What are the advantages of pricing American options using artificial neural networks?

Auteur : Lesuisse, Martin
Promoteur(s) : Hambuckers, Julien

Faculté : HEC-Ecole de gestion de l'Université de Liège
Diplôme : Master en ingénieur de gestion, à finalité spécialisée en Financial Engineering
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/13745

HEC LIÈGE
Management School - Liège Université

WHAT ARE THE ADVANTAGES OF PRICING AMERICAN OPTIONS USING ARTIFICIAL NEURAL NETWORKS ?

Jury :
Promoter :
Julien HAMBUCKERS
Readers :
Ashwin ITTOO
Bruno VAN DER SCHUEREN

Master thesis submitted by
Martin LESUISSE
In order to obtain the diploma of
Master in Business Engineering
with a specialised focus on
Financial Engineering
Academic year 2021/2022

Acknowledgements

First of all I would like to thank my master thesis supervisor Dr. Julien Hambuckers, professor of Finance at HEC Liège – Management School of the University of Liège, who was always ready to listen and available to help if needed. Then, I would also like to thank Dr. Ashwin Ittoo, professor in the field of Business Analytics and Supply Chain Management at HEC Liège, and Mr. Bruno Van der Schueren, head of FX desk at KBC banking and insurance, for agreeing to review my master thesis. Without these three persons, I would not have been able to discover such an exciting subject at the intersection of machine learning and derivatives pricing. Finally, I would like to thank my family and loved ones for supporting me and lifting my spirits when needed through the difficult times that have passed.

Table of contents

Abstract

In this research work, I will try to see if it is possible and advantageous to use artificial neural networks to predict American options, which are more difficult to predict than European options because of the possibility of early exercise. As there are only numerical or approximation methods available, the neural network is a perfect fit, as it is non-parametric and able to capture some extremely complex non-linear functional relations. Once the neural network is set up with an efficient structure, it will also be possible to vary the input features to gain information on the real contribution of the latter on the efficiency of the model.

This research includes definitions and demonstrations of key concepts in the field, a literature review of current knowledge, practices and trends on the subject, a construction of an efficient neural network structure to address the pricing problem and various feature tests on this network, each network being compared with its predecessors but also with the chosen benchmarks: the Black-Scholes model and the Cox Ross Rubinstein binomial tree model.

The main conclusions of this research work are that, once the right neural network structure was found, the use of the ANN to predict American options consistently outperformed its benchmarks. What this means for managers is that machine learning, and neural networks in particular, may be worth investigating for implementation, especially in a context where there is access to sufficient data to train the network properly. My research also shows that if one is in a context where one has to predict in real time many American option prices, then neural networks are advantageous. Indeed, once the learning phase is over, the prediction is instantaneous, contrary to the iterative method of the CRR binomial tree. This advantage can be massive in attempts to leverage the pricing algorithms.

In terms of conclusions for the academic side, this work shows that there is a need to continue to develop techniques for pricing American equity options using neural networks, and that one should not focus solely on European options in the belief that the latter are easier to tackle. I also demonstrate in this work that including the dividend yield in the neural network inputs increases the predictive power of the neural network. This parameter, too often omitted, can make a big difference by itself. My research also shows that taking the interest rate into account increases the predictive power, although a little less than the dividend yield, and confirms that volatility (in my case implied volatility) is very important in the input features.

However, I also find that some features do not add value. This is the case for the volume, which once added to the network does not increase its predictive power (and also makes the training phase more complex), and the open interest, who deteriorates the results.

So there are indeed advantages to using neural networks to predict American options. These advantages are the accuracy (by outperforming benchmarks such as the Black-Scholes or the binomial tree model), the taking into account of parameters that are sometimes difficult to integrate, the fact that a Put and a Call can be priced with the same algorithm, the fact that in the money, at the money or out of the money options can be priced efficiently with a single algorithm, and the instantaneous computational speed once the network has been trained However, there are also disadvantages, namely the learning phase can be long, the fact that one sometimes has to perform trial and error techniques to see what changes improve the network or not, the fact that one needs a lot of good quality data to train the network and the fact that a neural network is a black box that is difficult to analyse. It is up to each person to weigh up the pros and cons of each argument.

Introduction

Context

My thesis is set in a context where two elements meet: the fact that American options pricing has no closed-form solution and the fact that machine learning and more particularly artificial neural networks are increasingly developed, advanced and efficient for a multitude of tasks.

As far as the pricing of American options is concerned, it is done either by approximation or by numerical methods. For example, one approximation method is the method proposed by Barone-Adesi and Whaley in The Journal Of Finance (1987). The problem with such methods is that they tend to be effective only in certain cases. The method mentioned above has accuracy problems when the dividends of the underlying stock have to be taken into account.

To be more accurate, numerical methods derived from one of these three categories are used: the Cox Ross and Rubinstein binomial tree method, the partial differential equation method and the Monte Carlo method. For example, the mainstream data provider I chose uses a proprietary algorithm based on the Cox Ross Rubinstein binomial tree model.

As far as machine learning is concerned, there have been huge advances in recent years. This field of research had been on standby since the publication of the book "Perceptrons: an introduction to computational geometry" (Minsky & Papert, 1969), which sought to demonstrate that artificial neural networks are fundamentally limited. This is the "AI Winter". In 1970, the concept of Backpropagation was presented by Seppo Linnainmaa under the name Automatic Differentiation in his master thesis, and later in a research paper (Linnainmaa, 1976). However, the research remained at a standstill.

It was in the 1980's that research resumed in full swing when Kunihiko Fukushima discovered the Neocognitron (Fukushima, 1980) (which later inspired Convolutional Neural Networks), John Hopfield discovered Recurrent Neural Networks (Hopfield et al., 1983), and several researchers (notably David Rumelhart, Geoff Hinton and Ronald J. Williams) managed to apply the concept of Backpropagation to neural networks (Rumelhart et al., 1986). At the same time, computers were becoming more and more powerful, which inevitably helped the development of machine learning, a discipline that could require a lot of computing power.

After that, neural network research and applications began to flourish, until the use of deep neural networks in everyday applications in the 2000s.

Obviously, finance, like other disciplines, has not escaped the arrival of machine learning, whether it be for robot advisory, pricing, risk hedging, fraud detection or algorithmic trading. With regard to option pricing, research began in 1993. But why is it that the subject is still relevant today, even though many papers have been published so far? Well, for several reasons. Firstly, as mentioned above, the available computing power is increasing every year, and we know that neural networks need a lot of resources. Secondly, and related to the above reason, neural network techniques are improving every year, with new backpropagation algorithms, new ways of encouraging a network to generalise well, and new more efficient structures. Indeed, the more computing power available, the further one can go in neural network research. Thirdly, the volume of traded option contracts in the world has started to increase drastically again since 2016, from 9.33 billion contracts to 21.22 billion contracts in 2020 according to Statista (2021). This adds an incentive to do more research and to use all the current knowledge to better price these contracts.

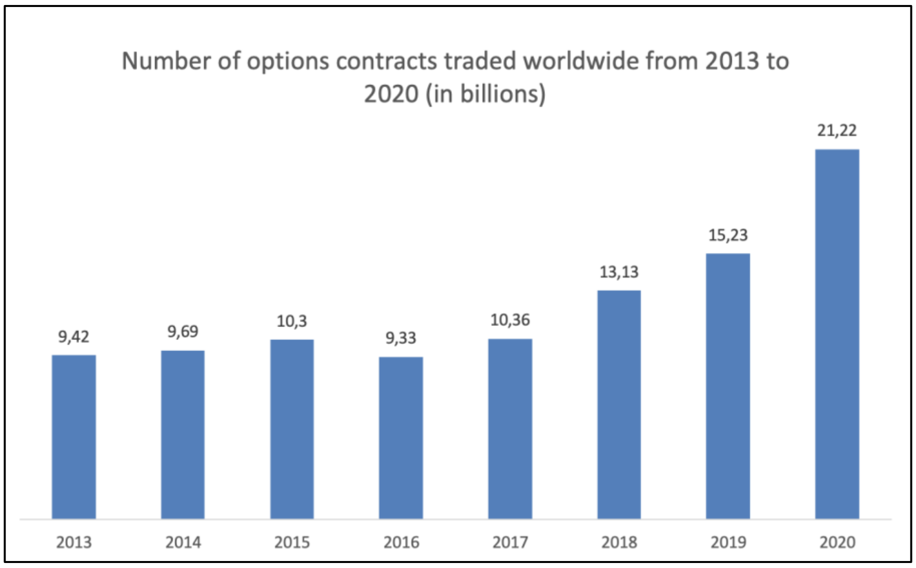


Figure 1 - Annual option volume traded worldwide – data from Statista

As for the continuous increase in the computing power of computers, this is best illustrated by Moore's Law. This law, formulated by Gordon Moore in 1965, states that the number of transistors in electronic circuits doubles every two years. This has been true more or less ever since. However, several researchers predict that this law will no longer hold, and some, such as the Royal Society, claim that it will end in 2021, all other things being equal (for example, without the intervention of new innovations) (Shalf, 2020).

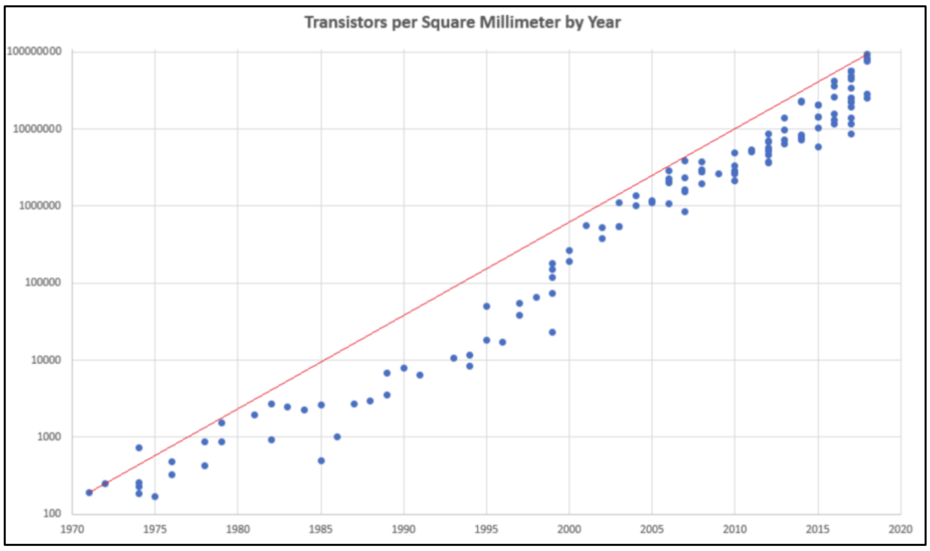


Figure 2 - Moore's law illustration from medium.com (2021)

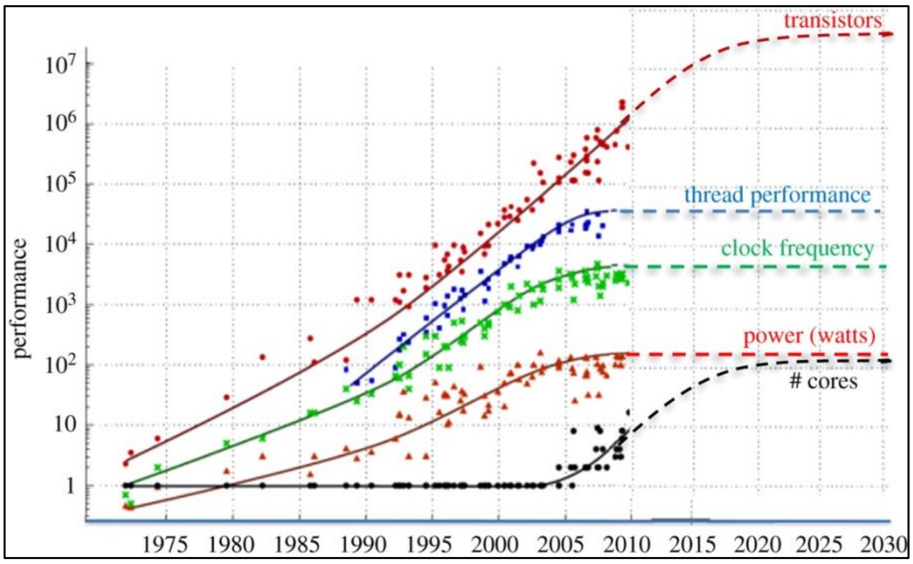


Figure 3 - Royal Society prediction – end of Moore's law (Shalf, 2020).

The fact that the increase in computing power may diminish in the near future is also a reason to explore more efficient numerical methods or approximations, in order to make the most of the computing power we have at our disposal, and not to rely indefinitely on the fact that tomorrow's computers will be more powerful than today's.

On top of that, we are in a context where we have access to more and more data on everything, and this is exactly what is needed to train neural networks.

This is why I think that the current context strongly invites to continue the research in the search of new algorithms of American option pricing, in particular by the use of more or less advanced artificial neural networks.

Main objective

Therefore, the main objective of this thesis would be to investigate the feasibility of using artificial neural networks (ANNs) in order to treat American option pricing problems and to determines the pros and the cons of using them instead of classical methods. Indeed, ANNs are non-parametric models able to capture some extremely complex non-linear functional relations. Since then, the objective of this research would be to design, feed, train and implement (in python) an ANN capable of pricing efficiently American option given all the necessary inputs. Those inputs will not be generated from scratch but will be real data collected from a generic data provider. Then, the performance and the results of the ANN will be assessed alone and by comparison to the most common - used and classical, i.e. The Black-Scholes and the Cox Ross Rubinstein binomial tree model.

The main problem is that in research work, the options treated are more often European options, which are less difficult to price because they can only be executed at expiration. The American options on the other hand are often considered as more exotic options because they have the characteristic of being able to be executed at any time between the purchase of the contract and the expiration date of the contract, which makes them more complex to price. It is therefore necessary to investigate the possibility of correctly pricing American options with a neural network, and to see if this method brings various advantages, either in accuracy or in calculation time.

Moreover, when the performance of a neural network is evaluated, it is done against the Black-Scholes model or some of its variants, which is not the most appropriate for American options. Here, we will try to compare our results to the Cox Ross Rubinstein binomial tree model, which allows to take into account the possibility of executing one's option before expiration. Therefore, it will be useful to compare the results proposed by our neural networks with the results provided by the binomial tree model.

Research motivations

In terms of managerial motivations, it is important to address this topic because machine learning in general is a source of debate and differing opinions. Some uninformed managers find these methods too approximate and do not want to take the trouble to dig deeper into the subject, preferring to stick to methods coming from classical mathematics. This is understandable given that this field of research is relatively new. Other people, convinced by machine learning (which is a part of statistics), try to apply it and advance the research with promising results. Some algorithmic trading or market making firms use it on a daily basis and do not pale in comparison to traditional investment firms.

Therefore, I think managers need more insights on the subject, if only to explore the possibilities that machine learning and neural networks offer. This technology is becoming so important in everyday life that it is impossible to pass by without even considering it. I think that managers could take this research work as a way of discovering the possibilities offered by neural networks in terms of American option pricing, which would allow them to take a closer look at this subject and why not dig deeper.

From an academic point of view, the gap lies in the fact that most of the research focuses on European options or index options, which are in the minority on the market. Indeed, the majority of stock and equity options traded in the market are American, while European options are more traded over-the-counter and on indexes. Therefore, it would be nice to have more research focusing on the pricing of American options. In addition to this, option pricing research does not often take into account the dividend yield. Indeed, many models omit it for ease of calculation, but in a neural network, we can very well try to add it as an input. This is something I will take into account in this master thesis, as it can be important in obtaining an accurate price.

My research questions are therefore consistent with the managerial and academic motivations. It would provide insights into the pricing of more complex options than European options, it would give managers more incentive to look at neural networks by exposing their pros and cons, and it would open some doors to future academic improvements in the accuracy of results.

Contributions

I hope that this research work will contribute to the managers as well as to the academic world, and that its added value will be the demonstration of the efficiency of neural networks for American option pricing, both in accuracy and in computation time, which could be an interesting indication for a real world application. I think that the added value of this work can also be the systematic comparison of several combinations of parameters in the structure of the neural networks and of several combinations in the features proposed as input to the network, which would allow to direct the research towards a more and more successful path.

For a manager, I think that my work could lead to the investigation of the use of neural networks (or machine learning in general) in financial mathematics projects, both in derivatives pricing and in other areas. If I can prove that a neural network can be as efficient or even more efficient than classical models, it could be very convincing. So I think managers should take a closer look at the technological and algorithmic advances in this area, and I hope that my work in this master thesis can contribute to this.

On the academic side, my research work would be of added value as it would provide new evidence of the effectiveness of the use of neural networks for American option pricing, but also new comments on the use of this or that feature, this or that activation function in the network, or other parameters. This would allow for a broader view of what can and cannot be done with American stock options, instead of continuing to focus on European index options. I also hope to provide a justification for the use (or not) of dividend yield in this pricing problem.

Structure of the thesis and general methodology

In terms of the main steps of this work and the methodology in summary, I will begin this dissertation with a literature review, from the earliest research to what is currently being done, including the important findings that have advanced the research on American options pricing using artificial neural networks. I will relate the divergent or non-conforming opinions of some authors and identify research trends, which will provide me with an informed starting point for my own work. Finally, I will position the objectives of my work in the current state of knowledge.

Secondly, I will explain and define the main concepts that are useful for the proper understanding of this work, namely options, their features and functioning, the Black-Scholes option pricing model, the Cox Ross Rubinstein binomial tree pricing model (as these two models will be used as a benchmark during this work), as well as what a neural network is and how it works.

Thirdly, I will define more precisely the methodology used in this work, both for the qualitative and quantitative part, and then I will justify my choice by explaining why I think this methodology is efficient, while looking at whether it has any flaws, partly based on some of the findings of the literature review.

Fourthly, I will go deeper into the quantitative part, explaining why I chose this or that data, explaining how I was able to extract them and how I analysed them to pre-process them. With this, I will go into detail about the structure of the starting neural network I chose, and why and how I chose it. I will show and explain each parameter of the network, so that everything is clear and the starting point of my analyses is solidly defined.

Fifthly, I will talk about the different variants I have made to the basic neural network and show and explain the results obtained by these. This will allow me to compare them with each other, to see the advantages and disadvantages of certain modifications. Then, I will also compare these results to the results obtained by my benchmarks, to be able to know finally if these networks are as efficient, more efficient or less efficient than the classical models.

Finally, these results will be put into perspective with my hypotheses and the whole of my work but also with the works analysed during the literature review, and will be discussed, before I can bring my very own conclusions.

Literature review

Historical and contemporary literature review

For what we are interested in, namely option pricing by artificial neural network, research began in 1993 with the works of M. Malliaris and L. Salchenberger "Beating the best: a neural network challenges the Black-Scholes formula" and "A neural network model for estimating option prices" (Malliaris & Salchenberger, 1993a and b). Their work focuses on European options, which can only be exercised at expiration, and uses extremely simple neural networks ("shallow neural networks"), including the features chosen by the authors as input (spot, strike, time to maturity, implied volatility, interest rate, lagged option price and lagged spot price), a single hidden layer containing 3 to 5 neurons and the output layer containing the price predicted by the network. In these papers, the two authors find that their neural network gives an option price closer to the market closing price than that calculated by the Black-Scholes model 50% of the time. This is just the beginning and it looks promising.

