LUISS

Dipartimento di Economia e Finanza
Percorso Banche ed Intermediari Finanziari

Option pricing
A comparison between Black-Scholes model and a Deep Learning approach

NICOLA BORRI

RELATORE

PIERPAOLO BENIGNO

CO-RELATORE

GIAMPAOLO TUMMINELLO

CANDIDATO

ANNO ACCADEMICO 2018/2019

Un sentito ringraziamento al Professor Borri,
per avermi permesso di realizzare un elaborato
i cui temi sento profondamente vicini.

Ai miei genitori, che mi hanno incondizionatamente
sostenuto ed aiutato in ogni mia scelta,
rendendomi la persona che sono oggi.

A Lisa, per la sua immensa pazienza e
per la forza e l'amore che mi ha trasmesso
lungo tutto il mio percorso universitario.

A John von Neumann, fonte di ispirazione e motivazione.

Contents

Chapter 1

Introduction

A system can be considered a group of variable interconnected interacting each other in a defined environment[1]. Building a model means attempting to describe a system using abstract concepts and logical language. One example of this concept is the financial market, an economic system in which financial securities and derivatives are traded. There are many mathematical models trying to describe and abstracting how the financial market and all its components work. The history of modern mathematical finance can be brought back to the work of Louis Bachelier (1900), *Théorie de la speculation*[2], in which is introduced the stochastic process called Brownian motion and its use for stock options pricing. Later, in 1973, starting from the assumption that stock prices follow a geometric Brownian motion, Fischer Black and Myron Scholes published their famous and influential paper *The Pricing of Options and Corporate Liabilities*[3], where a closed formula for European option pricing is given, solving a parabolic partial differential equation, with initial conditions based on continuous "Delta-hedging". After, many authors tried to improve the Black-Scholes model, by relaxing the assumption on volatility (Scott, 1987) (Heston, 1993) , or using other methods such as Monte Carlo simulation (Boyle, 1977). Nevertheless, between all mathematical (parametric) models for option pricing, the Black-Scholes model is still the most used in practice, due to its ease of implementation and simplicity of interpretation. With the informatic revolution in the new millennium and the rise of technological giants, the advent of Big Data changed the way data was analysed and how models were built. After a period, called the "AI winter", with fresh projects such as IBM Watson (High, 2012) , AlphaGo (Silver et al., 2016), and OpenAI Five (Berner et al., 2019), a new series of models, mostly derived from statistics and linear algebra, started to be used in practice in many fields. For instance, in the financial area, they're currently used for cluster analysis (Șchiopu, 2010), portfolio management (Liu et al., 2011), credit risk evaluation (Angelini et al., 2008), and asset pricing. As concerns pricing, a vast amount of literature has compared mathematical models with machine learning and deep learning methods for derivates pricing. Hutchinson et al. (1994)[4] proposed a non-parametric approach, using artificial neural networks for pricing and hedging derivatives, trying to replicate the Black-Scholes formula, but without the same restrictions in hypothesis. Other studies used deep learning specifically to relax the hypothesis of Black-Scholes model about volatility and showed that a semi-parametric model can outperform Black-Scholes model when an option is strongly in the money or strongly out of the money (Baruníkova & Baruník, 2011).

Although, none of the literature cited used the same variables when comparing mathematical models against artificial neural networks or machine learning models. This thesis aims to compare a simple feed forward

[1] https://www.merriam-webster.com/dictionary/system
[2] Bachelier, L. (1900a), Théorie de la spéculation, Annales Scientifiques de l'École Normale Supérieure, 3.
[3] Black, F.; Scholes, M. (1973), The Pricing of Options and Corporate Liabilities, Journal of Political Economy, 81.
[4] Hutchinson, J. M., A. W. Lo, & T. Poggio (1994), A nonparametric approach to pricing and hedging derivative securities via learning networks, The Journal of Finance 49 (3)

neural network with the Black-Scholes model, using for both models the same variables:

$\boldsymbol{S}$: the underlying price;

$\boldsymbol{K}$: the strike price of the option;

$\boldsymbol{T}$: the time to maturity;

$\boldsymbol{\sigma}$: the volatility of the option;

R : the risk free interest rate[5].

The next chapter will review some of the most important literature, useful for the contextualization of the thesis. The third chapter is a review of the main topics and hypothesis concerning neural networks and mathematical models for option pricing. In the second part, the study is explained, giving attention on the structure of the dataset used, the implementation of the two models, the results and the critical issues found.

[5] Dividends expected or already payed are not considered both for simplicity of treatment and lack of specific data.

Chapter 2

Literature review

Artificial neural network literature

Warren McCulloch and Walter Pitts (1943)[6] showed that simple neural network has the ability to approximate any arithmetic function. This concept is the base on which the later literature on artificial neural networks was built. The first attempt to emulate a biological neuron was the Perceptron, proposed by Frank Rosenblatt (1958)[7]. It is a weighted sum of the input variables, filtered by a non-linear function.

$$Output = f(\sum_{i=1}^{n} x_i w_i)$$

Where n is the number of inputs, x_i are the input values and w_i are the relative weights.

