buyers hands.

If we recursively repeat *Step 3* and *Step 4* for all t, we will obtain a final cash

flow matrix. By discounting the matrix to $t = 0$ on each path, we obtain the value for each option. In Table 9f we can observe the final discounted matrix giving us a price for each path.

Taking the mean of all values in column one of Table 9f, we obtain the price for the option at time $t = 0$.

Paths	t=0	t=1	t=2	t=3	t=4	t=5
1	100.0	91.48	80.59	65.38	60.99	70.26
2	100.0	110.21	105.49	102.34	97.08	92.42
3	100.0	103.39	120.81	136.38	136.31	132.06
4	100.0	111.08	128.12	160.46	174.79	154.19
5	100.0	92.2	97.88	102.52	109.82	117.21
6	100.0	98.28	85.47	76.93	78.21	82.03

(a) Simulated Paths

Paths	t=1	t=2	t=3	t=4	t=5
1	-	-	-	-	34.74
2	-	-	-	-	12.58
3	-	-	-	-	0.00
4	-	-	-	-	0.00
5	-	-	-	-	0.00
6	-	-	-	-	22.97

(b) Cash Flow Matrix at time 5

Paths	Y	X
1	34.74	60.99
2	12.58	97.08
3	-	-
4	-	-
5	-	-
6	22.97	78.21

(c) Regression params for time 4

Paths	Exercise	Cont.
1	44.01x.988	34.33
2	7.92x.988	12.43
3	-	-
4	-	-
5	-	-
6	26.79x.988	22.69

(d) Exercise vs Continuation at time 4

Paths	t=1	t=2	t=3	t=4	t=5
1	-	-	-	0.00	34.74
2	-	-	-	12.43	0.00
3	-	-	-	0.00	0.00
4	-	-	-	0.00	0.00
5	-	-	-	0.00	0.00
6	-	-	-	0.00	22.97

(e) Cash Flow Matrix at time 4

Paths	t=0	t=1	t=2	t=3	t=4	t=5
1	41.94	-	-	-	-	-
2	11.84	-	-	-	-	-
3	1.59	-	-	-	-	-
4	0	-	-	-	-	-
5	12.65	-	-	-	-	-
6	27.08	-	-	-	-	-

(f) Discounted Cash Flow Matrix

Table 8: LSM Method Illustration

4.2 Neural Networks

4.2.1 Overview

We can think of Neural Networks as an artificial system built to mimic the dynamics of a human brain. Figure 1, in page 5, 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[6]. Note that the architecture of the Neural Networks allow for more than one hidden layer, however, for simplicity this section assumes a Neural Network with just one hidden layer.

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 first hidden layer, let X_i represent a input node, and H_k is the k-th node from the hidden layer, then each hidden node is obtained as,

$$H_k = \phi(\sum_{i=1}^{N_i} w_{k,i} X_i + b_k) \qquad , \tag{4}$$

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

The bias is an extra node added to each layer (except the output layer). It

[6]A structure with more than one hidden layer adds more connections and weights between the hidden layers only.

works as an extra argument to the activation function and is always equal to one, meaning that the learning algorithm does not get affected by its inclusion[7]. 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. These steps will be approached in the subsection focused on the learning algorithm.

It is the activation function that will scale 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 depends on the activation function, with the weighted sum of the hidden nodes as the argument. Now that we have a value for each H_k, the output of the function is given by,

$$f = \phi(\sum_{k=1}^{k} v_k H_k + b) \qquad , \tag{5}$$

where f is the output value, v_k is the weight of the node H_k and b is the bias.

Several activation functions have been used in Neural Networks. The most commonly used activation function is the sigmoid function that can be seen in Figure 6.

However, Krizhevsky *et al.* [13] empirically compared it to a nonlinear function called Rectified Linear Units (ReLUs). The conclusions of the author are that the ReLUs consistently improve the training time of the Neural Networks without increasing its error. Furthermore Xu *et al.* [21] investigated the use of ReLUs and

[7]See the learning algorithm Subsection 4.2.2 for more detail.