The Journal of Finance first published a research paper on the subject written by Hutchinson et al. (1994). They worked on European options and use only two inputs, namely moneyness (ratio between spot price and strike price) and time to maturity, and had as output the ratio between the option price and the strike price. This is the first time that moneyness is used as an input, which will become recurrent much later. This work concludes that neural networks are not preferable to parametric derivative pricing formulas, but can be a good substitute when parametric methods fail. This research is promising but limited because it uses only Black-Scholes as a comparison, too few inputs and too limited data (a single instrument on a four-year period).

The first paper on American option pricing came later with "Valuing and Hedging American Put Options Using Neural Networks" by Kelly and Shorish (1994). Although his work focuses only on put options and mainly on their hedging possibilities, it too seems promising, concluding that the neural network is faster and more accurate than approximation methods. However, little precise information on its results and methodology is available, except that it uses strike price, spot price, time to maturity and historical volatility as inputs, which may seem little compared to more recent work.

Time goes by and research continues to progress at a rather slow pace, mainly on European index options, on futures options, but little or not on American equity options.

Some papers propose some innovations as the years go by, like Lajbcygier and Connor (1997) who proposed to use a neural network designed to predict the difference between the price predicted by a classical parametric model and the real market price in the paper "Improved option pricing using artificial neural networks and bootstrap methods", or Anders et al. (1998) who look at the impact of certain neural network input features in "Improving the pricing of options: a neural network approach", concluding that the use of implied volatility provided a good prediction improvement, and that too large networks (more than 3 hidden layers) could negatively impact the results.

Other research is also being done on the use of neural networks to perform another task than pricing directly, such as the calculation of implied volatility. Another study from Zapart (2003) named "Beyond Black-Scholes: a neural networks-based approach to options pricing" uses a neural network in his stochastic volatility model, then uses this volatility in a binomial model to price the option. He also proposes to use an ANN to reverse engineer the Black-Scholes formula. This shows the evolution in the use of neural networks in pricing research, which can be used in many different ways. Again, the conclusion is that his model performs as well as and sometimes better than the Black-Scholes model.

For research on American option pricing, one can find a paper by Pires and Marwala (2005) where they try to price American options using neural networks and support vector machines. A support vector machine is a type of machine learning model capable of performing linear and non-linear regressions and classifications that are particularly useful and efficient for classification problems on small and medium sized complex datasets. The authors conclude that support vector machines (SVMs) are more efficient than neural networks, the average prediction error being lower, even if the training time of SVMs is much longer than that of neural networks. However, their results are still encouraging for the use of neural networks.

After that, research on American option pricing using neural networks is more or less on standby. In the next attempts we can for example a study which uses very efficiently a neural network to predict the implied volatility as well as another neural network to predict the price difference between the classical model used with the so-called implied volatility and the market price (Amornwattana et al., 2007). Their model outperforms the Black-Scholes model, both with the historical volatility and the implied volatility predicted by the neural network.

Finally, two papers concerning American options are coming out: "Pricing of high-dimensional American options by neural networks" by Kohler et al. (2010) and "Deep optimal stopping." by Becker et al. (2019). These papers explain that ANNs can be used differently to price American options, by learning the value function or optimal stopping rules in a dynamic programming environment. In this case, the optimal stopping concerns the time at which we will want to execute our American option, which is useful for option pricing.

Throughout this period, most papers discuss the pricing of European options, and often simplify the issue by omitting certain inputs such as dividend yield. My research work is therefore quite adapted to the current stage of research. I will try to see if neural networks can be used to efficiently price American options, taking into account the dividend yield, and if one can find advantages in using these methods over some more classical ones. I will also take the opportunity to compare the results predicted by the neural network with those predicted by the Black-Scholes but also the binomial tree model of Cox Ross Rubinstein, which allows to predict numerically the American options. The objective of the comparison will be multiple: to know if a model created with a neural network can outperform a classical model in terms of accuracy, but also to know if the prediction speed is really much higher with neural networks after training. The speed of prediction can be very important, especially when implementing a pricing tool that needs to generate several prices simultaneously and continuously over a long period of time. Indeed, a too slow algorithm would prevent real-time pricing.

Theoretical foundations

In this section, I will define the theoretical concepts that are important for this master thesis. Firstly, I will recall what an option is and explain the different features it comprises. Secondly, I will define the different classical models used in comparison with the neural network, namely the Black-Scholes model and the Cox Ross Rubinstein binomial tree model. Thirdly, I will define what an artificial neural network is and the basis of its operation.

Options

To begin with, an option is a financial derivative, which means that its value is derived from the value of another asset, called the underlying. This underlying can be many things, such as a stock (which is of interest to us in this work), but also indexes or other derivatives such as futures or swaps (this list is not exhaustive). An option contract gives the buyer the right (not the obligation) to buy or sell the underlying asset at a certain date in the future and at a certain price, both fixed at the time the contract is created. The right to buy is called a Call option and the right to sell is called a Put option. To buy an option contract, one pays what is called the "premium". This is what we will try to determine.

There are several types of options. European options (also called vanilla options) are options where you can only execute your right to buy or sell the underlying asset on the expiry date of the contract. These are the least difficult options to understand. Secondly, American options are options where you can exercise your right to buy or sell during the entire term of the contract, until it expires. These options are sometimes more difficult to understand and to price, due to the fact that they can be executed whenever you want. This is the exercise we will be tackling in this work. We also have other types of options that are even more exotic, such as Asian options with a floating strike price or Barrier options for which the payoff depends on whether or not the price of the underlying asset has exceeded a certain threshold.

An American equity option contract has several features that I will present below:

- Its type: this specifies whether we are dealing with a call or put option.
- Its price (noted C or P): this is the premium that the buyer of the option pays the seller when he contracts the right to buy or sell.
- The spot price (noted S): this is the price of the underlying asset. In our case, it is the price of a share.
- The strike price (noted K): it is the price defined at the time of the purchase of the option at which one will be able to buy or sell the underlying share at the time of the execution of this right.
- Time to Maturity (noted τ): it is the number of years remaining before the expiration of the contract. For an American option, it is during this period that we have the right to execute our option.
- The rate (noted r): the continuously-compounded interest rate associated with the option. In our case, this interest rate is calculated from a collection of continuously-compounded zero-coupon interest rates at various maturities, collectively referred to as the zero curve.
- Volatility (noted σ): this is the volatility of the underlying stock. We will use here implied volatility, which is the volatility implied by option prices observed in the market (it is thus a non-observable factor). It is used to monitor the market's opinion about the volatility of a particular stock. Whereas historical volatilities are backward looking, implied volatilities are forward looking. It represents the expected fluctuations of the underlying stock over a specific time frame.
- The dividend yield (noted q): this is the projected annualised dividend yield.

These were the main characteristics of the options, and the ones that will be used mainly in this work. However, they also have other very important characteristics, such as the volume, the open interest and the “Greeks” risk metrics (which are partial derivatives of the Black-Scholes formula):

- The volume: this is the quantity of option contract that is traded over a certain period of time, usually one day.
- The open interest: this is used to get an idea of how much money is coming into or going out of the market. Indeed, it measures the number of "open" option contracts on the market, i.e. the number of contracts that have not yet expired (having a buyer and a seller). Therefore, if the buyer sells his option contract to another market player, the open interest does not change. However, if the buyer decides to execute his right to buy (for a Call option) or sell (for a Put option), then the contract ends and the open interest decreases. If a buyer and a seller meet and agree to establish an option contract between them, then the open interest increases.
- Delta (Δ): it represents the sensitivity of the option price to a change in the price of the underlying. Specifically, the variation of the option price relative to a one dollar change in the underlying price. This value is used to "delta hedge" a portfolio. If you own an option on a specific share, you can delta hedge your position by shorting a quantity Δ of this specific share.
- Gamma (Γ) : it represents the sensitivity of the delta to a change in the price of the underlying. Specifically, the variation of the delta relative to a one dollar change in the underlying price. This can help to determine whether our delta hedged portfolio is likely to remain so for a long time or not. The more sensitive the delta is to a price change in the underlying, the more often you will have to rebalance your position to stay delta hedged.
- Theta (Θ) : it represents the sensitivity of the option price to a change in time to maturity. Specifically, the variation of the option price relative to a one day change in the time to maturity. It express the time decay of the option.
- Vega (V) : it represents the sensitivity of the option price to a change in the volatility of the underlying. Specifically, the variation of the option price relative to a one percent change in the implied volatility.
- Rho (ρ) : it represents the sensitivity of the option price to a change in the interest rate. Specifically, the variation of the option price relative to a one percent change in the interest rate.
- There are other more specific Greeks which we will not discuss here, as Lambda and Vomma for example.

$$\Delta = \frac{\partial C}{\partial S}$$

$$\Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial^2 C}{\partial S^2}$$

$$\Theta = \frac{\partial C}{\partial t}$$

$$V = \frac{\partial C}{\partial \sigma}$$

$$\rho = \frac{\partial C}{\partial r}$$

When you decide to exercise your option, you receive the payoff. This is calculated differently depending on the type of option.

- For a Call : $Payoff = \max(0, S_T - K)$
- For a Put : $Payoff = \max(0, K - S_T)$

Here is a quick illustration of how the payoff and profit of an option behaves, whether you are long on a Call, long on a Put, short on a Call and short on a Put. For this simplistic illustration, I have chosen a strike price of 110 and an option price of 10.

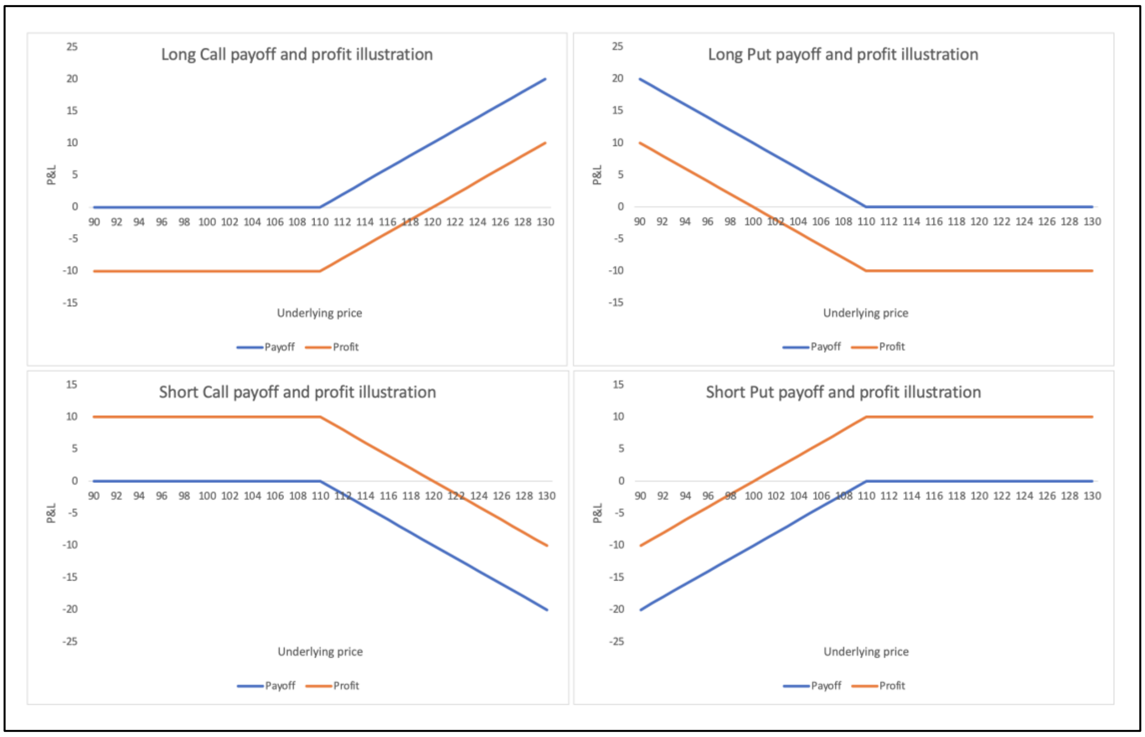


Figure 4 - Option payoff and profit behaviour

It is easy to see that the potential loss is limited when you buy an option, whereas it can be infinite if you sell it. This is due to the fact that the option is an asymmetric contract. Indeed, the buyer pays a premium to acquire the right to exercise his option while the seller receives the premium against the obligation to comply with the buyer's choice.

In addition to this, we can say that the value of an option is intuitively defined by two factors. These two factors are the intrinsic value of the option (the payoff of the option if it were to mature immediately) and the time value of the option (the additional premium due to the time remaining before the option matures). However, calculating the precise price of an option is a complicated challenge depending on the type of option you are dealing with, such as American options.

Finally, an option can be said to be Out of The Money (OTM), At The Money (ATM) or In The Money (ITM). This is important because it expresses three possible states of the option. When we say that an option is OTM, it means for a call that the spot price is lower than the strike price. An OTM option has no intrinsic value, its price only comes from the possibility of gain due to the time remaining before expiration, i.e. its time value. Therefore, the price of such an option is low, and the possibility of big profit is high, but so is the risk of not being able to execute it. For a put, this happens when the spot price is higher than the strike price.

The opposite of an OTM option is an ITM option. In effect, this means for a call that the spot price is higher than the strike price, and for a put that the spot price is lower than the strike price. These options generally cost more because they have an intrinsic value in addition to their time value. For an American option, it can be interesting to execute it before the expiration date of the contract when you are ITM.

Ultimately, an ATM option is one for which the spot price and strike price are equal. This means that the option has a time value but is also close to maybe having an intrinsic value.

Black-Scholes model

The Black Scholes model was proposed in 1973 and is still widely used in the financial industry. It is a partial differential equation used to price European options contracts, as it does not take into account the possibility of exercising the option before maturity. This model is based on two basic principles, namely the risk neutral (or delta hedging) principle and the absence of arbitrage possibilities. This means that stocks are purchased at any moment to hedge the risk of the derivative position and that we cannot use arbitrage to gain money as a portfolio composed of an asset and its hedge will return the risk-free interest rate (Black & Scholes, 1973).

This model is based on several important assumptions:

- The underlying stock price is modelled by a Geometric Brownian Motion with S_t a random variable lognormally distributed, the log-return of the stock normally distributed and W_t a Wiener process (also known as standard Brownian Motion).

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

- The volatility σ is constant
- The underlying stock pays no dividend
- The risk-free interest rate is constant
- There are no transaction costs
- Short-selling is allowed

This lead to the following partial differential equation (PDE) :

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

In this equation, V = V(S,t) is the value of the option, S is the spot price (the price of the underlying asset), t is the time, r is the risk-free rate and σ is the volatility of the underlying.

Conveniently, when talking about European options (that we exercise in t = T), and when respecting all the assumptions above, we can solve this PDE arithmetically to obtain a close form analytical formula.
The formula which gives us the price of a European Call option at time t and which matures in T is the following:

$$C(S,t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$$

And the formula that gives us the price of a Put European option at time t is the following :

$$P(S,t) = N(-d_2)Ke^{-r(T-t)} - N(-d_1)S_t$$

With N the standard normal cumulative distribution function :

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} dz$$

And with d_1 and d_2 being :

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

$$d_2 = d_1 - \sigma\sqrt{(T-t)}$$

Since I take into account the dividend yield in my neural network, I will use a version of the Black-Scholes that no longer takes into account the assumption of "no dividend". The partial differential equation can therefore be rewritten as follows, with q the constant dividend yield :

$$\frac{\partial V}{\partial t} + (r-q)S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0$$

This partial differential equation can be further solved to give an analytical formula for European options, now taking into account the dividend yield. These formulas are presented below, and use the same notation as for the previous formulas.

The price of a Call option:

$$C(S,t) = N(d_1)S_t e^{-q(T-t)} - N(d_2)Ke^{-r(T-t)}$$

The price of a Put option:

$$P(S,t) = N(-d_2)Ke^{-r(T-t)} - N(-d_1)S_t e^{-q(T-t)}$$

With d_1 and d_2 being :

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

$$d_2 = d_1 - \sigma\sqrt{(T-t)}$$

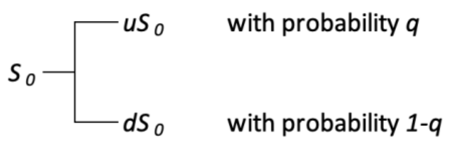
Cox Ross Rubinstein binomial tree model

The other model to tackle is the Cox Ross Rubinstein binomial tree model. The binomial model is a model introduced by Cox et al. (1979) in "Option pricing: A simplified approach" where a diagram is used to represent the paths that can be followed by the underlying price over the life of the option, time step by time step. This model is more general than the Black-Scholes, and when we decrease the duration of the time steps to infinitesimal, we can converge to the solution proposed by the Black-Scholes (what will not be demonstrated in this work).

This binomial tree model uses the no-arbitrage and risk neutral valuation conditions, and has several assumptions :

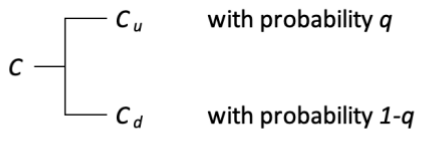
- The underlying price *S* follows a random walk;
- The risk-free interest rate *r* is constant;
- There are no transaction costs;
- Short selling is allowed;

As it is assumed that the underlying price follows a random walk, we have a probability *q* that, after one time step, the underlying price have a rate of return *u-1*, and a probability *1-q* that the underlying price have a rate of return *d-1*. Therefore, after a single time step, the price of the underlying *S* become *uS* or *dS*, with *u* standing for “up” dans *d* standing for “down”.



Under the no-arbitrage condition, *1+r* (with r expressing the risk-free interest rate over one period) must be between *u* and *d*.

To value a call option, we will start with a simple situation where we have only one time step between now and the expiration of the option. We can express the value of the call option as its payoff, i.e. the price of the underlying stock at expiration minus the strike price, but only when this value is positive, otherwise the value is 0 as it means that we do not exercise our right to buy the underlying stock (its price being below the strike price). Therefore, with C representing the value of the call option, we have :



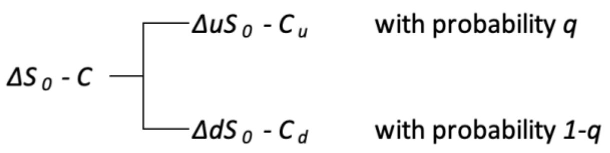
With

$$C_u = \max\,[0, uS - K]$$

$$C_d = \max\,[0, dS - K]$$

It is now possible to create a delta hedged (risk neutral) portfolio in several ways. The way to construct the risk-neutral portfolio here is the way presented by J.C. Hull (2017) in his book "Options, Futures and other derivatives", which I find more telling. Other ways of constructing this portfolio (leading to the same result) are used, notably in the paper by Cox Ross and Rubinstein. For this example, one can be long a Δ quantity of the underlying stock and short a call option on this underlying, with Δ chosen so that the portfolio is riskless. The current price of the underlying is S_0 and the price of the call option is C. We remain in a binomial tree with only one time step, which means that at maturity (in T), the underlying stock will either have a value of uS_0 or a value of dS_0, with $u>1$ and $d<1$. As above, let us assume that the option will have a value of C_u in the scenario where the underlying stock is at uS_0, and a value of C_d in the scenario where the underlying stock is at dS_0.