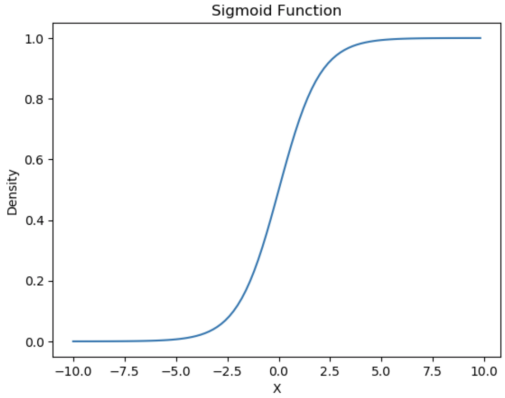

Figure 6: Sigmoid Function

other variants, such as Leaky ReLUs. Leaky ReLUs had consistently better results than the original ReLUs.[8] The equation for Leaky ReLUs is:

$$f(x) = \begin{cases} x, & \text{if } x > 0. \\ a * x, & \text{otherwise.} \end{cases} \tag{6}$$

The difference in both activation functions is the negative part of the function, where, in the case of Leaky ReLUs, a can take a small value, usually in the range (0.01 to 0.2), while for ReLUs, a is zero. Figure 7 plots both functions.

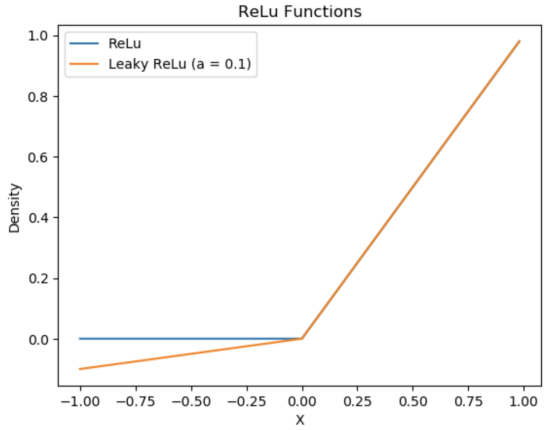

Figure 7: ReLUs Functions

[8]The authors conclude that the rigorous theoretical aspect of these empirical results requires further research.

For our Neural Network, we opted for this last activation function (instead of the traditional sigmoid function), following the most recent literature.

4.2.2 Learning Algorithm

The learning algorithm for Neural Networks was first introduced by Rumelhart *et al.* [17]. In order to get the Neural Networks to learn, we need 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 in the cost function. The cost function rates how good the Neural network did in the whole set of observations. Taking Mean Squared Error as an example of a cost function, we get

$$C(f_j(\theta)) = \frac{1}{n}\sum_{j=1}^{n} e_j^2, \qquad \text{for} \qquad e_j = f_j(\theta) - f_j(\theta)^* \qquad , \tag{7}$$

where n is the number of observations, f_j is the output that depends on the parameters of the model, represented as θ, and f_j^* is the observed value.

The cost function is minimized using a Gradient Descent optimization algorithm. This algorithm minimizes the cost function, by adjusting the parameters in the opposite direction of the gradient. The weights, or parameters, are thus adjusted by the value defined by the Gradient Descent obtained as,

$$\theta(t) = \theta(t-1) - \alpha\frac{1}{n}\sum_{j=1}^{n} \nabla f_{t-1}(\theta(t-1)) \qquad , \tag{8}$$

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.* [14] for an overview of learning rate issues.

Our algorithm starts with random weights drawn from the standard normal distribution. Then for each observation, weights are updated as follows,

$$\Delta w_{k,i+1} = -\alpha \nabla w_{k,i}, \qquad \text{with} \quad \nabla w_{k,i} = \frac{\partial C(f(\theta))}{\partial w_{k,i}} \qquad , \tag{9}$$

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. Apply the chain rule we get,

$$\frac{\partial C(f(\theta))}{\partial w_{k,i}} = \frac{\partial C(f(\theta))}{\partial out_n} \times \frac{\partial out_n}{\partial arg_n} \times \frac{\partial arg_n}{\partial w_{k,i}} \qquad , \tag{10}$$

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 (9).