Therefore, the value of this portfolio can be illustrated as follows:



Since we choose to have a Δ number of stocks in the portfolio so that it is riskless, this means that our portfolios at T have the same value in the “up” and “down” scenario, so we can create this equality:

$$\Delta u S_0 - C_u = \Delta d S_0 - C_d$$

What allows us to find the delta that makes this equality possible:

$$\Delta = \frac{C_u - C_d}{S_0(u-d)}$$

In this way, our portfolio is risk free, and for there to be no possibility of arbitrage, it must have a return equal to the risk free rate. We can then write the present value of the portfolio as follows:

$$PV_{Portfolio} = (\Delta u S_0 - C_u)e^{-rT}$$

Given that the value of the portfolio at t_0 is equal to :

$$\Delta S_0 - C$$

We can write :

$$\Delta S_0 - C = (\Delta u S_0 - C_u)e^{-rT}$$

This gives us, after development and substituting Δ:

$$C = e^{-rT}\left[\left(\frac{e^{rT}-d}{u-d}\right)C_u + \left(\frac{u-e^{rT}}{u-d}\right)C_d\right]$$

Which can be rewritten in this way:

$$C = e^{-rT}[pC_u + (1-p)C_d]$$

By defining :

$$p \equiv \frac{e^{rT} - d}{u - d}$$

With these last two equations, it is now possible to price an option when the underlying stock price is modelled by a one-step binomial tree, when we assume a market with no arbitrage opportunities.

It can be noted that the probability of the underlying share price going up or down, namely *q* and *(1-q)*, does not appear in the equation. This is because the option value is calculated in terms of the underlying share price (the latter being the only random variable on which the option price depends), and the probability of up or down movement is already included in the underlying share price. Next, it can also be noted that the valuation of the option does not depend on the investors' relationship to risk. This is because we are in a risk neutral valuation perspective. Indeed, in this world, taking more risk does not mean that you are expecting a bigger return. It is in the risk neutral world that *p* (having a value always between zero and one) can be interpreted as the probability of an up movement of the underlying share (with $u > e^{rt}$). This is easily proven by expressing the expected stock price like so :

$$E(S_T) = puS_0 + (1-p)dS_0$$

By replacing *p* by the expression above, we obtain :

$$E(S_T) = S_0 e^{rT}$$

This shows that when we take p as the probability of up movement, the price of the underlying grows on average by the risk free rate, and this is what we expect from a risk neutral world.

Now that all the "basics" are in place, very few modifications are needed to take into account several time steps as well as the dividend yield.

Firstly, instead of using *T* in the formulas as before, we will use Δt, i.e. the duration of a time step. Indeed, before our unique time step was the duration of the option until maturity. Now, Δt is the duration of the option's maturity divided by the number of steps in our binomial tree. It is easy to see that it is sufficient to repeat the situation of a one-step binomial tree (with Δt instead of *T* in the formula) for each step starting from the end of the tree and calculating in each node of the tree by moving backward. This will not be demonstrated here.

Second, we need to know how we define *u* and *d*. This is where a parameter comes into play that we have not yet mentioned, namely the volatility of the underlying σ. Indeed, one can intuitively realise that the extent to which the underlying stock price moves up or down is related to the volatility. The greater the volatility, the greater the size of the price movement. Since the variance of returns during Δt is defined by $\sigma^2\Delta t$ and the variance of a variable *X* is defined by $E(X^2) - [E(X)]^2$, with *E* the expected value, and since during Δt we have a probability *p* that the underlying stock has a return of *u-1* and a probability *1-p* that it has a return of *d-1*, we can write :

$$p(u-1)^2 + (1-p)(d-1)^2 - [p(u-1) + (1-p)(d-1)]^2 = \sigma^2\Delta t$$

Replacing *p*, we get :

$$e^{r\Delta t}(u+d) - ud - e^{2r\Delta t} = \sigma^2 \Delta t$$

By using the series expansion of e^x, we obtain the binomial tree formulas :

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = e^{-\sigma\sqrt{\Delta t}}$$

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

These expressions are valid in the risk-neutral world and in the real world.

Finally, to take into account the continuous dividend yield *q*, we have to think that: the full return received in a risk-neutral world is *r*, but the dividends offer a return *q*, so the capital gain returns *r*-*q*. Therefore, the expected value of a stock after a *Δt*, which is $S_0e^{(r-q)\Delta t}$ can also be written :

$$S_0 e^{(r-q)\Delta t} = puS_0 + (1-p)dS_0$$

So that

$$p = \frac{e^{(r-q)\Delta t} - d}{u - d}$$

But since we are dealing with American and not European options, we have to check at each node of the binomial tree if exercising the option is an optimal choice or not. To do this, as we progress backwards from the end of the tree, we must compare at each node the value given by the formula of the binomial tree with the value of the payoff obtained if we exercise our option at this node, and take the larger value. If the payoff value is higher than the value given by the binomial model equation, it means that the exercise of the option is optimal at this node.

We have now presented all the tools available to create a numerical method capable of using the Cox Ross Rubinstein binomial model to predict an American option.

All that is required is to define a number of steps to be used (generally, the literature, whether it is the Cox et al. (1979) paper or Hull's book (2017) recommends the use of a number of steps greater than 30). Then we calculate our *Δt* by dividing the time to maturity *T* by the number of steps. After that, we simply calculate *u*, *d* and *p* using these three formulas:

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = e^{-\sigma\sqrt{\Delta t}}$$

$$p = \frac{e^{(r-q)\Delta t} - d}{u - d}$$

This allows us to construct the "price tree", tracing the evolution of the underlying stock price from t_0 to T as illustrated below:

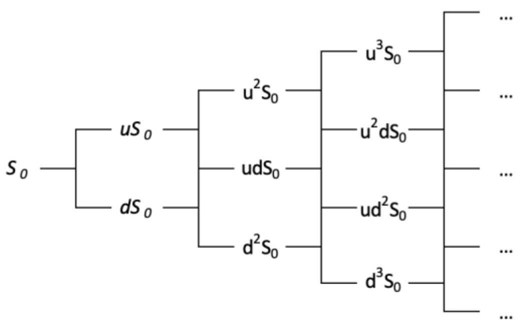


Figure 5 - illustration of the underlying's price tree

Once this price tree is obtained, it is sufficient to create an option price tree, to calculate the final values of this tree by taking the payoff of each final node and to start calculating backward "layer by layer" each node up to the first one by using the formula of the one step binomial tree, i.e. :

$$V = e^{-r\Delta t}[pV_u + (1-p)V_d]$$

With V the value of the option, as the formula is valid for a Call or a Put option. And this is how the Cox Ross Rubinstein binomial model is defined and works.

Artificial neural networks

Finally I will explain what an artificial neural network is and how it works. Artificial neural networks (more often simply called neural networks), are a sub class of machine learning algorithms. The difference with "deep" neural network is that the latter has more than one hidden layer of neurons. But what is a neural network ? It is a mathematical structure loosely inspired by the human neurons. As in our brain, a single neuron is useless; the power of this structure comes from interconnectivity between the neurons, organised in layers.

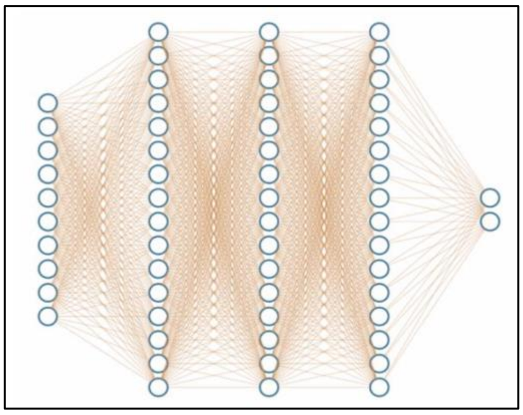


Figure 6 - Deep neural network representation, (Kinsley & Kukieła, 2020)

On *figure 5*, you can see a representation of a neural network with one input layer of 10 neurons on the left, followed by 3 hidden layers of 16 neurons each, and finally the output layer of 2 neurons on the right, with all the connections in orange.

Before going further, I will define some terms that are useful to understand what we are talking about:

- A feature is a variable that we will give as input to the neural network, for example the price of the underlying stock of the option (the spot price). But we do not give the feature directly, we give some realisations of it.
- A realisation of the feature is a value that it has taken or that it could take, for example 10 dollars for the spot price. In general, we will build our data tables with each column representing the realisations of a single feature.
- A sample is a row of this data table. It contains one realisation case for each feature.
- The batch size is a hyperparameter defining the number of samples we will pass to the neural network before we update its internal parameters with the optimiser.
- The number of iterations is the number of batches that must be passed through the network to have entered the whole training set.
- An epoch designates the passage of the whole training set in the neural network.

So to better understand those terms, here is an example. We can use the whole training sample as a batch, or use batches which are fractions of the training sample. If we have a training sample of 1000 realisations of each features and we define a batch size of 10, then in an epoch it will take 100 iterations to pass the whole training sample, and the parameters of the network will be updated 100 times (at each iteration).

Now let's dive in the details of the artificial neural network structure. Think about each neuron as a container, with one number inside of it, typically (but not always, as it depends on the activation function) between 0 and 1. For the input layer, each neuron will contain a realisation of a feature. Then, each neuron of this layer is linked to each neuron of the next layer. We can think of these links as the path that a number encapsulated in a neuron of a certain layer will take to the other neurons of the next layer. As it moves from one neuron to the next one, the number is multiplied by a weight *w* specific to that link.

Therefore, to compute the values contained in one neuron of the second layer (which is the first hidden layer), we have to take the values of each neuron of previous layer it is connected to, multiply all of those values by the associated weight, and sum them all together. But that is not it, as each "destination" neuron also contains a bias term *b* that have to be added to this sum. Finally, we input this value into an activation function (either a sigmoid function like the logistic function, a more convenient and efficient function like the ReLU function, which stands for Rectified Linear Units function, or one of its variants) to "compress" the value (between 0 and 1 for the logistic, between 0 and + infinity for the ReLU), to obtain the value of one neuron. Those steps need to be repeated for each neuron of each layer until reaching the output neurons. This whole process is the "feed-forward" part, as you feed the model with inputs, that propagates forward till reaching the output. But how are these weights and biases determined? If they are not correctly set, how can the neural network give the expected value in output? Well, to determine these parameters, they have to be tuned. And of course, we will not do this by hand because we quickly realize that there are a lot of parameters to train even for not very developed networks. Therefore, we will start by initialising these parameters in a random way. There are several ways to do this, and it often depends on the activation function that we are going to use on the neuron. For example, we will prefer to use the initialisation of He et al. (2015) to go with an activation function of the ReLU family.

After this initialisation, we will be able to tune these parameters. To do this, we will define a cost function, which will measure the error between the prediction of the neural network and the expected value in output (in this work I will use the Mean Squared Error (MSE), that measure the average squared deviation of our predicted value from our real value), and the whole objective will be to minimise this cost function by tuning our parameters by the use of an optimisation function.

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

Basically, the optimisation function compute the gradient of the cost function, which gives the direction of the steepest ascent on the cost function (i.e. of how much we need to modify the weighs and biases to increase the most the function). Therefore, to minimize our cost function step by step, we simply have to subtract from our weights and biases vector the opposite of the gradient of the cost function. As the parameters are changed, the cost function too, and the gradient too! So we have to repeat the process until finding the global minimum. Different optimisation functions have different techniques to converge more or less quickly to the global minimum of the cost function and to avoid getting stuck in a local minimum for example. Finally, all these modifications need to be propagated back to the network, what is done by a backpropagation algorithm.

When we use neural networks, we can distinguish two distinct phases: the learning phase and the prediction phase. It is during the learning phase that we give the neural network the realisations of our features composing the training sample as well as the corresponding output, so that the network can tweak the parameters in order to match the combination of inputs with the desired output. This is the optimisation part of the network, which is done with the help of the optimiser. If this learning phase has gone well and the network has learned from our data, we move on to the prediction phase, where we will give new realisations of our features as inputs to the network but not the corresponding outputs. Indeed, we let the network give us its predictions, which we can then compare with the data we expected. This allows us to see if our network has been able to generalise what it has learned from our data and if it has not overfitted the training sample. If our network has predictive power, we can measure it with the same cost function we used for training, and compare the values between the learning phase and the prediction phase. If the error is higher during the prediction phase, it means that the neural network has some problems generalising and may have overfitted during the training phase.

But what is overfitting? Overfitting is when the network takes "too much" from the training set, and becomes good at only calculating values from the training set. This prevents it from generalising to data it has never seen. The network becomes too specific to the training set. The opposite of overfitting is underfitting. This happens when the neural network has not learned enough from the training set data. To have a good predictive power, we will try to place ourselves between the two, so that our algorithm has learned enough to have a predictive power, but not too much to be able to generalise to other realisations of inputs. Here is an illustration of this phenomenon (from medium.com).

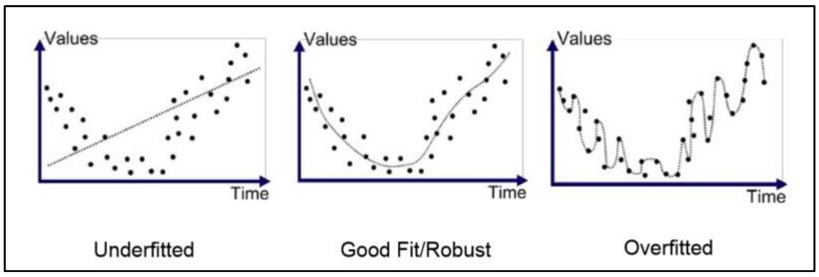


Figure 7 - illustration of different fittings – from medium.com

Overfitting can be strongly reduced with the help of several regularisation techniques, such as early stopping, technique that I will use in my neural networks. To practice early stopping, we will need a third data sample, namely a validation sample, composed of data that are neither in the training sample nor in the test sample. We will give this validation sample to the network during the learning phase by telling it to calculate its performance after each epoch on the validation sample. We can then put a condition saying that if the network does not improve its performance on the validation sample during *x* epochs, then we stop the learning phase. This prevents the network from overfitting the training sample.

Finally, I would like to add that the two phases of use of the neural network, i.e. the training phase and the prediction phase, can be well separated but also intertwined. Indeed, one can train a neural network entirely and then use it to make predictions, but one can also continue to train it each time one uses it to make a prediction! This is called online learning. With each prediction, we will modify the parameters of the network to take into account the latest inputs received in the network. This can be very useful to ensure that the network continues to train over time, and to ensure that it continues to be effective in applications where the context is important (for example during a crisis in the financial markets).

Similarities, oppositions and trends in the literature

In the problem of pricing American options using artificial neural networks, we can observe several trends, agreements and disagreements between authors, notably concerning the features used as input to the neural network. We can also observe practices that strongly diverge, such as the aim of the network, the choice of the benchmark used to evaluate the performance of the network, or the type of asset underlying the option. I will elaborate on this in this section.

One of the most important points in terms of agreement and disagreement concerns the features that we will input to our network. Indeed, authors choose the features they use, and these vary greatly.

First of all, in terms of agreement, all authors are unanimous on the fact that we must use the strike price and the spot price. However, the way of inputting them differs according to the works. Some input them as two distinct features, such as Malliaris and Salchenberger (1993b) or Kelly and Shorish (1994). However, this seems to be an older practice, as the trend seems to be towards the use of moneyness (*S/K*) for some years now. This is what Palmer (2019) in "Evolutionary Algorithms and Computational Methods for Derivatives Pricing", Zheng et al. (2019) in "Gated deep neural networks for implied volatility surfaces" and many others do. The use of moneyness reduces the number of input features in the neural network, making the learning phase less complex (HUTCHINSON et al., 1994). Moreover, moneyness is a stationary feature, unlike strike and spot price, which helps the generalisation of the network and reduces overfitting (Garcia & Gençay, 2000).

Next comes a feature for which there is both agreement and disagreement. Indeed, everyone agrees on the fact that volatility is an important feature to use as a neural network input. Where opinions diverge is in the choice of the volatility. Indeed, some choose historical volatility, others implied volatility, others the VIX index and others a volatility calculated with various models such as the GARCH. Since the results of a neural network do not always depend only on the features offered as input, but also on how these features are processed upstream, whether and how they are scaled, the structure of the network, the optimisation functions and many other factors, it is not always easy to decide on this issue. For example, Blynski and Faseruk (2006) show that a neural network performs better than the Black-Scholes model when using historical volatility as input, but not when using implied volatility, while Andreou (2008) show that replacing historical volatility with implied volatility as input to a neural network improves its performance. It is therefore not always easy to choose.

On my side, since the option prices calculated by my data provider used the implied volatility as an input, I will also use it as an input feature of my neural networks.

Among the other features, there has been a very recurrent use of time to maturity in recent years, and a possible loss of interest in the use of the interest rate, although this has been used in a number of papers that I have read. As for the other features, many authors try to use other features, without it really being the norm in the research. For example, Montesdeoca and Niranjan (2016), with "Extending the feature set of a data-driven artificial neural network model of pricing financial options", tried to add the open interest, the volume of the underlying and other features. They concluded that the volatility is indeed beneficial, as well as the interest rate and even the volume (which can bring more precision in the calculation of the option price).

Finally, a feature that is never used in neural networks or benchmarks is the dividend yield. Indeed, it would seem that it is more difficult to take it into account in the option pricing process and it is therefore generally discarded from the start. Another reason is that many models are based on pricing options with other underlying assets than stocks, or stocks without dividends. Since I will be dealing with dividend stocks, I will take this feature into account.

In terms of divergent practices, we can see that the use of neural networks in the context of option pricing is not always done in the same way. Some researchers use them to directly price an option, some use them to predict the difference between the result of a parametric model and the market price, others to calculate implied volatility that is later given as an input in different models. Nevertheless, the dominant practice is to use neural networks to directly predict the option price.

Next, I will talk about the choice of benchmarks used to compare the performance of neural networks in research work. Not surprisingly, the Black-Scholes model wins hands down. Indeed, this model is one of the most influential models in the world when we talk about derivatives pricing, and is still widely used today, whether as a calculation tool, for calculating risk indicators (such as the Greeks) or as a communication tool. Moreover, it should be recalled that most of the research is done on European options pricing. As for the research on the pricing of American options, the Black-Scholes is used, sometimes with the Cox Ross Rubinstein binomial tree model or the Barone-Adesi and Whaley model (1987) which allow to take into account the early exercise of the option. On my side, since my data used a method derived from the binomial tree model, I will take this model as my main benchmark, alongside the Black-Scholes model. And as for the choice of the cost function, it varies between the mean absolute error (MAE) and the mean squared error (MSE). In order to penalise larger deviations more strongly, I simply choose the MSE. It is this measure that will allow us to compare the performance of our neural network predictions with those of classical models.

As for the underlying assets and the period of data used, this necessarily depends on the context in which the researchers worked. In older works, the authors have little data, whereas in newer works a greater amount of data is available. While some authors simulate data by various methods like Monte-Carlo simulations (as Hirsa et al. (2019) did in “Supervised deep neural networks (DNNs) for pricing/calibration of vanilla/exotic options under various different processes”), many work on real data, the majority on S&P500 options. Thus, much of the research is working on the pricing of European index options. The small part of the research working on American options (having been cited during this work) uses stocks as underlying. This is what I will also do in this work.

Literature review summary

At the end of the literature review you should summarize the main key concepts and elements developed in relation to your research questions.

According to your topic this summary can take the form of a table, a diagram, a model,...
It is also important to remember the common thread of your work and to make the link with the emprirical part.

For the key concepts and elements on this subject, we can say that American option pricing is a subject that can be quite complex because it does not have a closed form solution due to the possibility of exercising one's right to buy or sell before the expiration date.

As a result, several models attempt to approximate the value of these options by various means. These models have been developed for several decades, such as the binomial tree model of Cox Ross Rubinstein from 1979. In parallel, we are witnessing the rise of successive discoveries in the field of machine learning and more particularly that of neural networks, as well as the gradual increase in the computing power of computers. This allows researchers to think more and more about using these promising technologies in various fields, notably options pricing.

Several attempts have been made over the years, but they are mainly focused on European option pricing on indexes, which is less complex to approach. As for American options, there have been a few promising attempts but no ground breaking. More recently, we have also seen the indirect use of neural networks in pricing problems, such as calculating or predicting implied volatility, learning and predicting the difference between the price calculated by a classical model and the market price, or learning to find the optimal time to execute an American option.

Although the research has progressed a long way, it is clear that the results are more focused on European options and not on the direct pricing of American options by neural networks. However, thanks to the research on European option pricing, we were able to identify some practices, trends and key points that may be useful for American option pricing, such as :

- The increasingly recurrent use of the moneyness (S/K) as an input feature in neural networks instead of inputting the Strike (K) and Spot (S) price separately;
- The importance of other input features of the neural network to improve its results, such as time to maturity (T), volatility (σ) and interest rate (r);
- The possible usefulness of volume as a feature;
- The lack of consideration for dividend yield;
- the selection of the Black-Scholes and the binomial tree model as benchmarks;

Based on these starting points, we will try to create an optimal artificial neural network structure that allows us to start predicting American options. From there, we will sometimes move away from the literature to make gradual changes in structure and inputs to try to improve the network.

What the literature has also taught us is that there are many different ways of doing things with neural networks, and that it is up to us to test the ones that seem most promising and then draw our own conclusions.

Methodology

Methodology choice

In this section, I will discuss the methodology used in my research about the development of the structure of the neural networks, the training phase, the prediction/test phase, the implementation of benchmarks and the comparison with these.

To tackle the American option pricing challenge, I obviously chose to use the quantitative approach. But how did I build the methodology I was going to use? Well, regarding the neural network, I have chosen to use a method that mixes trial and error with the use of findings from previous research, what means that I also adopt a qualitative approach upstream of my quantitative work. Indeed, given that there is no rule to define the structure of the neural network that we are going to use, it is sometimes necessary to try several combinations regarding the number of hidden layers and the number of neurons. However, some findings have emerged over the years, whether in research on the use of neural networks for pricing or not. Once I have defined a starting network structure using the literature and my knowledge, it will be possible for me to modify it to see if this improves the results, whether it is through the change in the number of hidden layers, the number of neurons per layers or the features given as input to the network. This is the trial and error part.

Regarding the methodology used for the training of neural networks, it is based on best practices in the field. For example, implementing regularisation techniques, such as early stopping, allows me to prevent my neural network from overfitting the training sample and to avoid generalisation problems. Another good practice consists in using a random initialisation of the parameters corresponding to the activation function that is used. This (combined with the choice of certain activation functions) avoids the problem of "vanishing / exploding" gradients, which is a problem that occurs as the learning phase progresses. Indeed, during the backpropagation step, further we advance in the lower layers, the more the gradient decreases. This ensures that the connections in these lower layers no longer updates, and that the network can no longer converge towards an optimal solution. This is the problem of the vanishing gradient. In some cases the gradient may increase instead of decreasing causing the exploding gradient problem, that cause the model to quicky diverge from an optimal solution. It was proven in "Understanding the difficulty of training deep feedforward neural networks" that it is possible to reduce the problem of unstable gradients, by associating certain technical initialisations with certain activation functions (Glorot & Bengio, 2010). This study defines the “Glorot” method to associate with the logistic activation function (and others), which was one of the functions mainly used during these years. From our side, given that we will be using activation functions of the Rectified Linear Unit activation function family, the optimal initialisation strategy is that of "He" (He et al., 2015).

Then, once the neural network was successfully trained, it was necessary to determine a methodology for the interpretation of the results. First, I start by measuring the performance of my network on a “test sample” of my dataset, which obviously was not used during the network learning phase (this is out of sample prediction). I therefore run the trained network on the test sample, to obtain a vector of American option price predictions, which I will compare to the real prices in my data. To do this, I calculate once again the Mean Squared Error (MSE), which I can compare to the MSE obtained during the training of the network. If the MSE obtained on the predictions is much higher than that obtained during the learning phase, it means that my neural network does not generalise well what it is supposed to have learned, and that it has in fact probably overfitted the training sample. If this is the case, it is necessary to investigate a possible implementation of new regularisation techniques. If the MSE of my test sample is in the same range as that of the training sample, and seems to be correct, we then need a point of comparison. This is where the benchmarks come in.

So as far as the benchmarks are concerned, they had to be implemented first. The two benchmarks chosen are the Black-Scholes and the Cox Ross Rubinstein binomial tree model (CRR). To implement the Black-Scholes it is quite straightforward as I coded a python function that takes the type of option (Call or Put), the spot price *S*, the strike price *K*, the time to maturity *T*, the volatility *σ*, the interest rate *r* and the dividend yield *q* as inputs, calculates d_1 and d_2 and then applies the Call or Put formula according to the type of option proposed as input. Finally, the function returns the value of the option. For the CRR binomial tree model, I followed the algorithmic method described at the end of the model presentation in the literature review section. I also implemented it in a python function that takes as inputs the spot price *S*, the strike price *K*, the volatility *σ*, the interest rate *r*, the time to maturity *T*, the number of steps, the interest rate *q* and the type of option (Call or Put) and outputs the option price.

Once these two functions have been implemented, I simply pass the sample test to them as input, collect the results and calculate the MSEs in relation to the expected results I have in my initial data. Indeed, this does not only allow me to check if the predictions of my ANN are good, but if they are at least at the same level, if not better than those of classical models whose value we already know. Indeed, it is good to have a neural network that works, but it is even better if it can outperform classical methods. It also allows me to compare the performance of various variations of the neural network with each other and with the benchmarks.

Finally, I can also compare the R^2 and Adjusted R^2 of the artificial neural network models with each other and with the classical models, what can give an additional idea of the importance and contribution of certain features for example. If necessary, graphs can be added to visualise regressions between predictions and target values.

Methodology justification

I think that this choice of methodology, combining qualitative and quantitative, is the right one and can allow me to start with a clear advantage. Indeed, especially concerning the design of the neural network, if I had started with a construction from scratch, I could have lost a lot of time and especially maybe not conclude with an optimal conclusion.

A purely quantitative methodology would have been to start with a certain neural network structure and test all the possible feature combinations to be given as input to the network, before reiterating with other network designs, to finally find the solution that seems optimal. The problem with this methodology is that it is difficult to take all the combinations into account, as there may be too many, and we may miss some elements when drawing our conclusions, such as the way inputs are processed for example, which may strongly influence the results of our networks. Another problem with this methodology is the time it can take to set up. It would probably be impossible to realistically test all of the possible combinations.

This is why a qualitative research and a reflection upstream of the quantitative work can be useful in my work. It saves me time and prevents me from testing possible unnecessary combinations, by being informed about which structures and combinations of features are most likely to work best.

However, it must be admitted that this methodology can be fragile under certain conditions. Indeed, just because I find a result and an assertion in a piece of research work does not mean that it should be taken as truth. As explained in the section on trends, agreements and disagreements between researchers, some assertions are not shared, such as the choice of the type of volatility used in the neural network. And even if a researcher's assertion is not contradicted, this does not necessarily mean that it should be taken as truth.

Therefore, I try to treat the information I find with care, to cross-check it if possible, but above all to make up my own mind, using my common sense and trying by myself to see if I come up with the same conclusions. This is why there is still a part of trial and error in this work, what has to be done to make sure that one does not take certain network structures or input features with closed eyes. Also, if I see too bad or abnormal results, I will not jump to conclusions and will try to go deep to find out the cause of the problem in a rational way.

About the data

Data selection

As regards the choice of data, I chose to work on real data rather than on simulated data. While remaining in the field of research, this will allow me to obtain more concrete and real results, but perhaps a little less precise as real data has often more noise. This also allows me to avoid that the results obtained depend on too many theoretical assumptions on the data generation, which could distort the results by giving very convincing results on the theoretical tests but less good results on the tests on real data.

I was able to select all the features that seemed to be of interest from a large database of a generic supplier. Before starting this work, I therefore made a large pre-selection of data. Firstly, the features I decided to extract from this database were :

- The underlying security id, allowing me to identify the underlying stock to link several options together if necessary;
- The date (t_0) at which the data is calculated and collected;
- The type of option (Call or Put);
- The option price in t_0;
- The spot price (the price of the underlying stock) at t_0;
- The strike price;
- The expiry date of the option (which allowed me to calculate the time to maturity, by calculating the duration between the date t_0 and the maturity date T in years);
- The continuously-compounded interest rate associated with the option. This interest rate is calculated from a collection of continuously-compounded zero-coupon interest rates at various maturities, collectively referred to as the zero curve;
- The implied volatility;
- The volume;
- The open interest;
- The Delta of the option;
- The Gamma of the option;
- The Theta of the option;
- The Vega of the option;
- The forward price associated with the underlying stock and the expiration date of the option;
- The projected annual dividend yield;

Then, in this database, there was data available for the years 1996 up to and including 2015. So I decided to take my data for all the years. The first problem that appeared to me was that, for American stock options, there were several hundred million data available. So I did a first sampling, as it would have taken me at least one 500 gigabytes to store all this data. I decided to take an array of data 30,000 long and 17 wide (the number of features I selected) by random draw for each month of each year, figuring that would be plenty.

This gave me an average of 1000 observations per day in a raw dataset that counted 7,200,000 realisations of my 17 features over 20 years. Of course, this is a raw and unprocessed dataset, and I will not use all of this data in this work.

Data collection

In order to collect this data, I had to access a database provided by HEC Liège - School of Management of the University of Liège. To do this, I created a Python script that connects to the database, and sends several SQL queries to fetch the features I am interested in from several tables. These features are then imported in a Python object of type "Pandas.DataFrame" (it is a structure coming from a data science package called "Pandas" similar to a table, but allowing to perform many more actions in an optimised way).

Then this script was responsible for creating a large general table with my 7,200,000 rows of data by combining all the features I asked it to fetch. To do this, I had to combine each data correctly, like assigning the interest rate to the right options based on the continuously-compounded interest rate table, or changing the maturity date to time to maturity (expressed in years). Once some basic actions are done, and I have my big general table, the script saves it in CSV, which I can use for my work as much as I want.

Although this step does not seem complicated, it took me a long time. Indeed, the organisation of the data in the tables on the server was rather "chaotic". It was not easy to find all the features I was interested in, and even less easy to combine these features into a single, orderly table. It took me a lot of trial and error to get the result I was looking for.

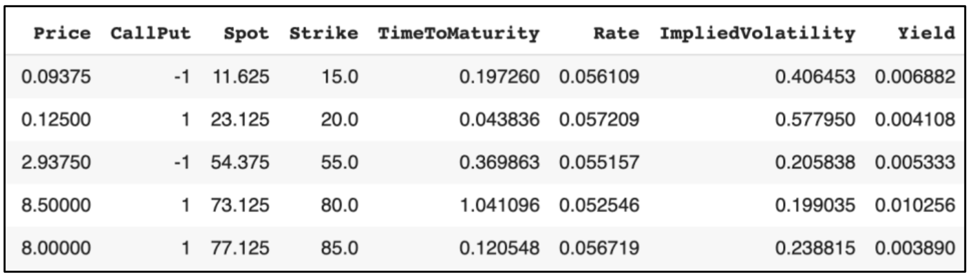
Data pre-processing

For the analysis and pre-processing of the data, there are several steps to be taken.

Firstly, the data must be expressed in the right format, for example the interest rate must be expressed as a proportion of the unit (i.e. between 0 and 1) instead of as a percentage, or the time to maturity must be expressed in years and not in days. Something obvious but very important to remember when dealing with neural networks is that you cannot input anything other than numbers! So you have to express the type of option, Call or Put, by a number. I have chosen to express the Call option by -1 and the Put option by 1.

Secondly, we can remove the missing values. Indeed, the implied volatility and the dividend yield are missing for many options. This is because the data provider did not have enough data to calculate them accurately. Therefore, it marks the data as missing by indicating "*-99*" instead of the value.

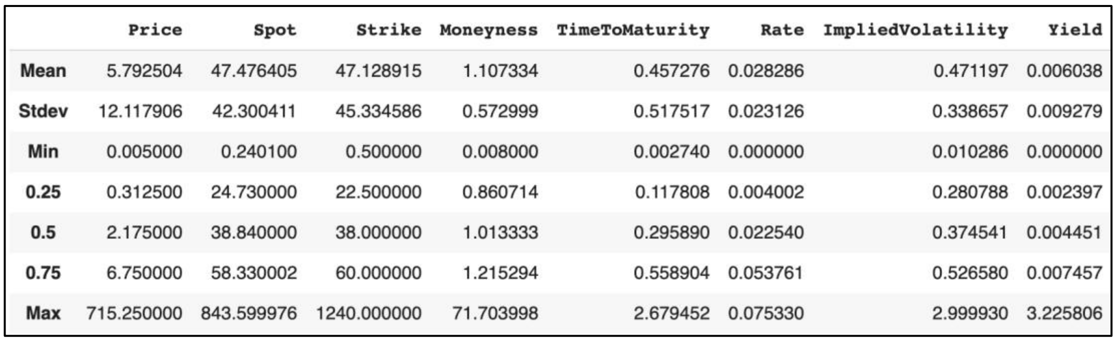
Once this is done, I focus on certain features, namely the type of option, its price, spot price, strike price, time to maturity, interest rate, implied volatility and dividend yield. I am left with a table of 2,763,968 realisations of 8 features.



Price	CallPut	Spot	Strike	TimeToMaturity	Rate	ImpliedVolatility	Yield
0.09375	-1	11.625	15.0	0.197260	0.056109	0.406453	0.006882
0.12500	1	23.125	20.0	0.043836	0.057209	0.577950	0.004108
2.93750	-1	54.375	55.0	0.369863	0.055157	0.205838	0.005333
8.50000	1	73.125	80.0	1.041096	0.052546	0.199035	0.010256
8.00000	1	77.125	85.0	0.120548	0.056719	0.238815	0.003890

Figure 8 - Overview of my main data table

To better visualise the characteristics of these features, one can build a table with some statistics on them, such as the mean, the standard deviation, the correlation with the option price, the minimum, the maximum and the 25th, 50th and 75th percentiles. Before doing so, I also add a "Moneyness" column by simply taking the ratio "spot price over strike price", *S/K*.



	Price	Spot	Strike	Moneyness	TimeToMaturity	Rate	ImpliedVolatility	Yield
Mean	5.792504	47.476405	47.128915	1.107334	0.457276	0.028286	0.471197	0.006038
Stdev	12.117906	42.300411	45.334586	0.572999	0.517517	0.023126	0.338657	0.009279
Min	0.005000	0.240100	0.500000	0.008000	0.002740	0.000000	0.010286	0.000000
0.25	0.312500	24.730000	22.500000	0.860714	0.117808	0.004002	0.280788	0.002397
0.5	2.175000	38.840000	38.000000	1.013333	0.295890	0.022540	0.374541	0.004451
0.75	6.750000	58.330002	60.000000	1.215294	0.558904	0.053761	0.526580	0.007457
Max	715.250000	843.599976	1240.000000	71.703998	2.679452	0.075330	2.999930	3.225806

Figure 9 - Descriptive statistics table

To complement this table, histograms and boxplots are available in the *appendix 1 and 2*. We can see that the distributions are clearly not normal and highly positively skewed (the right tail of the distributions is way longer than the left tail). This means that the median comes early and that after it, the realisations extends quite far from it, as the mean is greater than the median. The first half of the realisations of a feature are compressed "in a tiny interval". This can happen, for example, because you have a lot of "classic" options a little bit out of the money, at the money or in the money, and some extremely out of the money and extremely in the money options that influence the statistics. In a first step, I will try to process all these options at the same time in a single neural network, to see if it can learn to correctly predict options more or less at the money and options deeply out of the money. These "non normal" distributions are not problematic, as a neural network is a non-parametric model (and as we will scale the data before feeding the network, what will be developed later). The boxplots are showing the same conclusions.

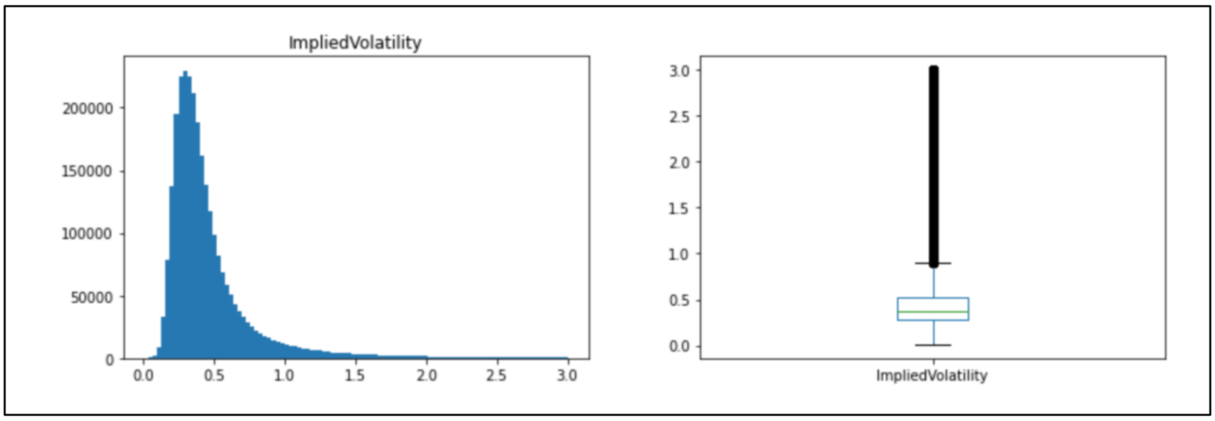
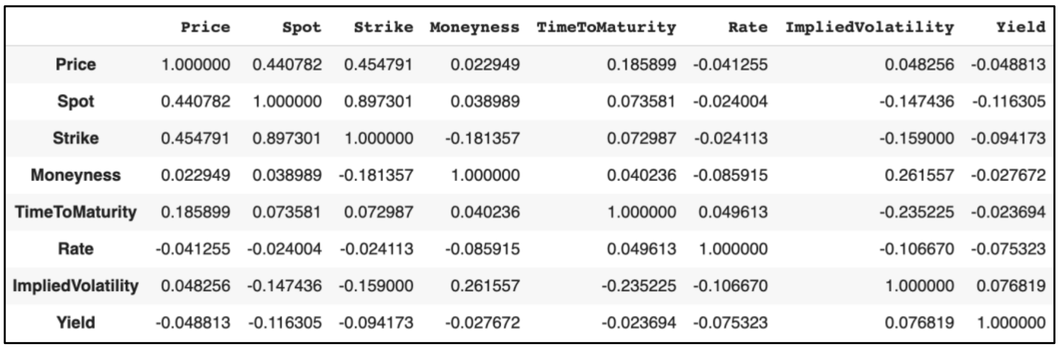


Figure 10 - Histogram of the distribution of the implied volatility and the associated box plot

Another point of interest is the correlation of each feature with the price of the option. At first sight, they seems quite low. But this is where we will be able to evaluate the performance of our model and to add and remove inputs to see the real impact of an input into the model calculations. So we do not have to worry if we see some "light" correlations, as the complex combinations of the inputs will be highly correlated with the output.