Following this logic for every observation, the total error of the model will decrease. In this dissertation we consider a variation of this algorithm called Stochastic Gradient Descent, proposed by Bottou [3] to deal with large-scale problems in the

standard Gradient Descent algorithm. In this variation, instead of updating the parameters after iterating the full training set, we set a size for a sample, called batch size, and update the parameters on each random sample. This process accelerates the computation time, and is more efficient to use in large data sets, as shown in LeCun *et al.* [14] and Bottou [3].

In terms of the data used, the input variables of our Neural Networks are different from the ones used in the Least-Square Monte Carlo Method method. For Model 1, we exclude interest rate and dividend yield. In machine learning methods, data is usually scaled to a specific range (0 to 1 for example) in order to increase the accuracy of the models, and allowing our loss function to find a global, or local, minimum. The transformation we use is given by,

$$y_i = \frac{\mathrm{x_i} - X_{min}}{\mathrm{X_{max}} - X_{min}} \quad , \tag{11}$$

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. The output layer consists of the put price, it is also scaled to the same range as the input variables in the case of Model 1.

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.* [17], 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.* [6] 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 Neural Networks, the global minimum is not relevant in practice because it may lead to overfitting the model.

In order to minimize the error of a model, we need to calibrate the model to our data with a beforehand optimization of the parameters. The next section goes through the structure of the models and values given to the learning algorithm.

4.3 Calibration

Starting with Model 1, we use as input variables the Stock Price, Strike Price, Implied Volatility and Interest Rate. A simple architecture with only 1 hidden layer was the choice. In order to train the model, we need to select the number of hidden nodes. The number of nodes should not be chosen randomly, so we did a test for 3 to 10 hidden nodes and we can see the results in Figure 8.

The comparison between the networks was made through a cross-validation test where we compared the networks using the mean square error as the comparison metric,

$$MSE = \frac{1}{n}\sum_{j=1}^{n} e_j^2, \qquad \text{for} \qquad e_j = y_i - y_i^* \ . \tag{12}$$

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. When comparing the values in Figure 8,

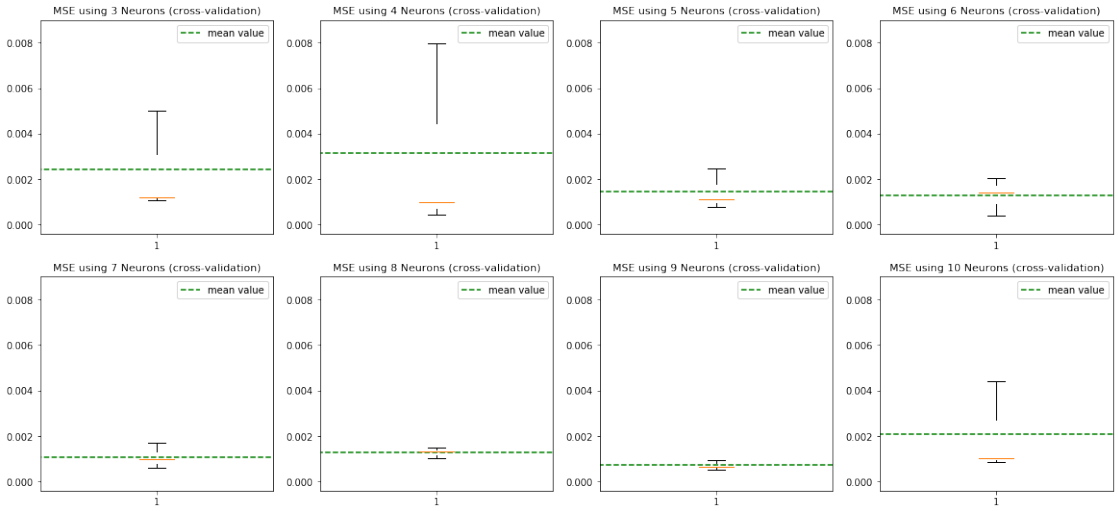

Figure 8: Number of nodes

we concluded that the networks with 7,8 and 9 hidden nodes have the lowest variance in MSE and also the lowest mean values. The structure of Model 1 is now consisting of 1 hidden layer with 4 input variables, 9 hidden nodes and 1 output node.