	Price	Spot	Strike	Moneyness	TimeToMaturity	Rate	ImpliedVolatility	Yield
Price	1.000000	0.440782	0.454791	0.022949	0.185899	-0.041255	0.048256	-0.048813
Spot	0.440782	1.000000	0.897301	0.038989	0.073581	-0.024004	-0.147436	-0.116305
Strike	0.454791	0.897301	1.000000	-0.181357	0.072987	-0.024113	-0.159000	-0.094173
Moneyness	0.022949	0.038989	-0.181357	1.000000	0.040236	-0.085915	0.261557	-0.027672
TimeToMaturity	0.185899	0.073581	0.072987	0.040236	1.000000	0.049613	-0.235225	-0.023694
Rate	-0.041255	-0.024004	-0.024113	-0.085915	0.049613	1.000000	-0.106670	-0.075323
ImpliedVolatility	0.048256	-0.147436	-0.159000	0.261557	-0.235225	-0.106670	1.000000	0.076819
Yield	-0.048813	-0.116305	-0.094173	-0.027672	-0.023694	-0.075323	0.076819	1.000000

Figure 11 - Correlation matrix of the features

Looking at this table, we can see that the spot price and the strike price are the two most important components in the option pricing process, which is indicated by the strong correlation with the option price. The next important feature is the time to maturity. Together, these are the three most used features in research, and this table seems to confirm that this is not done wrong. Interest rate, implied vol and dividend yield seem to contribute more or less equally to the option price, so it is interesting to consider them and see the impact they will have in the neural network. As for the moneyness, we will have to test replacing the spot and the strike by the latter to get a better idea of the impact it can have. This quick analysis allows us to confirm certain results noted during the qualitative analysis of previous work and research.

To finish the pre-processing of the data, we are going to scale them down, which is a good practice for the use of neural networks. On the one hand, if the input values are not scaled down, we risk having a model that learns with very large w weights. This poses a problem because it makes the model unstable, much less precise and too sensitive to the input values, which can cause generalisation problems. On the other hand, if the output values are not scaled, there is a risk of having very large error gradients, modifying the weights too much and preventing the network from learning in an optimal way.

Normalising your data means transforming it so that it is between 0 and 1. This is very useful since neural networks are sensitive to large values.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Standardising your data means scaling it so that it follows a standard normal distribution. This is also useful, as neural networks sometimes tend to work better with inputs following a normal distribution.

$$z_i = \frac{x_i - \mu}{\sigma}$$

Again, there is no absolute truth in terms of neural networks. Therefore, I chose to standardise my data with the Fit and Transform methods of the *StandardScaler()* machine learning packages Tensorflow and Keras. These two packages are toolboxes that can be used to do a lot of things related to machine learning in python. This is what I am going to use to create, train and use my neural networks.

The dataset is now almost ready to be entered into a neural network. We just need to partition it to create a training sample, a validation sample (allowing me to practice early stopping during the learning phase), and a test sample, to evaluate the performance of the network once the learning phase is over.

In research, two methods are used in equal measure. Some researchers do not pay attention to the temporal dimension of their data and create their samples by randomly drawing from the whole of their data (whether these data are simulated or real, although this type of method is more often found on simulated data). This can be useful, but it can also artificially lower the MSE made by the network in its testing phase. Indeed, with this method it happens for example that we try to predict the price of an option in time *t* with a network that has been trained on options between *t-x* and *t+x*. Since the network has “seen” options "from the future" relatively to time *t*, it contains information that it should not have when trying to predict an option in time *t*.

Others respect the time dimension, and take a training and validation sample from their data between time *t* and *t+x*, then take the test sample on options starting from *t+x*. This allows to see whether the network would be useful in real life or not. However, taking a test sample that goes too far into the future reduces its effectiveness, as the network starts to run out of information on the period during which it is trying to price the option.

In this work, I will of course respect the temporal dimension of the data, in order not to decrease artificially my MSE.

Neural Network structure

In this section, we will explain in more detail the starting structure of the neural network we will use, based on the methodology explained above, mixing qualitative and quantitative analysis. I specify starting structure, because variations will be made later.

To begin with, a study from Anders et al. (1998) concluded that more than 3 hidden layers could degrade the network performance. So I chose to start with this number of hidden layers. Regarding the number of neurons per layer, it used to be customary to decrease the number of neurons as the hidden layers progress. This can be intuitively justified by thinking that the network combines a lot of low-level features into fewer high-level features when going from a hidden layer to the next one. However, this practice was abandoned because it seems that using the same number of neurons in each hidden layers gives equivalent results in most cases, and sometimes even better results. Moreover, as Vincent Vanhoucke explains in his "Stretch pants approach": it is more efficient to use a larger network than necessary and to include methods of early stopping and regularisation. Therefore, I choose to put 100 neurons per hidden layers as the starting quantity, and to set up an early stopping strategy. To do this, I tell my network to stop the learning phase when the error measurement on the sample validation does not improve during 20 epochs, and to recover the parameters (weights and bias) at the time when the sample loss validation was most optimal (i.e. 20 epochs before the learning phase was stopped). Early stopping is already a regularisation method, so we will just start with this one. Since I am implementing an early stopping method, I can set an arbitrarily high number of epochs, knowing that the learning phase will stop before the end. So I choose 1000 epochs. As for the cost function, I choose the Mean Squared Error, which is the standard cost function used in regression problems and which penalises big mistakes more than small ones.

As for the activation function, I choose to use functions from the ReLU family, starting with the ReLU function itself. ReLU stands for Rectified Linear Unit function. This function is defined as follows:

$$ReLU(x) = \max(0, x)$$

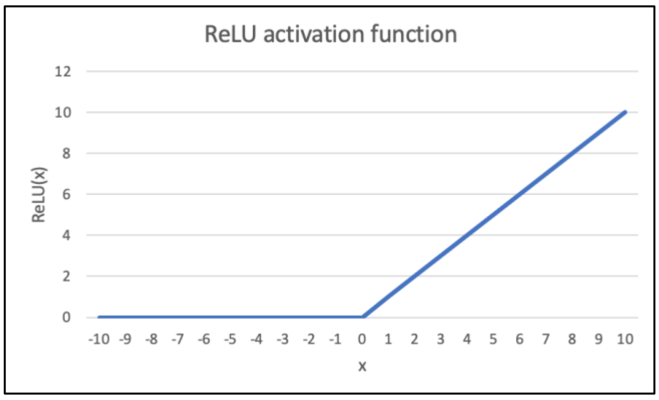


Figure 12 - ReLU activation function

It is one of the most used activation functions because it does not saturate for positive values and it is fast to calculate, unlike the sigmoid activation function, which we will not discuss further here as it is becoming increasingly obsolete.

However, the ReLU activation function has a problem: during the learning phase, neurons die, which means that they only return 0's. This is because the function returns 0 for any negative value. If the weights *w* are such that the weighted sum of the values sent by the neurons of the previous layer is negative for each realisation of the features in the training set, then the neuron dies.

This then prevents the network from training because the gradient no longer passes the dead neurons during backpropagation. To avoid this, there are several functions derived from the ReLU function. This is why I was talking about the activation functions of the ReLU family. One of these functions is the LeakyReLU. It introduces a "leak" parameter α, which represents the slope of the function when the input is negative. It is defined as :

$$LeakyReLU_{\alpha}(x) = \max(\alpha x, x)$$

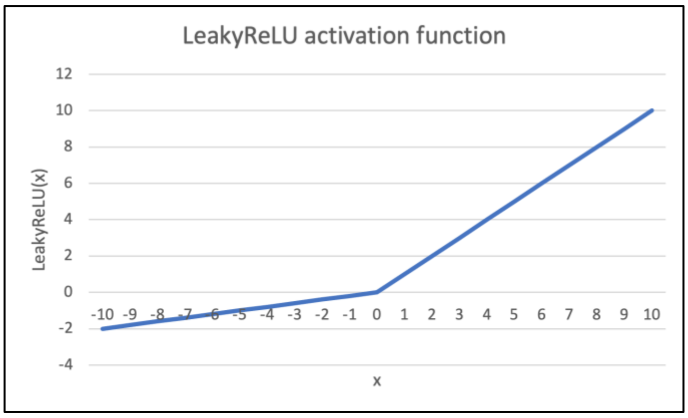


Figure 13 - Leaky ReLU activation function

A study from Xu et al. (2015) named "Empirical Evaluation of Rectified Activations in Convolutional Network” compares this function with ReLU and concludes that Leaky ReLU is always more efficient, even if it can sometimes lead to overfitting on small datasets if we are not careful. An α between 0.01 and 0.2 is typically chosen, the latter being preferred by the authors.

Other variants of the ReLU activation function exist, such as the ELU (exponential linear unit) function, proposed by Clevert et al. (2015), which according to its author decreases the training time of the neural network and increases its efficiency in the prediction phase, or even the SELU, which is a variant of the ELU (but we will not talk about this last one). The ELU activation function is defined as follows:

$$ELU_{\alpha}(x) = \begin{cases} \alpha(e^x - 1) & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$$

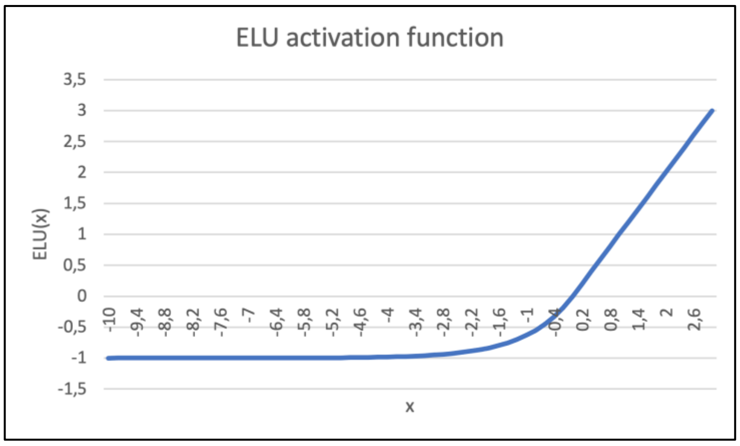


Figure 14 - ELU activation function

In this case I set α equal to 1, which is common with this function. One of the advantages of this function is that it is smooth around 0 unlike the LeakyReLU. This advantage allows to speed up the gradient descent, because it does not "bounce" when we have an *x* close to 0.

Therefore, I will start by using the ReLU function as the activation function in all my hidden layers, and I will explore the differences in performance with the LeakyReLU and ELU functions. Reminder: you also need an activation function on the output neuron. However, as my output data is standardised, there are negative data. So we should not put a ReLU function that would return 0 for all negative values and prevent the network from learning, except if we do not standardise the output. For this output neuron, I am using either the ReLU activation function or a linear activation function depending on the situation.

As for the random parameter initialisation method, which is done before launching the learning phase of the neural network, I will use He's method, according to its authors He et al. (2015) who concluded that it is the best method to use with the ReLU family of activation functions in order to reduce unstable gradient problems. I will not go into further detail about this initialisation technique.

Finally, another element is the selection of which optimiser to use. Many of them are available, such as the classic gradient descent, which we have already discussed as it is the most basic one. To explain a little better what the gradient descent is, we must define the gradient vector of the cost function:

$$\nabla_\Theta MSE(\Theta) = \begin{pmatrix} \frac{\partial MSE(\Theta)}{\partial \Theta_0} \\ \frac{\partial MSE(\Theta)}{\partial \Theta_1} \\ \vdots \\ \frac{\partial MSE(\Theta)}{\partial \Theta_n} \end{pmatrix} = \frac{2}{m} X^T (X\Theta - y)$$

With Θ the vector of model parameters, *X* the whole input training matrix, *y* the output training vector, *m* the number of realisations of our features and *n* the number of parameters. This gradient vector is a vector where we store the variation of the cost function associated with a small change of each parameter one by one. Therefore, this allows us to perform the "gradient descent", taking the vector of the model parameters and subtracting the gradient vector from it like this :

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta MSE(\Theta)$$

With η the “learning rate”, which determines the size of the "step" that we will take along the cost function to “go down” to the minimum. It allows to influence the speed at which we converge towards this minimum, but it can be dangerous to set it to a too large or too small value, under penalty of deviating from the minimum, converging too slowly or being trapped into a local minima.

Now that we know the principle of the gradient descent, we can say that much more optimisation techniques start from the latter, such as the stochastic gradient descent (SGD), which takes at each iteration a random realisation of the training set to calculate the gradient. This technique is faster and requires less computer memory, but it is also less regular by nature. It allows not to get stuck in local minimum but prevents approaching infinitely close to global minima, except if the learning rate is gradually reduced.

We also have the mini-batch gradient descent, which does the same but by taking a random set (called mini -batch) of realisation of our features to calculate the vector gradient. It also does not require a lot of memory and is more stable than the stochastic gradient descent.

We can therefore see that it is possible to use several ideas to improve the gradient descent. But some more interesting ideas exist. For example, Polyak (1964) introduced in "Some methods of speeding up the convergence of iteration methods" the momentum optimisation. With this technique, a "speed" factor and a "friction" factor are introduced. This makes it possible not to go down towards the global minimum of the cost function by taking small regular steps, but rather by taking larger and larger steps when the slope is steep and smaller and smaller when approaching the minimum. It is intuitively like letting a ball descend in a large curve: it will pick up speed on the slope and slow down once the slope is less steep. For this, we introduce the momentum vector m and the hyperparameter β representing a "friction" parameter (usually set on 0.9, a low friction), preventing the momentum of growing too much. We have :

$$m \leftarrow \beta m - \eta \nabla_{\Theta} MSE(\Theta)$$

$$\Theta \leftarrow \Theta + \mathrm{m}$$

With momentum, it is possible that the optimiser exceeds the global minimum by going further, then comes back towards it and exceeds it again, until it reaches the minimum, like a marble in a bowl. This is why it is useful to have frictions.

Now imagine that the cost function is not a bowl but a much more exotic form. Then it could be that the gradient descent goes down rapidly towards the steepest slope, but that this slope does not point directly to the global minimum. The gradient descent would then make a "diversion" before arriving at the minimum, increasing the time of the training phase. This can be avoided by using an optimiser that corrects the trajectory of the gradient descent as it progresses, reducing the gradient vector along the steepest slopes. This method allows the shape of the cost function to be adapted to converge even faster to the global minimum. This method is provided by the AdaGrad algorithm, developed by Duchi et al. (2011). The problem with this algorithm is that it often stops before the global minimum because the learning rate is too scaled down. Another algorithm called RMSProp solves this problem by only taking into account the last gradient in this method, and not the accumulation of all previous gradients. I will not develop this optimiser further.

Finally, there is an optimiser called Adam (for adaptive moment estimation), which combines momentum optimisation and learning rate adaptation (from the RMSProp method). It was presented by Kingma and Ba (2014) in a famous paper named "Adam: a method for stochastic optimization". The authors show that this optimiser is highly efficient compared to others. Moreover, I have seen it used in practice in many works. Here is a quick presentation of the equation of the Adam algorithm :

$$m \leftarrow \beta_1 m - (1 - \beta_1) \nabla_{\Theta} MSE(\Theta)$$

$$s \leftarrow \beta_2 s + (1 - \beta_2) \nabla_{\Theta} MSE(\Theta) * \nabla_{\Theta} MSE(\Theta)$$

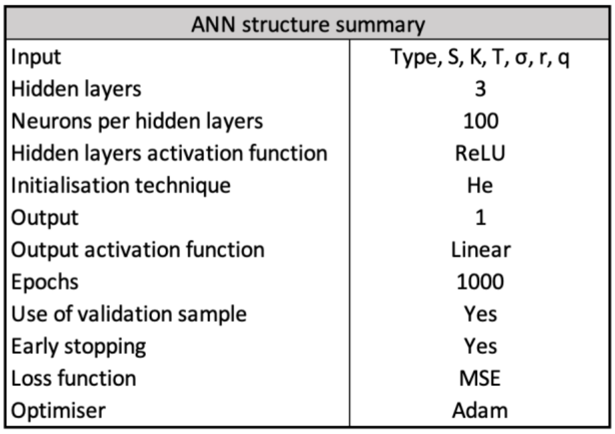
$$\widehat{m} \leftarrow \frac{m}{1 - \beta_1^t}$$

$$\hat{s} \leftarrow \frac{s}{1 - \beta_2^t}$$

$$\Theta \leftarrow \Theta + \eta \widehat{m} / \sqrt{\hat{s} + \varepsilon}$$

With *t* the iteration number, *m* the momentum vector, β_1 the momentum decay parameter (friction), *s* the vector of the squared gradients, β_2 the scaling decay parameter and ε the smoothing term. In this work, I will use this optimiser.

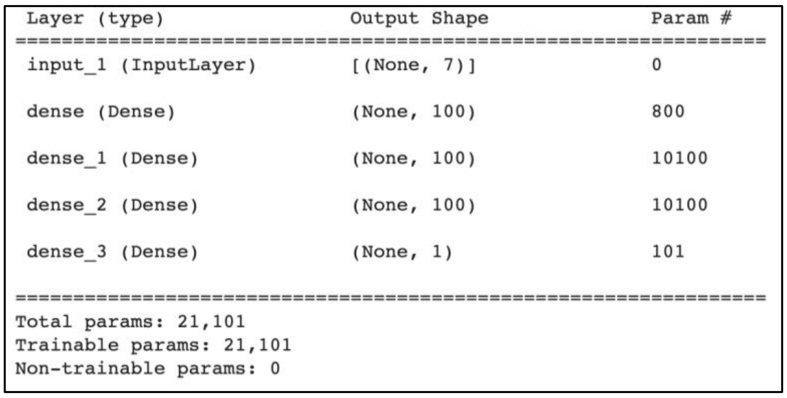
All the characteristics of my neural network are now decided. What I will vary during my research are the number of hidden layers, the number of neurons per layer, the activation functions and the inputs. Here is a table summarizing the structure of my starting neural network:



ANN structure summary	
Input	Type, S, K, T, σ, r, q
Hidden layers	3
Neurons per hidden layers	100
Hidden layers activation function	ReLU
Initialisation technique	He
Output	1
Output activation function	Linear
Epochs	1000
Use of validation sample	Yes
Early stopping	Yes
Loss function	MSE
Optimiser	Adam

Figure 15 - Starting ANN structure

Finally, here is another table highlighting the number of trainable and non-trainable parameters on the structure. The trainable parameters are the weights *w* and the biases *b*. Thus, between the input layer and the first hidden layer, each neuron in one layer is connected to each neuron in the other, which means 700 connections (and thus 700 weights *w*), and each neuron in the hidden layer has a bias term *b*, which means 100 biases. So, on the first hidden layer, there are in total 800 parameters that the network must optimise. On the entire structure, we have 21,101 trainable parameters. The *appendix 3* show the code used to create this starting ANN structure.