In order to start the learning process of the networks, we give random weights from the normal distribution to the hidden nodes, with mean equal to zero and standard deviation equal to 0.5. The starting values of the hidden nodes are a random choice, and will immediately change after the first learning cycle ends, making that choice arbitrary. We define the Leaky ReLUs, our activation function, with an $\alpha = 0.1$.

We define the number of epochs at 400. 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. This means that the weights of the model will be updated every 64 observations.

We also have to define a learning rate for the learning algorithm inside the Neural Networks, the Stochastic Gradient Descent. The learning rate is set at 0.005, with

a decay per epoch defined as 1e-6. This means that per epoch, the learning rate will decrease 1e-6. The intuition behind this parameter is that at an initial point, the steps towards convergence are bigger than the steps taken when you are close to converge to the local minimum.

A more complex model was also studied, we call it Model 2. When compared to Model 1, the changes are: number of hidden layers, number of input variables and number of epochs. Model 2 has 3 hidden layers, with 16, 8 and 4 hidden nodes in each structure. When observing the learning curve of the model, the decrease is not as smooth as in Model 1, and does not stop decreasing at 400 epochs, so the number of epochs was increased to 3000, we illustrate this in Section 5.

In order to decrease the error we made some feature engineering to feed more valuable information to our model. As we have 4 different companies in our data set, 4 dummy variables[9] were created called "$Company\_name$" (i.e. $Company\_KO$). The introduction of such variables is an alternative to training a different model for each company. The last variable added is the mean of the put price. This variable represents the mean of put price in the training set[10] by company. Also note that for Model 2 we did not normalize the output variable[11].

In terms of learning curves for Model 1 and Model 2, in Figure 9 we can see the MSE curve from Model 1 going very fast to 0.001. Model 2 also seems to decrease very fast but if we take a better look at Figure 10 we see that there is still space for improvement. Also notable is the high variance when converging to the local

[9] A dummy variable takes only 0 or 1 as values. In our case, $Company\_KO$ would be 1 if the company is Coca-Cola Company and 0 otherwise.

[10] Note that in the test set, the values for the mean of the put price come from the train set.

[11] For this reason the MSE values in Figure 9 have different ranges.

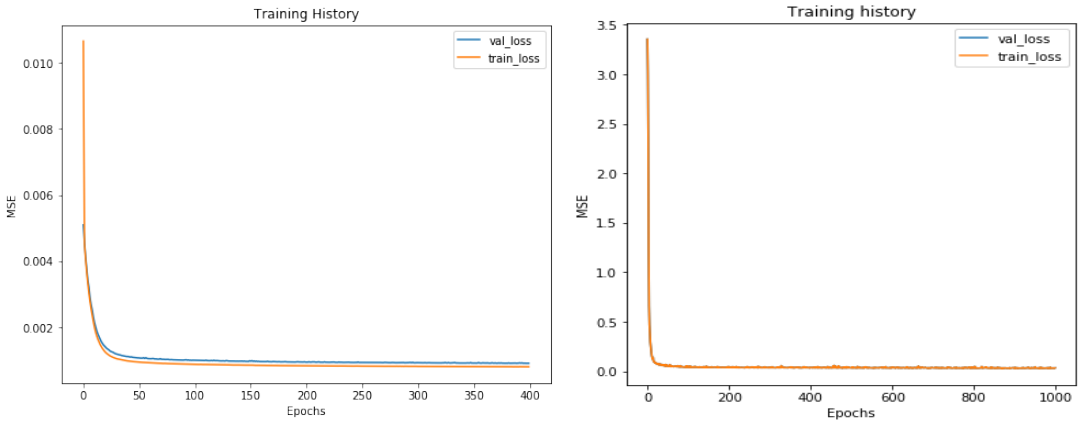

Figure 9: Learning Curve - Model 1 vs Model 2

optimum.