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 7)]               0

 dense (Dense)               (None, 100)               800

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 100)               10100

 dense_3 (Dense)             (None, 1)                 101

=================================================================
Total params: 21,101
Trainable params: 21,101
Non-trainable params: 0
```

Figure 16 – Visualisation of the ANN model parameters

Development and results

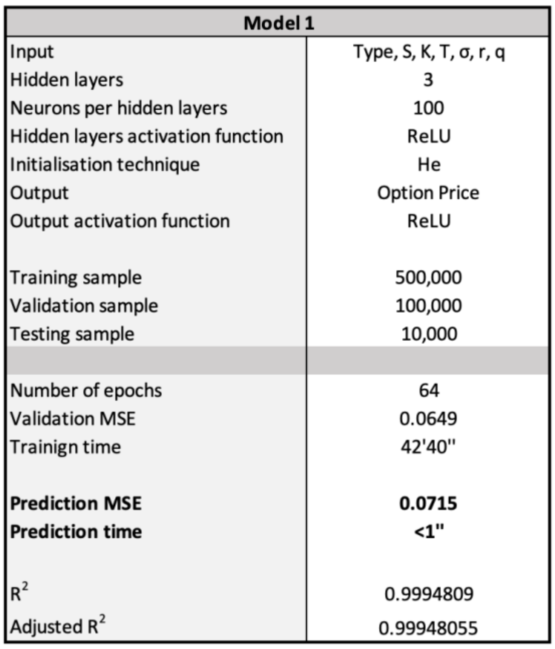
It is finally time to start the training phase of our first neural network. For this first version, I choose to normalise only the inputs and not the price of the options I am looking for in the output. Indeed, I would like the network to directly predict the variable of interest as it is. I also choose to respect the pricing chronology of the options. That is to say that I will train my network thanks to 600,000 realisations of my features taken between *t* and *t+x*. I draw 100,000 of them randomly to be part of my validation sample, and the 500,000 others will constitute the training sample. This way I train and validate with data from the same period. As for the test sample, I take the 10,000 realisations of my features starting from *t+x*. This will allow me to evaluate the capacity of the neural network to generalise on data that arrive further in time than the data used for the training phase.

This neural network took 64 epochs to train on the input sample, with one epoch taking an average of 40 seconds. This gives us a training time of 42 minutes and 40 seconds. The network stopped with an MSE on the validation sample of 0.0649. When tested on the test sample of 10,000 realisations, we get an MSE of 0.07672. This seems to be a good result which already proves one thing: the network seems to lose a little bit of accuracy in predicting events that have not yet happened. I think this is a trend that will be found for every network respecting the data timeline.

As for the time taken to make the 10,000 predictions, it is less than one second. In other words, it is almost instantaneous.

To get an idea, I also decided to calculate the R^2 and adjusted R^2 of the model. The first is equal to 0.9994809 and the second to 0.99948055. This is very good news, showing us that our neural network regression is very good. It expresses that almost all of the variance in option prices is explained by input features.

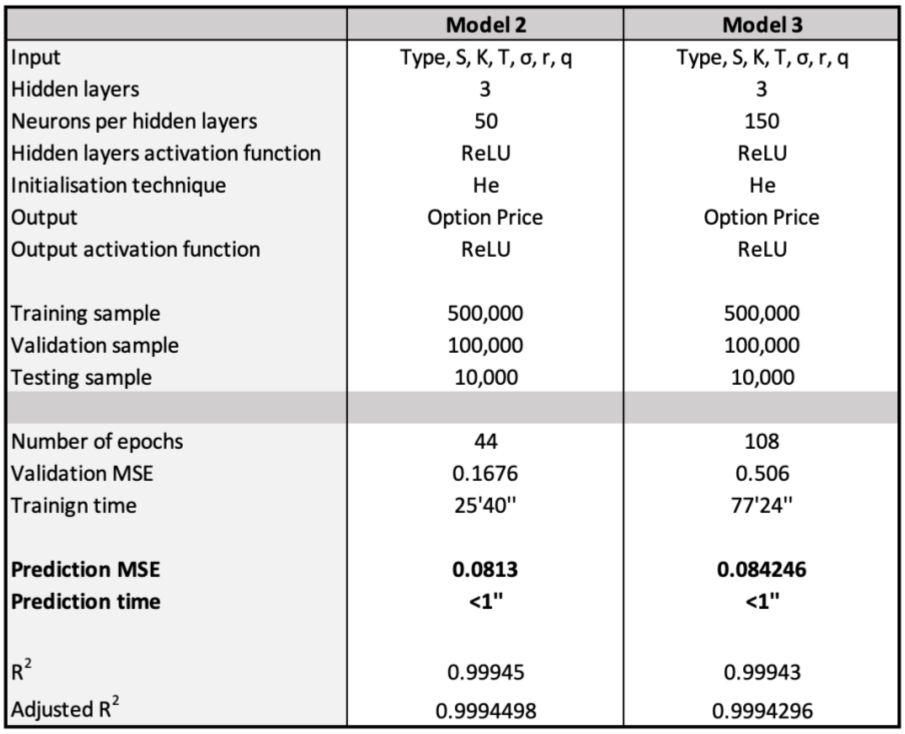
Here is a summary table of these first promising results:



Model 1	
Input	Type, S, K, T, σ, r, q
Hidden layers	3
Neurons per hidden layers	100
Hidden layers activation function	ReLU
Initialisation technique	He
Output	Option Price
Output activation function	ReLU
Training sample	500,000
Validation sample	100,000
Testing sample	10,000
Number of epochs	64
Validation MSE	0.0649
Trainign time	42'40''
Prediction MSE	**0.0715**
Prediction time	**<1''**
R^2	0.9994809
Adjusted R^2	0.99948055

Figure 17 - Results of the starting ANN structure

After this first test, I decided to test variates for the network structure. I started by varying the number of neurons per hidden layer. For model 2, I used 50 neurons, and for model 3 I used 150 neurons. This will allow me to see the impact, in this particular case, of increasing or decreasing the number of neurons, all other things remaining equal. In both cases, the results are good but a little less good than for 100 neurons per layer.

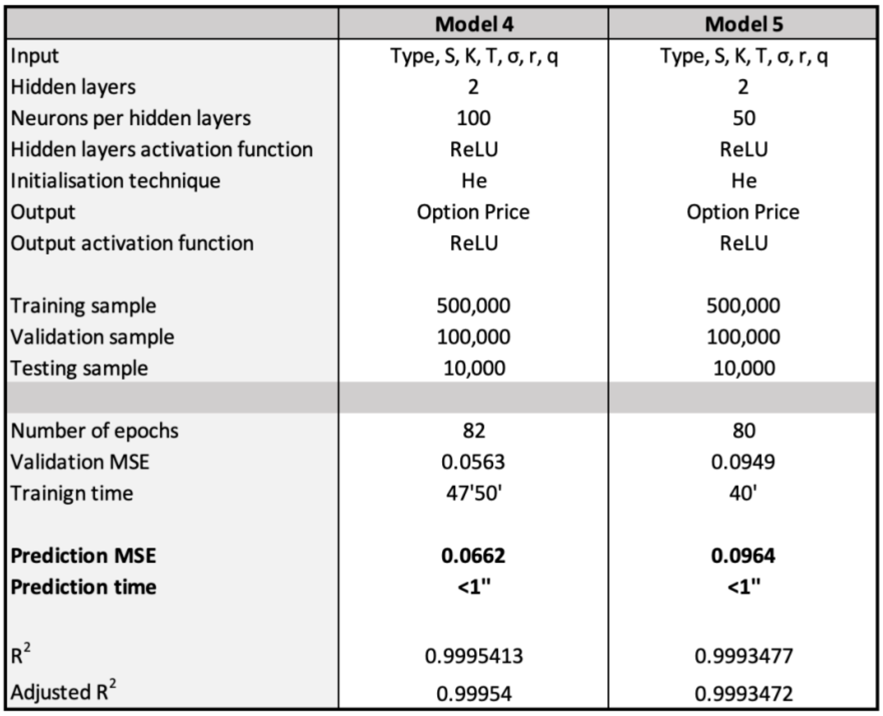


	Model 2	**Model 3**
Input	Type, S, K, T, σ, r, q	Type, S, K, T, σ, r, q
Hidden layers	3	3
Neurons per hidden layers	50	150
Hidden layers activation function	ReLU	ReLU
Initialisation technique	He	He
Output	Option Price	Option Price
Output activation function	ReLU	ReLU
Training sample	500,000	500,000
Validation sample	100,000	100,000
Testing sample	10,000	10,000
Number of epochs	44	108
Validation MSE	0.1676	0.506
Trainign time	25'40''	77'24''
Prediction MSE	**0.0813**	**0.084246**
Prediction time	**<1''**	**<1''**
R^2	0.99945	0.99943
Adjusted R^2	0.9994498	0.9994296

Figure 18 - ANN results : model 2 and 3

The MSE on the validation set varies very much between each epoch, so it is not very representative. I include it in the results to keep an idea of what is going on, in case we want to find something interesting. It can be seen that a model with more neurons requires more epochs to complete its training phase, and that the average time per epoch is longer.

Following these results, we can guess that adding a new hidden layer will not be positive for the network, since the model with fewer neurons (model 2) performs better than the model with more neurons (model 3), and since the model that gives the best results at the moment is in between (model 1). I still decided to explore smaller structures. Therefore, I choose to explore the reduction of the total number of neurons in the network, by first removing one hidden layer and keeping only two, each with 100 neurons. This will be model 4. For model 5, I choose to go further and use a neural network with two hidden layers of 50 neurons each.



	Model 4	Model 5
Input	Type, S, K, T, σ, r, q	Type, S, K, T, σ, r, q
Hidden layers	2	2
Neurons per hidden layers	100	50
Hidden layers activation function	ReLU	ReLU
Initialisation technique	He	He
Output	Option Price	Option Price
Output activation function	ReLU	ReLU
Training sample	500,000	500,000
Validation sample	100,000	100,000
Testing sample	10,000	10,000
Number of epochs	82	80
Validation MSE	0.0563	0.0949
Trainign time	47'50'	40'
Prediction MSE	**0.0662**	**0.0964**
Prediction time	**<1"**	**<1"**
R^2	0.9995413	0.9993477
Adjusted R^2	0.99954	0.9993472

Figure 19 - ANN results : model 4 and 5

Surprisingly, deleting one hidden layer gives better results than keeping 3 hidden layers with 50 neurons per layer, and also give better results than model 1 with three hidden layers of 100 neurons each. As for model 5, the performance is worse than all of the other models. It seems that a limit has been reached, and that a certain complexity in the network is still necessary to capture the relationships between input features and option prices.

After these attempts to modify the structure, I decided to stay with a neural network with 2 hidden layers and 100 neurons per layer and a neural network with 3 hidden layers and 100 neurons per layer, in order to compare the two once more. The next modification I would like to test is a modification of the activation function. Indeed, I will select here the LeakyReLU function, by assigning to the α parameter the value of 0.2, as advised in the work of Xu et al. (2015). Model 6 will be a 2-layer model with the LeakyReLU activation function and Model 7 will be the 3-layer model with the same activation function.

	Model 6	**Model 7**
Input	Type, S, K, T, σ, r, q	Type, S, K, T, σ, r, q
Hidden layers	**2**	**3**
Neurons per hidden layers	100	100
Hidden layers activation function	**LeakyReLU**	**LeakyReLU**
Initialisation technique	He	He
Output	Option Price	Option Price
Output activation function	ReLU	ReLU
Training sample	500,000	500,000
Validation sample	100,000	100,000
Testing sample	10,000	10,000
Number of epochs	76	89
Validation MSE	0.0961	0.07
Trainign time	45'10''	59'20''
Prediction MSE	**0.0961844**	**0.0875**
Prediction time	**<1''**	**<1''**
R^2	0.9993492	0.9994083
Adjusted R^2	0.9993487	0.9994079

Figure 20 - ANN results : model 6 and 7

Surprisingly, I find no improvement over the models with the ReLU function, but a slightly worse performance. I will now take the opportunity to point out something that does not often appear in papers on the subject, and which is one of the reasons why some researchers may find contradictory results: running the same neural network twice will not necessarily give the same results, and some conclusions may come within a specific framework, depending on the exact structure of the network, the features' characteristics or even the pre-processing of the data. I will develop this part in more depth in the next chapter. For now, let's continue testing the variations to see if we can get a major improvement in the performance of the neural network predictions. In these variations, we will test other activation functions of the ReLU family.

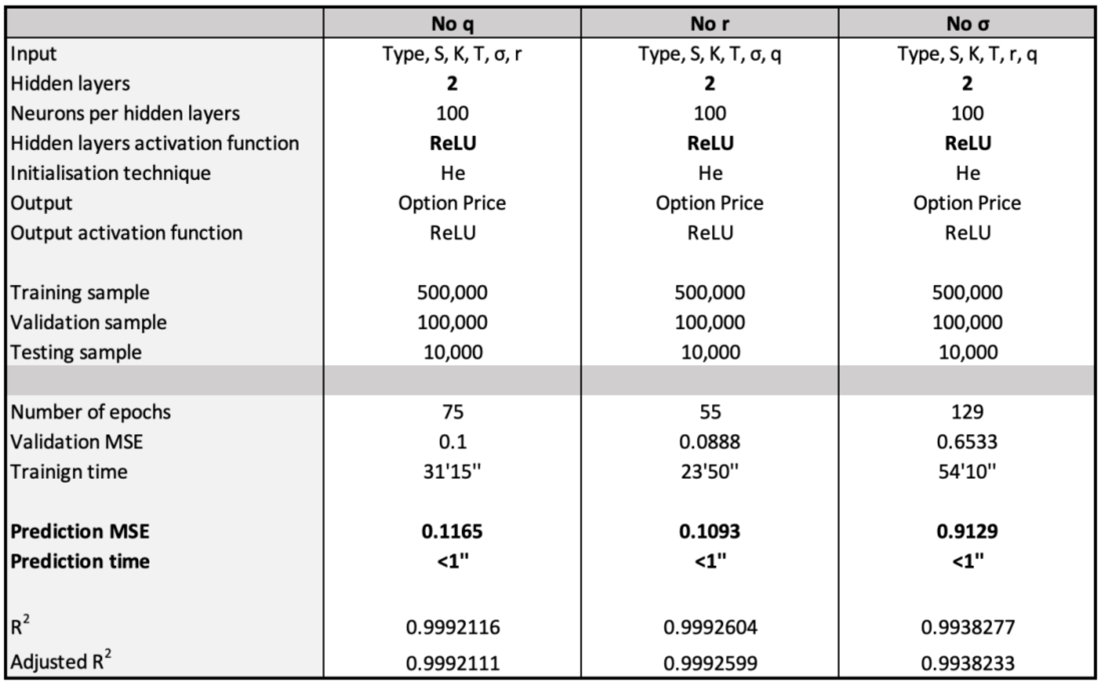
	Model 8	**Model 9**
Input	Type, S, K, T, σ, r, q	Type, S, K, T, σ, r, q
Hidden layers	2	**2**
Neurons per hidden layers	100	100
Hidden layers activation function	**PReLU**	**ELU**
Initialisation technique	He	He
Output	Option Price	Option Price
Output activation function	ReLU	ReLU
Training sample	500,000	500,000
Validation sample	100,000	100,000
Testing sample	10,000	10,000
Number of epochs	131	68
Validation MSE	0.0544	0.0827
Trainign time	87'20''	37'24''
Prediction MSE	**0.07153**	**0.09287**
Prediction time	**<1''**	**<1''**
R^2	0.999516	0.99937165
Adjusted R^2	0.99951565	0.99937123

Figure 21 - ANN results : model 8 and 9

From this table it can be seen that using PReLU or ELU as an activation function does not increase the performance of my network. I will therefore stick with the classic ReLU activation function.

Now that the structure variations have shown that the neural networks with 2 hidden layers and 100 neurons per layer are the ones that work best in my case, I wanted to be able to block this structure and tackle the feature variation.

Then, I decide to start from the optimal model (2 hidden layers of 100 neurons each) and to create 3 other models, the first where I remove the dividend yield, the second where I remove the interest rate and the third where I remove the implied volatility. This will allow me to see if these features had an impact or not in the current optimal model. Here is the table of results:



	No q	**No r**	**No σ**
Input	Type, S, K, T, σ, r	Type, S, K, T, σ, q	Type, S, K, T, r, q
Hidden layers	**2**	**2**	**2**
Neurons per hidden layers	100	100	100
Hidden layers activation function	**ReLU**	**ReLU**	**ReLU**
Initialisation technique	He	He	He
Output	Option Price	Option Price	Option Price
Output activation function	ReLU	ReLU	ReLU
Training sample	500,000	500,000	500,000
Validation sample	100,000	100,000	100,000
Testing sample	10,000	10,000	10,000
Number of epochs	75	55	129
Validation MSE	0.1	0.0888	0.6533
Trainign time	31'15''	23'50''	54'10''
Prediction MSE	**0.1165**	**0.1093**	**0.9129**
Prediction time	**<1''**	**<1''**	**<1''**
R^2	0.9992116	0.9992604	0.9938277
Adjusted R^2	0.9992111	0.9992599	0.9938233

Figure 22 - ANN results when removing one feature

It can be seen that all three models are less efficient than the current optimal model, the worst being the model without implied volatility. We see that the latter is very important. These results will be analysed in the following section.

Finally, I also tried adding volume and open interest, but none of these additions improved the results on the sample test compared to the optimal model. Adding the volume did not change anything and adding the open interest even degraded the network performance a bit. The result table can be found in *appendix 4*.

But what about replacing S and K by a single input, the moneyness, calculated as S/K? Well, in my case, I get strange results that I cannot explain. These go against what most researchers are doing today. Indeed, in my case, it leads me to an MSE on the test set of 84.669, which is simply colossal compared to previous results. I tried to investigate this result further but could not find the cause. Furthermore, the adjusted R^2 I get with this model is 0.426. This is the only value I got that is below 0.99. This frustrates me but I have not been able to find the cause of this problem. This is also something that is not obvious with neural networks, they are a bit like black boxes, it is difficult to always understand what is going on inside when a problem occurs. On this graph, we can see a scatter plot representing a pair between a value predicted by the neural network and the target value, with the red line representing the linear regression between these points. It is clear that there is a problem.

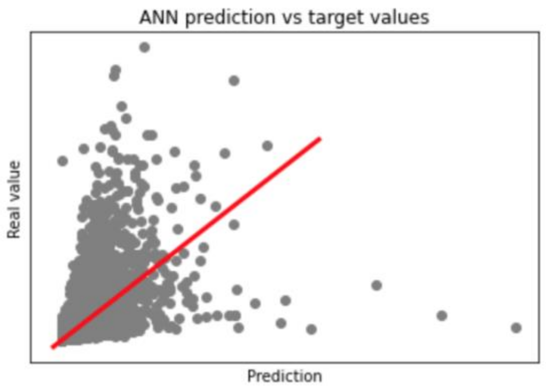


Figure 23 - Regression between prediction and target values

Anyway, now that we have all these results, we know that our optimal structure is almost the one we deduced at the beginning! It is the structure with 2 hidden layers with 100 neurons each, which uses a standardisation of the input data, which uses the ReLU activation function and the Adam optimiser, and which takes as input the type of option, the spot price, the strike price, the time to maturity, the implied volatility, the interest rate and the dividend yield in order to output the option price. It is therefore time to compare its results with those of the benchmarks, namely the Black Scholes and the Cox Ross Rubinstein binomial tree model (the python code implementation for these models can be found in *Appendix 5 and 6*).

	Black-Scholes	**CRR binomial tree**	**ANN**
Input	Type, S, K, T, σ, r, q	Type, S, K, T, σ, r, q, **steps**	Type, S, K, T, σ, r, q
Output	Option Price	Option Price	Option Price
Test sample size	10,000	10,000	10,000
Prediction MSE	**0.106661**	**0.103355**	**0.0662**
Prediction time	**9"**	**47"**	**<1"**
R^2	0.9992783	0.9993007	0.9995413
Adjusted R^2	0.9992778	0.9993002	0.99954

Figure 24 - Benchmarks performance on the ANN test sample compared to our model

And here is the result that confirms the hypothesis: the neural network outperforms the benchmarks. Several tests were performed on different portions of the available data, and the neural network consistently outperforms the benchmarks. Indeed, the neural network has smaller average errors, it has a higher R2 adjustment, which means that the model explains the output better, and it has a much lower prediction time, which can be really useful for practical application.

Between the benchmarks, we also see a logical difference: the Black-Scholes performs worse than the Cox Ross Rubinstein. Indeed, the former does not take into account the possibility of exercising the option before maturity while the latter does, so it is more adapted to American option pricing.

Discussion

Meanings of the findings

What these results mean is that it is possible to outperform classical models such as the Black-Scholes and especially the Cox Ross Rubinstein binomial tree model for pricing American options. Even better: it is possible to beat these benchmarks consistently to price Call and Put options whether they are in the money, at the money or out of the money with a single neural network! I had made the hypothesis that it was indeed advantageous to use such structures for derivatives pricing, both from an accuracy and speed point of view, but the results exceed my expectations.

As far as the theoretical framework is concerned, it can be seen that it was a good idea to make an initial qualitative analysis before embarking on solving the problem. Indeed, confronting the ideas and practices of different authors to draw a trend was more than beneficial, even if these trends were implemented in other contexts, such as European option pricing and implied volatility calculation. This allowed me to have enough information at hand to form an opinion and even informed guesses on certain parameters of the network.

For the second part of my framework, namely the quantitative part, the fact of trying several combinations of parameters and features allowed me to get a better idea of the impact of certain elements in the pricing problem. For example, I find that including the implied volatility in the network greatly increases its predictive power, which is in line with what I found in my literature review. When you think about it, it seems logical: an option whose underlying price varies very much will not be priced in the same way as the same option with an underlying price that varies little. The risks associated with these two options are different.

In the same way, I confirm that the interest rate has its place in the inputs. The fact of adding it increases the predictive power of the neural network on the option price. Indeed, when interest rates increase, the impact on a Call and Put option is not the same. It is beneficial for the Call option and penalising for the Put option. Therefore, it is useful to take into account this interest rate to take into account its effect on the value of the options. But this is not its single added value. I will come to it in two paragraphs.

Finally, I demonstrate by facts that the dividend yield is also an important feature to input into the network. Indeed, as soon as I remove it from the network, the latter loses some of its predictive power. However, one could say that when dividends fall, the price of the underlying is directly impacted, and that the simple spot price therefore allows the neural network to be given enough information relating to dividends. However, since we are dealing with American options, an interesting scenario may occur.

For this, I will have to introduce the "cost of carry". The cost of carry represents an intermediate cash flow that is the result of holding an asset. Dividends are a positive cash flow for the owner of the stock, so the cost of carry is positive for him. For the owner of the option on the underlying stock, the cost of carry is represented by the interest earned from holding cash minus the dividends that are paid to the owner of the underlying stock. Since an option has a certain extrinsic value that would be given up by exercising its right before the maturity of the contract, the choice to exercise depends on the difference between the extrinsic value of the option and the interest earned by having cash against the benefit of owning the underlying stock (cost of carry). Therefore, we can say that it is thanks to the interest rate and the dividend yield that the neural network can more or less take into account the impact of the cost of carry in the calculation of the option value. I was also able to demonstrate that adding the open interest and volume did not improve the performance of my network, as MSE and R2 are worse when taken into account. About the moneyness, I cannot draw a conclusion

Answer to the research question

Yes, it is possible to trade US stock options with neural networks. But there are advantages and disadvantages. These advantages are :

- Accuracy: indeed, the neural network has been able to outperform benchmarks such as the Black-Scholes model or the binomial tree model consistently and to a greater or lesser extent;
- The inclusion of parameters that are sometimes difficult to integrate: in a neural network it can sometimes be enough to add an input parameter to make a difference, whereas in a complex equation it can be difficult;
- The fact that a Put and a Call can be evaluated with the same algorithm;
- The fact that in the money, at the money or out of the money options can be evaluated efficiently with a single algorithm;
- The instantaneous calculation speed once the network has been trained.

However, there are also drawbacks that need to be noted:

- The learning phase which can be long depending on the type of neural network used and the structure given to it
- The fact that it is sometimes necessary to proceed by trial and error to see if the changes improve the network or not, which, combined with the sometimes low learning speed, can take a long time
- The fact that a lot of good quality data is needed to train the network, as too little data would make learning more complicated and overfitting more likely;
- The fact that a neural network is a black box that is difficult to analyse: as I noticed with my attempt to use moneyness as input, it is sometimes very complex to understand the reason for an error or an unexpected result.

This is in line with what we have found in the literature review, showing that we do not yet have a miracle solution, but that the results obtained from various sources are very encouraging for further research on the subject.

Obstacles and difficulties encountered

As far as obstacles and difficulties are concerned, I will say that the design itself is very complicated, because it is enough to make a small mistake or to change a single hyperparameter of the model for nothing to work. Machine learning is a field of research in which you always have to be very careful about what you do. On top of that, the fact that a neural network is like a black box that is difficult to inspect can be quite frustrating. Sometimes you just don't understand why you get this error or that result. This is one of the disadvantages of the advantage of machine learning: the machine learns, and we can't inspect everything.

Another difficulty I encountered was that the learning phase of a neural network can take a long time, especially if you do not have a very powerful computer. As my computer is more than 5 years old now, I had to face memory problems, crashes and interruptions of my code because the RAM was overloaded, and even programs that I had to leave running all night. Sometimes I even had to use cloud services to do some calculations. This is one of the reasons why I chose to train my network only on 500,000 realisations of my features and no more, because it was virtually impossible for me to do otherwise with the hardware at my disposal. Still, it was interesting to try to meet the challenge!

Conclusion

Executive summary

During this dissertation, I learned how to use neural networks and how to understand how they work so I could try to apply them to the problem of pricing derivatives. It seemed very complicated at first, but as I tried many different things and learned, I realised that it was possible to get satisfactory results.

I then proceeded to modify the structure and the inputs until I found a more or less optimal network, which I compared to my own implementations of the Black-Scholes and Cox Ross Rubinstein binomial tree model. This allowed me to start with a first conclusion: it is possible to predict American equity options at least as efficiently (if not more) with non-parametric neural network models.

Then, by digging a little deeper into the problem and by looking at my own newly acquired experience, I was able to establish a non-exhaustive list of the advantages and disadvantages of pricing with ANNs.

Managerial implications

The managerial consequences of my results are clear: managers involved in the pricing of derivatives such as American options or other more complex derivatives should be interested in machine learning techniques and neural networks. Indeed, they are able to learn the complex relationships between inputs and outputs in a way that can outperforms even classical models (used as benchmarks here). It can sometimes be more practical to “ask” a neural network to find these relationships than to create complex mathematical models from scratch.

On top of that, once the model is trained (which can take some time), the time needed to price the product is extremely low; pricing is almost instantaneous, which can be a huge advantage when you need to price a large number of products in real time.

I would therefore encourage managers to consider the benefits of this technique (as well as its drawbacks) objectively, so that they can decide whether or not to explore this area. But if I were to give advice purely on the basis of my findings, I would say that yes, it is worth looking into it.

Theoretical implications

In terms of the theoretical implications of my research, I was able to prove that the use of dividend yield in a pricing neural network was beneficial and I was able to confirm that the use of interest rate was useful.

I was able to verify other statements such as the fact that the spot and strike price are the most important elements in option pricing, along with the time to maturity.

I was also able to explore different activation functions and different optimisation algorithms, to confirm that the ReLU family of functions were indeed the most efficient (despite the fact that I had some counter-intuitive results with the Leaky ReLU and ELU functions) than for example the sigmoid function, or other functions that I tested during my experimental phases. I was able to confirm that the Adam optimizer was very efficient unlike others that had more trouble like the Stochastic Gradient Descent.

I was able to verify an assertion made in the literature review that beyond 3 hidden layers, the neural network lost performance for a pricing exercise.

Where a lot of research separated Put and Call options in different networks, and even in the money, out of the money and at the money options, I was able to prove that a single network could learn enough to distinguish between them and deal with all kinds of options at once.

In the end, it even goes beyond what I hoped to prove at the beginning of my research.

Limitations and suggestions for future research

My methodology faces several limitations:

- The optimal structure found for my neural network may depend heavily on the data I input to it. If I had had other implementations or data from another generic provider, the results might have been different.
- I only used fairly simple neural networks. More complex neural networks such as Recurrent Neural Networks (RNN), which would seem to be particularly suited to the problem of pricing a path dependent financial product, or Long Short Term Memory (LSTM) algorithms which keep certain data in memory longer could be more efficient. Indeed, these more complicated networks, often referred to as deep ANNs, require much more effort and knowledge.
- The choice of benchmarks could be wrong in the sense that some approximation techniques much more complex and efficient than Black-Scholes and CRR already exist. It is possible that these outperform my neural network.
- The way I selected my data over time: although I tested several possibilities and the neural network was always the best performing, I did not test everything. For example, I didn't look at whether the neural network was still performing better than the benchmarks in times of economic crisis.
- Not separating ITM ATM and OTM options is an advantage in one sense as it allowed me to prove that we could afford to take everything together, but it is perhaps also a disadvantage in the sense that several separate networks adapted to each type of option might be more efficient.

My recommendations for future research would be to continue in the same vein, taking these limitations into account and testing, for example, the capacity of the networks in times of crisis, testing the use of different networks for each type of option or trying to use RNN or LSTM type networks.

Bibliography

Amornwattana, S., Enke, D., & Dagli, C. H. (2007). A hybrid option pricing model using a neural network for estimating volatility. *International Journal of General Systems*, *36*(5), 558–573. https://doi.org/10.1080/03081070701210303

Anders, U., Korn, O., & Schmitt, C. (1998). Improving the pricing of options: a neural network approach. *Journal of Forecasting*, *17*(5–6), 369–388.

Andreou, P. C. (2008). Parametric and Nonparametric Functional Estimation for Options Pricing with Applications in Hedging and Trading. *PhD Thesis, University of Cyprus*.

BARONE-ADESI, G., & WHALEY, R. E. (1987). Efficient Analytic Approximation of American Option Values. *The Journal of Finance*, *42*(2), 301–320. https://doi.org/10.1111/j.1540-6261.1987.tb02569.x

Becker, S., Cheridito, P., & Jentzen, A. (2019). Deep optimal stopping. *Journal of Machine Learning Research*, *20*, 74.

Bennell, J., & Sutcliffe, C. (2004). Black–Scholes versus artificial neural networks in pricing FTSE 100 options. *Intelligent Systems in Accounting, Finance and Management*, *12*(4), 243-260.

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, *81*(3), 637–654. https://doi.org/10.1086/260062

Blynski, L., & Faseruk, A. (2006). Comparison of the effectiveness of option price forecasting: Black-Scholes vs. simple and hybrid neural networks. *Journal of Financial Management & Analysis*, *19*(2), 46.

Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *ArXiv Preprint ArXiv:1511.07289.*

Cox, J. C., Ross, S. A., & Rubinstein, M. (1979). Option pricing: A simplified approach. *Journal of Financial Economics*, *7*(3), 229–263. https://doi.org/10.1016/0304-405x(79)90015-1

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7).

ÉLève, E. G. (2021, December 12). *La loi de Moore au futur? - Etienne Gagnon élève*. Medium. https://medium.com/@gageti31/la-loi-de-moore-au-futur-f5fe09db40d1

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. https://doi.org/10.1007/bf00344251

Garcia, R., & Gençay, R. (2000). Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics*, *94*(1–2), 93–115. https://doi.org/10.1016/s0304-4076(99)00018-4

Geigle DS, Aronson JE. 1999. An artificial neural network approach to the valuation of options and forecasting of volatility. *Journal of Computational Intelligence in Finance 7*(6): 19–25.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O'Reilly Media.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics.*, 249–256.

Goncalves, S., & Guidolin, M. (2006). Predictable dynamics in the S&P 500 index options implied volatility surface. *The Journal of Business*, *79*(3), 1591-1635.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *In Proceedings of the IEEE International Conference on Computer Vision.*, 1026–1034.

Hirsa, A., Karatas, T., & Oskoui, A. (2019). Supervised deep neural networks (DNNs) for pricing/calibration of vanilla/exotic options under various different processes. *ArXiv Preprint ArXiv:1902.05810.*

Hopfield, J. J., Feinstein, D. I., & Palmer, R. G. (1983). 'Unlearning' has a stabilizing effect in collective memories. *Nature*, *304*(5922), 158–159. https://doi.org/10.1038/304158a0

Hull, J. (2017). *Options, Futures, and Other Derivatives* (10th ed.). Pearson.

HUTCHINSON, J. M., LO, A. W., & POGGIO, T. (1994). A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks. *The Journal of Finance*, *49*(3), 851–889. https://doi.org/10.1111/j.1540-6261.1994.tb00081.x

Kelly, D. L., & Shorish, J. (1994). Valuing and hedging American put options using neural networks. *Unpublished Manuscript, Carnegie Mellon University.*

Khashei, M., & Bijari, M. (2010). An artificial neural network (p, d, q) model for timeseries forecasting. *Expert Systems with applications*, *37*(1), 479-489.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980.*

Kinsley, H., & Kukieła, D. (2020). *Neural Networks from Scratch in Python* (1st ed.). nnfs.io.

Kohler, M., Krzyżak, A., & Todorovic, N. (2010). PRICING OF HIGH-DIMENSIONAL AMERICAN OPTIONS BY NEURAL NETWORKS. *Mathematical Finance*, *20*(3), 383–410. https://doi.org/10.1111/j.1467-9965.2010.00404.x

Lajbcygier, P. R., & Connor, J. T. (1997). Improved Option Pricing Using Artificial Neural Networks and Bootstrap Methods. *International Journal of Neural Systems*, *08*(04), 457–471. https://doi.org/10.1142/s0129065797000446

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, *16*(2), 146–160. https://doi.org/10.1007/bf01931367

Malliaris, M., & Salchenberger, L. (1993a). A neural network model for estimating option prices. *Applied Intelligence*, *3*(3), 193–206. https://doi.org/10.1007/bf00871937

Malliaris, M., & Salchenberger, L. (1993b). Beating the best: A neural network challenges the Black-Scholes formula. *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*, 445–449.

Minsky, M., & Papert, S. A. (1969). *Perceptrons: An Introduction to Computational Geometry* (1st ed.). The M.I.T. Press.

Montesdeoca, L., & Niranjan, M. (2016). Extending the feature set of a data-driven artificial neural network model of pricing financial options. *In 2016 IEEE Symposium Series on Computational Intelligence (SSCI).*, 1–6.

Palmer, S. (2019). Evolutionary algorithms and computational methods for derivatives pricing. *Doctoral Dissertation, UCL (University College London).*

Pires, M. M., & Marwala, T. (2005). American option pricing using Bayesian multi-layer perceptrons and Bayesian support vector machines. *In IEEE 3rd International Conference on Computational Cybernetics, 2005.*, 219–224.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1–17. https://doi.org/10.1016/0041-5553(64)90137-5

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Shalf, J. (2020). The future of computing beyond Moore's Law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *378*(2166), 20190061. https://doi.org/10.1098/rsta.2019.0061

Statista. (2021, March 15). *Number of futures and options contracts traded globally 2013–2020*. https://www.statista.com/statistics/377025/global-futures-and-options-volume/#:%7E:text=Number%20of%20futures%20and%20options%20contracts%20traded%20globally%202013%2D2020&text=In%202020%2C%2025.55%20billion%20futures,contracts%20in%20the%20same%20period.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *ArXiv Preprint ArXiv:1505.00853.*

Zapart, C. A. (2003). Beyond Black–Scholes: A Neural Networks-Based Approach to Options Pricing. *International Journal of Theoretical and Applied Finance*, *06*(05), 469–489. https://doi.org/10.1142/s0219024903002006

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, *50*, 159-175.

Zheng, Y., Yang, Y., & Chen, B. (2019). Gated deep neural networks for implied volatility surfaces. *ArXiv Preprint*.

Appendix

Appendix 1 – Histograms of the input features

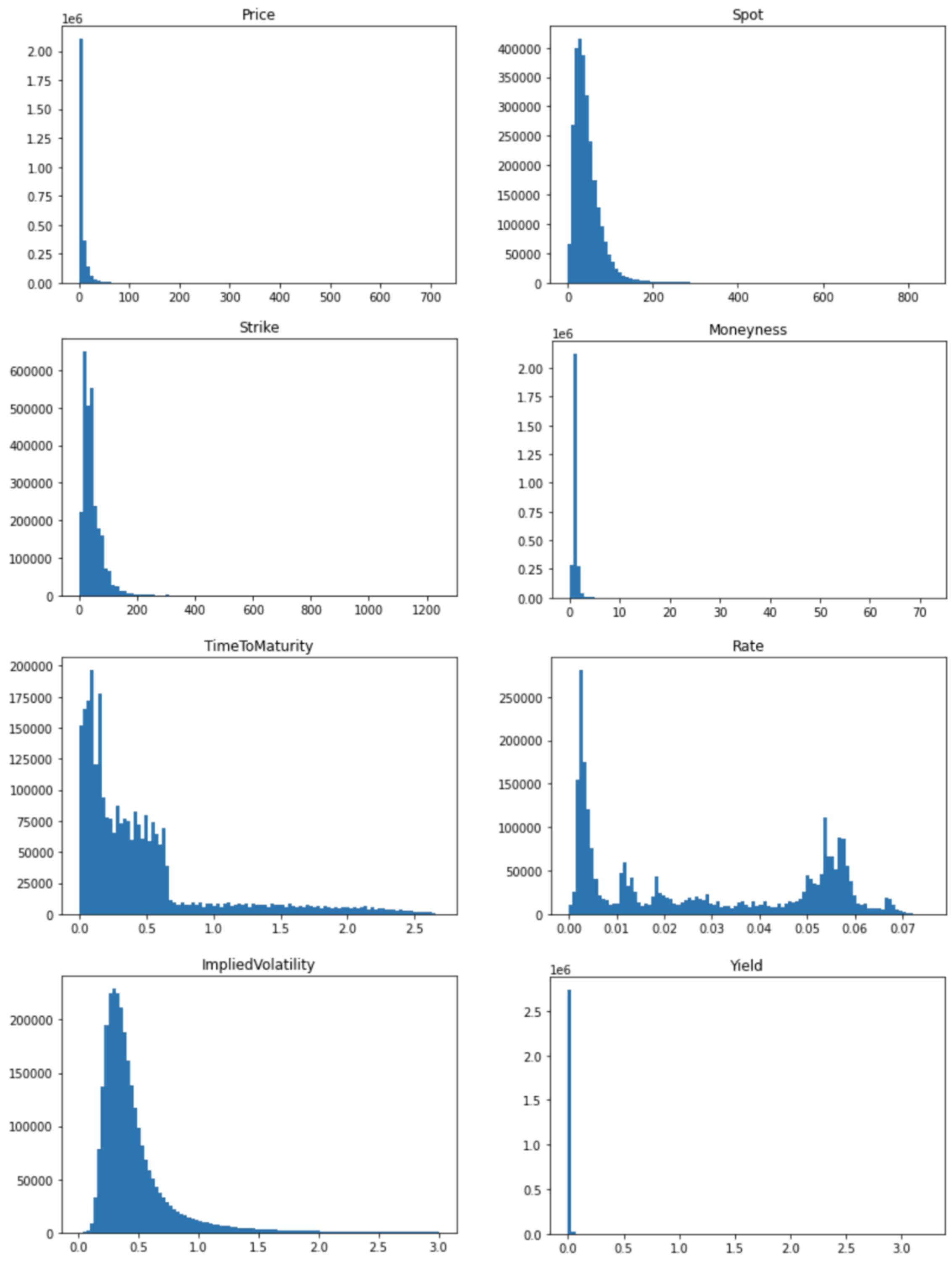


Figure 25 - Histograms of the input features

Appendix 2 – Box plots of the input features

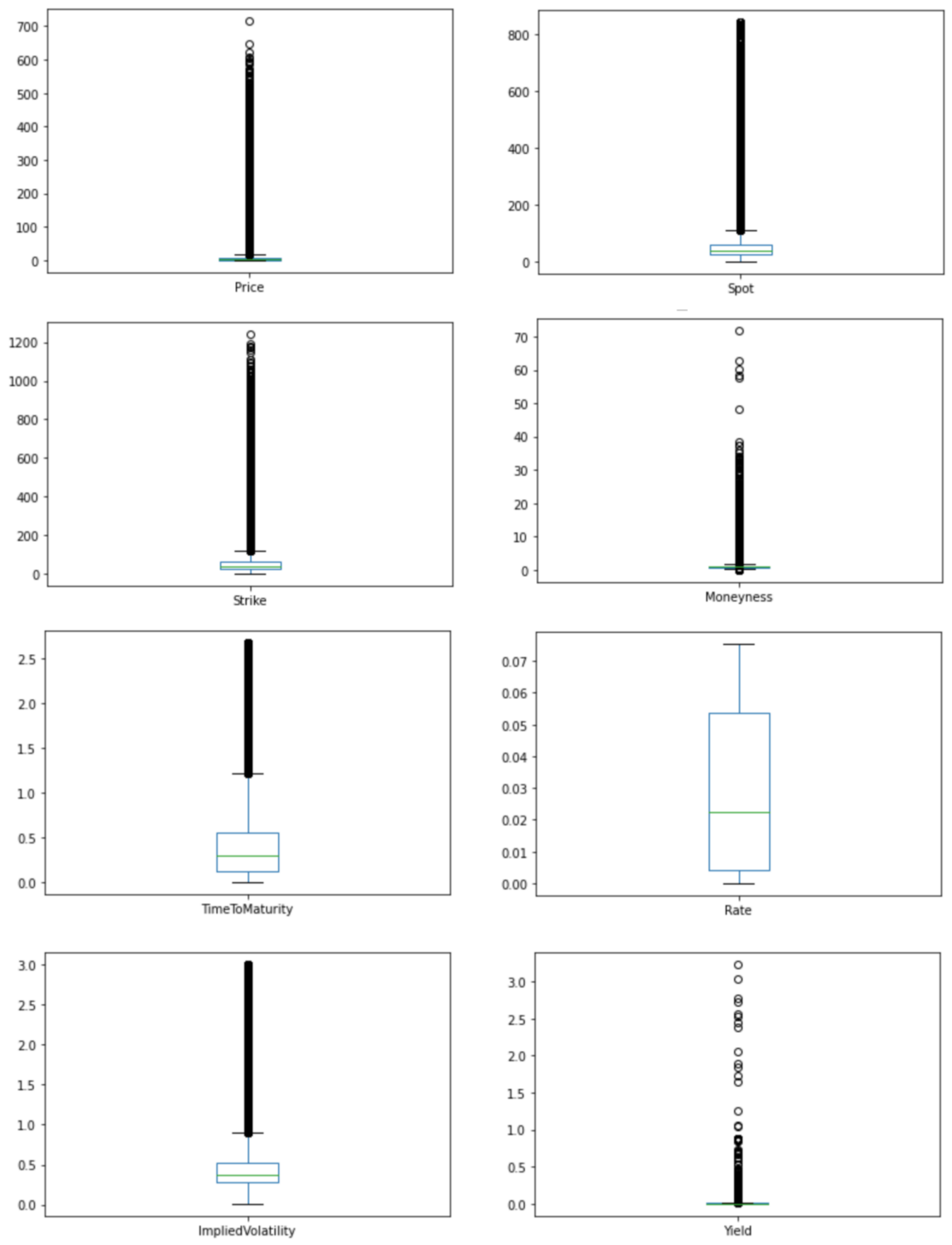
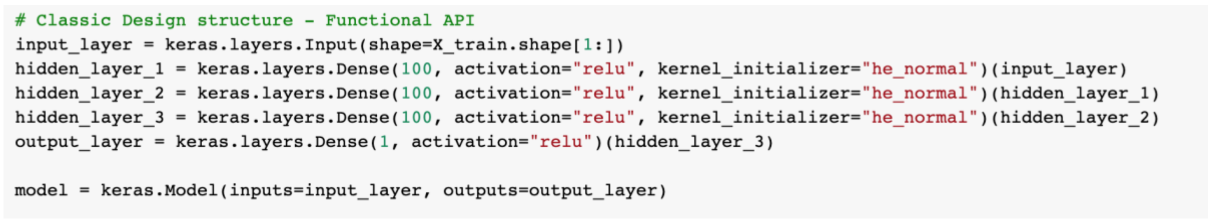


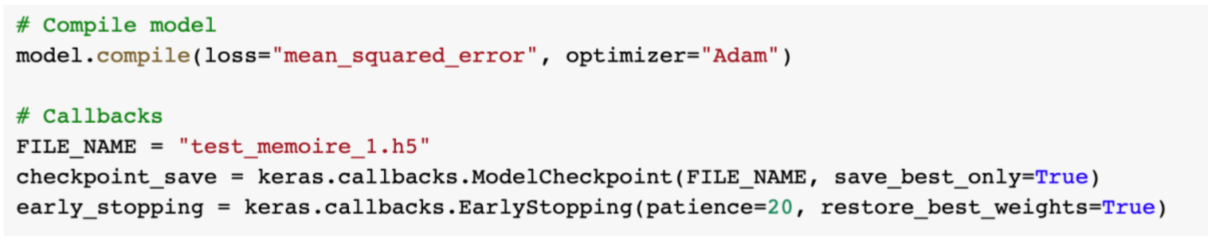
Figure 26 - Box plots of the input features

Appendix 3 – code snippet : ANN with Functional API



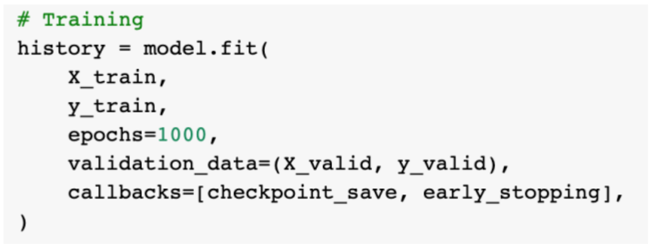
```
# Classic Design structure - Functional API
input_layer = keras.layers.Input(shape=X_train.shape[1:])
hidden_layer_1 = keras.layers.Dense(100, activation="relu", kernel_initializer="he_normal")(input_layer)
hidden_layer_2 = keras.layers.Dense(100, activation="relu", kernel_initializer="he_normal")(hidden_layer_1)
hidden_layer_3 = keras.layers.Dense(100, activation="relu", kernel_initializer="he_normal")(hidden_layer_2)
output_layer = keras.layers.Dense(1, activation="relu")(hidden_layer_3)