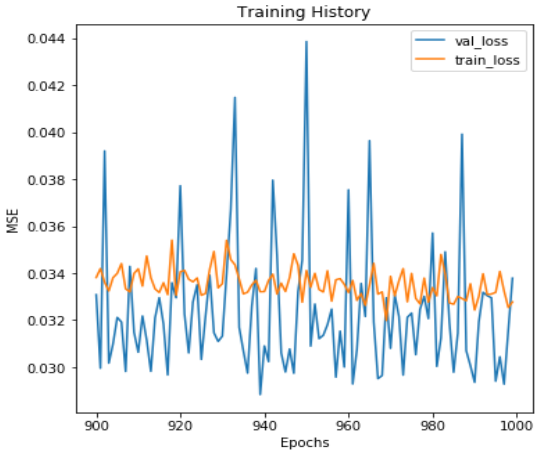

Figure 10: Learning Curve - Model 2 Zoomed

In order to know the importance that our new features may have in Model 2, we use a permutation method on each variable. For each variable we sort the column of the variable randomly, ceteris paribus, run the new predictions in our Model and compare the new RMSE, given by,

$$RMSE = MSE^{0.5} \tag{13}$$

with the original RMSE from the correct test set, in terms of percentual deviation. Figure 11 shows the ordered feature importance. When we mix up the interest rate

values, the total RMSE increases 27%, which is not as substantial as the absence of a correct strike price, which increases the RMSE by approximately 6000%.

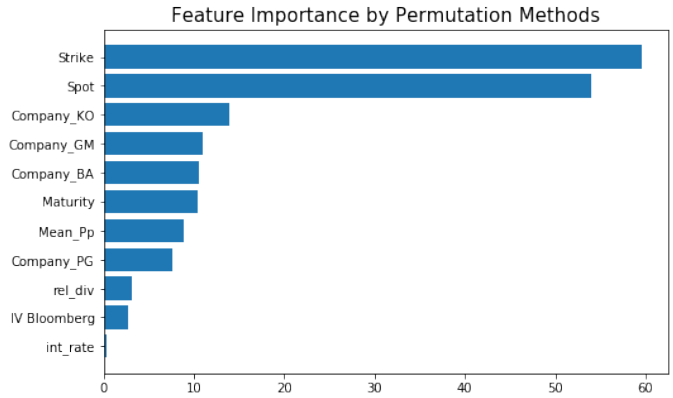

Figure 11: Model 2 Feature Importance

To what regards the LSM, for each option, the number of paths generated was 1000, and the number of total points in the grid for time is 50. As said previously, the OLS function is a 5-degree polynomial.

Having the models fine tuned, we are now able to compare the results of all models.

5 Results

In this section we compare our Neural Networks results with the Least Square Monte Carlo 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 Model 2 is about 5 minutes due to Tensorflow[12] and Keras[13] modules from python that substantially increase the calibration time. After the calibration time, the prediction per option is immediate. LSM takes about 0.38 seconds to price one option alone. In terms of time, Model 2 has a better execution time than LSM. If we include the calibration time in the pricing of each option, the summed value would be an average of 0.07 seconds, less than 20% of the pricing time per option using LSM. In the case of Model 1, the average value is of 0.03 seconds, 50% less than Model 2.

Model 1	**RMSE**
BAC	0.166
GM	0.215
KO	0.198
PG	0.457
Total	0.223

Model 2	**RMSE**
BAC	0.099
GM	0.163
KO	0.139
PG	0.273
Total	0.161

LSM	**RMSE**
BAC	0.112
GM	0.268
KO	0.302
PG	0.400
Total	0.259

Table 10: Results per Company

[12]Tensorflow is an open-source library used in Machine Learning models, check Abadi *et al.* [1] for a deep understanding on why Tensorflow greatly increases time spent in training any Neural Networks model.

[13]Keras is a Neural Networks 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.

In terms of results we can observe that in Table 10 the RMSE is for Model 2 performs better in with respect to our test set. The RSME can be interpreted as follows: on average, each option price is deviated from the true value by x\$. Procter & Gamble is the company with the highest deviation from the option true value across the Models. In order to investigate this issue, in Table 11 the result are ordered by company and moneyness.