model = keras.Model(inputs=input_layer, outputs=output_layer)
```

Figure 27 - ANN structure design



```
# Compile model
model.compile(loss="mean_squared_error", optimizer="Adam")

# Callbacks
FILE_NAME = "test_memoire_1.h5"
checkpoint_save = keras.callbacks.ModelCheckpoint(FILE_NAME, save_best_only=True)
early_stopping = keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True)
```

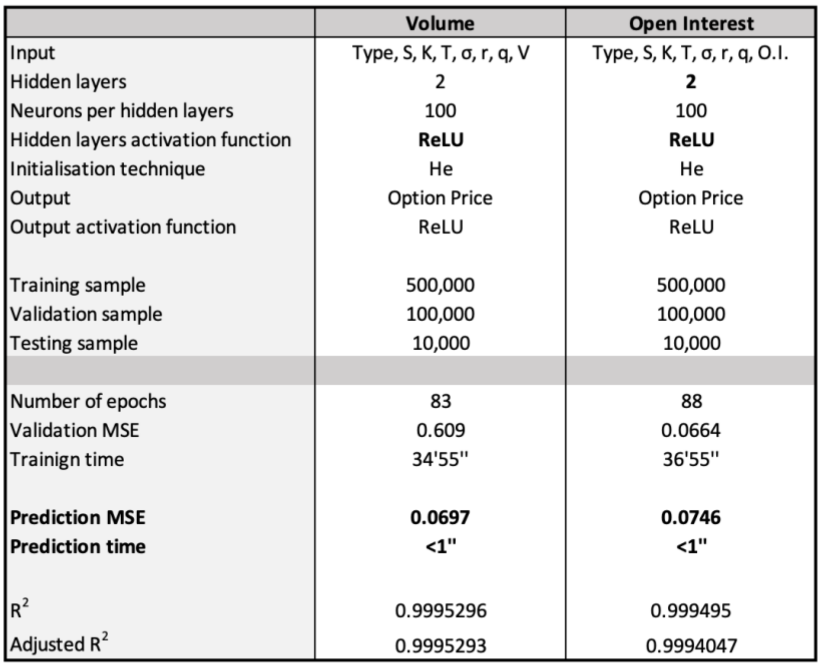
Figure 28 - model compilation and call-backs definition



```
# Training
history = model.fit(
    X_train,
    y_train,
    epochs=1000,
    validation_data=(X_valid, y_valid),
    callbacks=[checkpoint_save, early_stopping],
)
```

Figure 29 - learning phase

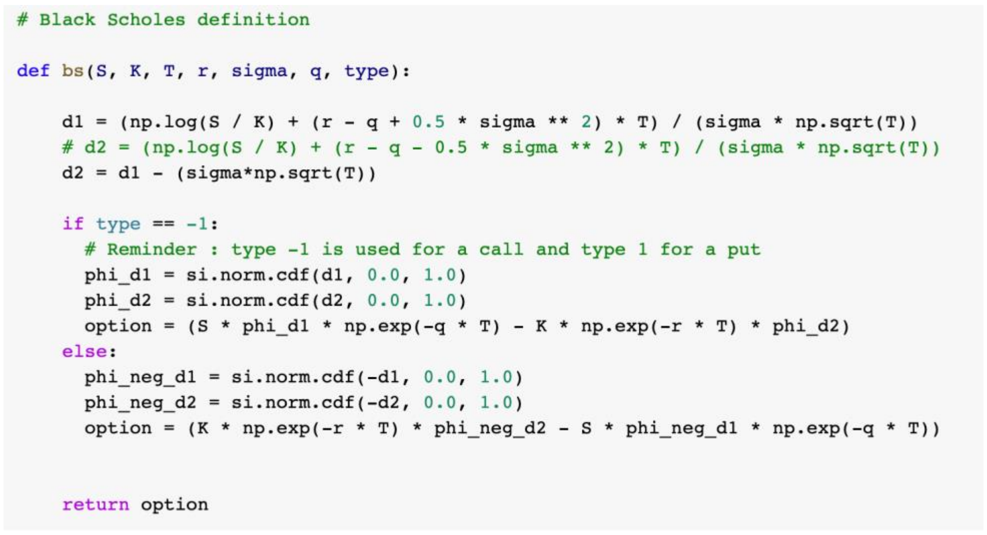
Appendix 4 – ANN result table for volume and open interest in the optimal structure



	Volume	Open Interest
Input	Type, S, K, T, σ, r, q, V	Type, S, K, T, σ, r, q, O.I.
Hidden layers	2	**2**
Neurons per hidden layers	100	100
Hidden layers activation function	**ReLU**	**ReLU**
Initialisation technique	He	He
Output	Option Price	Option Price
Output activation function	ReLU	ReLU
Training sample	500,000	500,000
Validation sample	100,000	100,000
Testing sample	10,000	10,000
Number of epochs	83	88
Validation MSE	0.609	0.0664
Trainign time	34'55''	36'55''
Prediction MSE	**0.0697**	**0.0746**
Prediction time	**<1"**	**<1"**
R^2	0.9995296	0.999495
Adjusted R^2	0.9995293	0.9994047

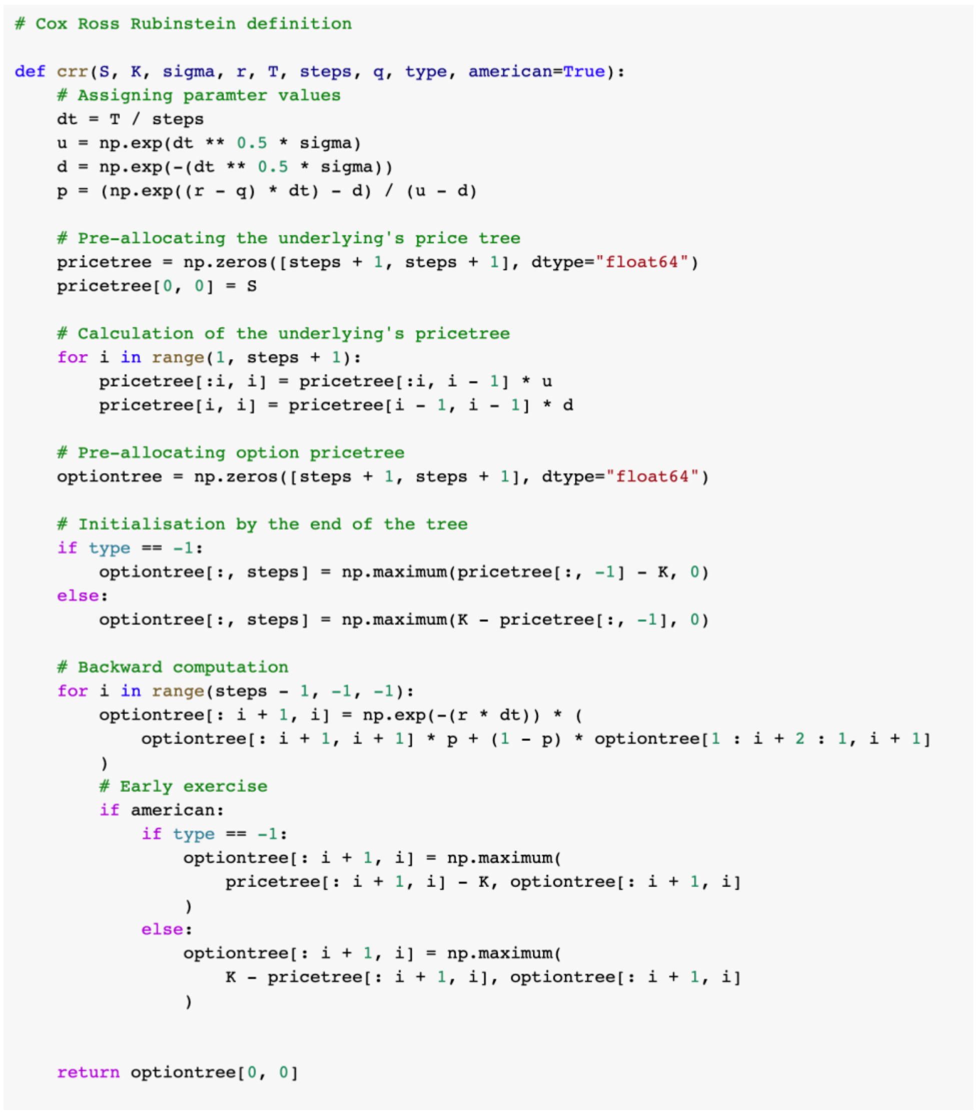
Figure 30 - ANN with Volume and OpenInterest as extra input feature

Appendix 5 – code snippet : Black Scholes model definition



```python
# Black Scholes definition

def bs(S, K, T, r, sigma, q, type):

    d1 = (np.log(S / K) + (r - q + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    # d2 = (np.log(S / K) + (r - q - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - (sigma*np.sqrt(T))

    if type == -1:
      # Reminder : type -1 is used for a call and type 1 for a put
      phi_d1 = si.norm.cdf(d1, 0.0, 1.0)
      phi_d2 = si.norm.cdf(d2, 0.0, 1.0)
      option = (S * phi_d1 * np.exp(-q * T) - K * np.exp(-r * T) * phi_d2)
    else:
      phi_neg_d1 = si.norm.cdf(-d1, 0.0, 1.0)
      phi_neg_d2 = si.norm.cdf(-d2, 0.0, 1.0)
      option = (K * np.exp(-r * T) * phi_neg_d2 - S * phi_neg_d1 * np.exp(-q * T))


    return option
```

Appendix 6 – code snippet : CRR binomial tree model definition



```python
# Cox Ross Rubinstein definition

def crr(S, K, sigma, r, T, steps, q, type, american=True):
    # Assigning paramter values
    dt = T / steps
    u = np.exp(dt ** 0.5 * sigma)
    d = np.exp(-(dt ** 0.5 * sigma))
    p = (np.exp((r - q) * dt) - d) / (u - d)

    # Pre-allocating the underlying's price tree
    pricetree = np.zeros([steps + 1, steps + 1], dtype="float64")
    pricetree[0, 0] = S

    # Calculation of the underlying's pricetree
    for i in range(1, steps + 1):
        pricetree[:i, i] = pricetree[:i, i - 1] * u
        pricetree[i, i] = pricetree[i - 1, i - 1] * d

    # Pre-allocating option pricetree
    optiontree = np.zeros([steps + 1, steps + 1], dtype="float64")

    # Initialisation by the end of the tree
    if type == -1:
        optiontree[:, steps] = np.maximum(pricetree[:, -1] - K, 0)
    else:
        optiontree[:, steps] = np.maximum(K - pricetree[:, -1], 0)

    # Backward computation
    for i in range(steps - 1, -1, -1):
        optiontree[: i + 1, i] = np.exp(-(r * dt)) * (
            optiontree[: i + 1, i + 1] * p + (1 - p) * optiontree[1 : i + 2 : 1, i + 1]
        )
        # Early exercise
        if american:
            if type == -1:
                optiontree[: i + 1, i] = np.maximum(
                    pricetree[: i + 1, i] - K, optiontree[: i + 1, i]
                )
            else:
                optiontree[: i + 1, i] = np.maximum(
                    K - pricetree[: i + 1, i], optiontree[: i + 1, i]
                )


    return optiontree[0, 0]
```