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

Table 11: Results per Company & Moneyness

Procter & Gamble is consistently the worst predictable company across all models, with a particular emphasis on ITM options. Model 2 however has a better RMSE when compared to the other models. This error might be explained by the high volatility and price shocks that occurred to this particular stock in the period studied. Procter & Gamble had an uncommon 2019 start, with news and profit warnings that changed the underlying price at a high rate, which might have impacted the high RMSE in all models.

As predicted, the ITM options are consistently worse in each company, due to the fact that in terms of representation in our data, those options only represent

16% of the whole dataset.

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

Table 12: Results per Company & Maturity

In terms of time to maturity, Model 2 consistently predicts better results across all maturities, with an exception made to BAC and KO in $1-6$ months maturity. Results are presented across companies and maturities in Table 12.

In terms of total RMSE, Model 2 is deviated, on average 0.16$ per option, while Model 1 is deviated, on average, 0.22$ per option, and the LSM has a deviation of 0,26$ per option. This means that for our randomly selected test set, the Neural Networks did outperform a simulation method that, as said in the literature review of option pricing, is one of the most accurate methods to price American options.

6 Conclusion and Further Research

Neural Networks have advantages and disadvantages, when it comes to option pricing. They need a lot of data to train, which means that more exotic options (Over the Counter options have lower volume trades than common options) cannot be priced as fairly as more traditional options.

In terms of time, while Least-Square Monte Carlo Method produced a good amount of time to price a single option, if a more complicated jump process, for example with stochastic volatility, is added to the LSM, the model takes a higher amount of time to price a single option. Even using a big amount of paths will increase exponentially the execution time.

Should the Neural Networks gather good results, private investment companies can start using them, in order to price an option and complete a trade instantaneously. Once calibrated, the models have an immediate execution time, and this study demonstrates that Neural Networks can beat traditional methods in terms of performance. One should not forget that Neural Networks learn with past data, so 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 terms of limitations of the analysis here presented, the number of options used, although larger than most studies in the literature, is still scarce when compared to Neural Networks models used in image recognition and other fields of study.

Further research could focus on trying to apply Neural Networks to options with more than one underlying, and, more interestingly, trying to get the valuation of the greeks in order to hedge the position of a trade. Also, this study trained a Neural

Networks with a data set from 4 different companies, from 4 different sectors and on a small period. It would be meaningful to research if Neural Networks trained for a certain sector would have better results, and if a Neural Networks trained on a complete year would change the outcome of the results.

References

[1] M. Abadi, 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)*, pages 265–283

[2] F. Black and M. Scholes [1973]. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654

[3] L. Bottou [2010]. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer

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

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

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

[7] R. Cont [2001]. Empirical properties of asset returns: stylized facts and statistical issues

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

[9] S. Haykin [1994]. *Neural networks*, volume 2. Prentice hall New York

[10] J. M. Hutchinson, 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

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

[12] M. Kohler, 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

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

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

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

[16] R. C. Merton *et al.* [1973]. Theory of rational option pricing. *Theory of Valuation*, pages 229–288

[17] D. E. Rumelhart, 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

[18] W. Schoutens [2003]. *Lévy processes in finance: pricing financial derivatives*

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

[20] J. A. Tilley [1993]. Valuing american options in a path simulation model. In *Transactions of the Society of Actuaries*. Citeseer

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

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

[23] J. Zhao [2018]. American option valuation methods. *International Journal of Economics and Finance*, 10(5)

Appendix A

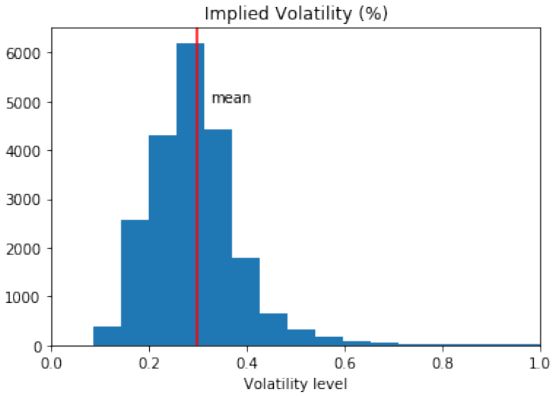

Figure A1: Volatility Density

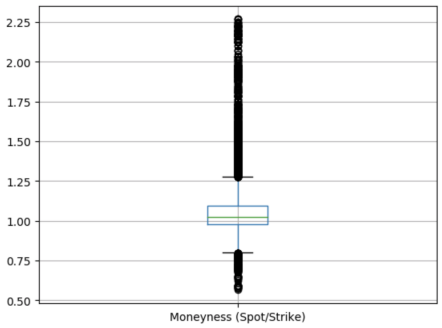

Figure A2: Moneyness Boxplot

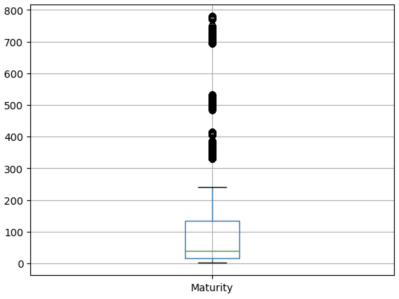

Figure A3: Maturity Boxplot

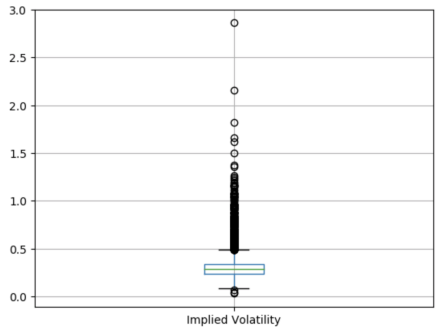

Figure A4: Volatility Boxplot

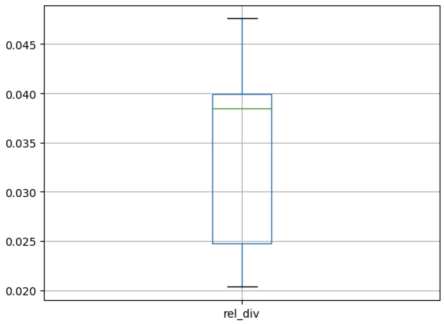

Figure A5: Dividend Yield Boxplot

	Put Price	Spot	Strike	Maturity	absolute_div	rel_div	int_rate	Implied Volatility	Moneyness (Spot/Strike)
Put Price	1.00	0.09	0.25	0.45	0.13	0.12	0.36	-0.18	-0.46
Spot	0.09	1.00	0.97	-0.07	0.90	0.07	-0.00	-0.33	-0.08
Strike	0.25	0.97	1.00	-0.12	0.88	0.09	-0.05	-0.37	-0.29
Maturity	0.45	-0.07	-0.12	1.00	-0.09	-0.05	0.75	-0.12	0.32
absolute_div	0.13	0.90	0.88	-0.09	1.00	0.49	-0.06	-0.20	-0.11
rel_div	0.12	0.07	0.09	-0.05	0.49	1.00	-0.14	0.20	-0.11
int_rate	0.36	-0.00	-0.05	0.75	-0.06	-0.14	1.00	-0.16	0.27
Implied Volatility	-0.18	-0.33	-0.37	-0.12	-0.20	0.20	-0.16	1.00	0.24
Moneyness (Spot/Strike)	-0.46	-0.08	-0.29	0.32	-0.11	-0.11	0.27	0.24	1.00

Table A1: Correlation Matrix

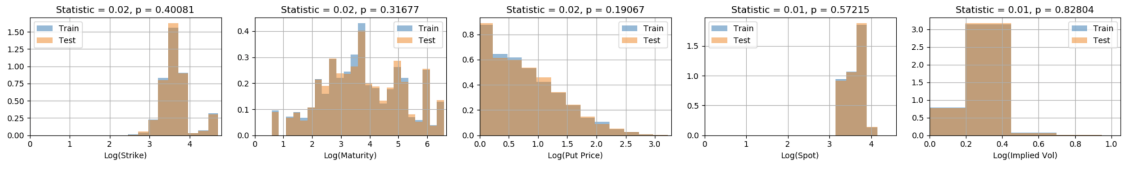

Figure A6: Kolmogorov-Smirnov by feature