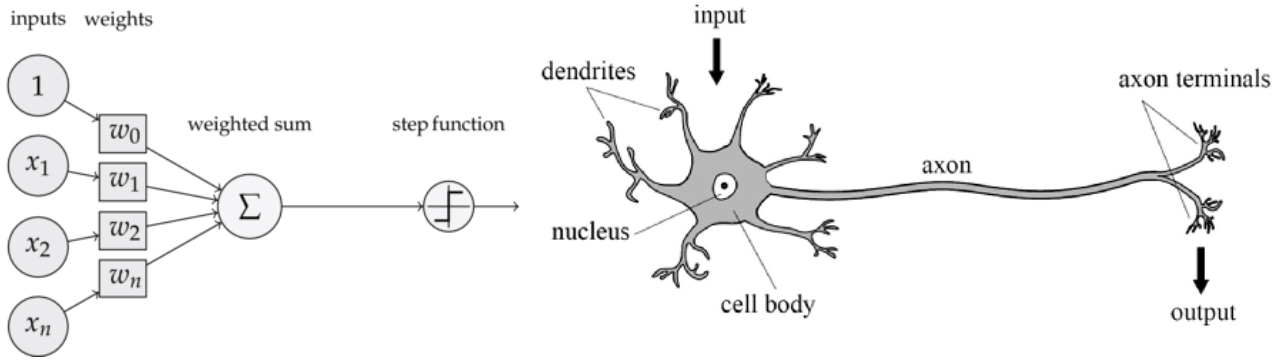

Figure 1: *a schematic representation of a perceptron, compared to a biological neuron.*
Source: Stack Overflow for the image on the left, Research Gate for the image on the right.

The problem with this model was its inability to learn from data, it was a static model without changing in its weights. This led to the so called "AI winter", during which the interest of the scientific community for artificial intelligence declined. The situation changed with the invention of the backpropagation algorithm (Kelley, 1960) (Rumelhart, , Hinton, & Ronald, 1986) and the multiple layers perceptron (Hinton, 2006). These two topics will be discussed in the next chapter.

[6] Warren S. McCullochWalter Pitts (1943), A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics Vol. 5, 115-133
[7] Frank F. Rosenblatt (1958), The perceptron: a probabilistic model for information storage and organization in the brain, Psychological review

Option pricing literature

As discussed before, the most influential paper can be considered *The Pricing of Options and Corporate Liabilities* by Fisher Black and Myron Scholes (1973), later expanded by Robert Merton in his *Theory of Rational Option Pricing*[8]. Their work legitimized and contributed to the spread of the Chicago Board Options Exchange, established in 1973, which uses standardized contracts guaranteed by a clearing house. All the variables in the Black-Scholes model are unequivocally observable, except the volatility, which is non-constant and must be deduced with various methods (that will be discussed later). Heston (1993)[9] tried to avoid this problem by modelling the volatility with a stochastic process based on a Brownian motion. Other methods use Fast Fourier transform (Carr & Madan, 1999) or Jump diffusion processes (Kou, 2002).
Another method used for option pricing is the Binomial model, firstly proposed by William Sharpe in 1978[10], later formalized by J. Cox, S. Ross and M. Rubinstein (1979)[11]. This model considers the underlying asset price as a discrete-time random walk, that can go up or down by a fixed amount every time with a fixed probability.

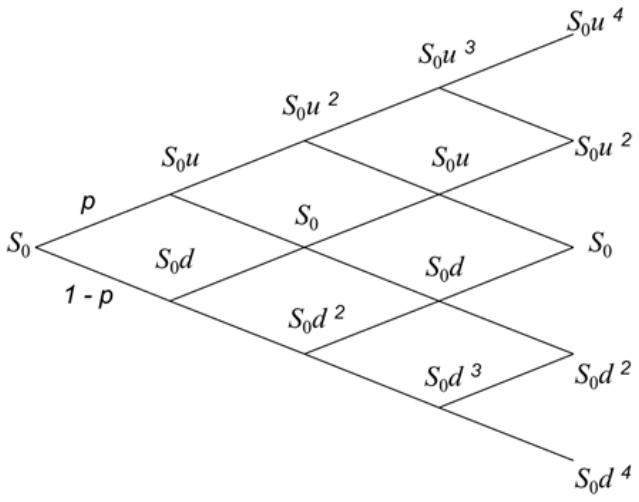

Figure 2: *representation of the binomial "tree" that schematize the movement of the asset price. Source: Finance Train*

In the model the price P of the option is given by:

$$P = \frac{1}{r^n}\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}\max(0, u^i d^{n-i}S - K)$$

Where r is the single period interest rate, p is the risk neutral probability, u and d are the up and down factors, S is the underlying asset price at time 0, K is the option's strike price and n is the number of periods

[8] Merton, Robert (1973), Theory of Rational Option Pricing, Bell Journal of Economics and Management Science. 4 (1): 141–183
[9] Heston, S. L. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The Review of Financial Studies, 6 (2).
[10] Sharpe, William (1978), Investments, Englewood Cliffs, N.J., Prentice-Hall
[11] Cox, J. C.; Ross, S. A.; Rubinstein, M. (1979), Option pricing: A simplified approach, Journal of Financial Economics 7 (3): 229

considered. It can be demonstrated that, under similar assumptions, the binomial distribution of the binomial model converges to the lognormal distribution assumed by Black-Scholes model, thus, in this perspective, the binomial model can be considered a discrete time approximation of the continuous process in the Black-Scholes model (Leisen & Reimer, 2006) (Chance, 2008). Technical features of European options and Black-Scholes model are discussed in the next chapter.

Chapter 3

Theoretical Part

Option fundamentals

A European option is a contract which gives the owner (the holder or the buyer) the right, but not the duty, to buy (or sell) an underlying asset at a specified price (strike price) and at specified date (expiration date, exercise date or maturity). The other part involved in the contract (the seller) has the obligation to sell (or buy) the underlying asset if the buyer exercises her right. Buyers are referred to as having long positions, sellers as having short positions. In the case the holder has the right to buy the underlying instrument, the option is referred to as Call option, in the opposite case it's called Put option. The two counterparties must specify in the contract:

- the type of option: Call or Put;
- the quantity of the underlying asset;
- the strike price (K);
- the expiration date (T).

The price of the underlying asset (S_0) is not considered at the time of the stipulation of the contract (except for pricing purposes), because the underlying price matters only at the settlement date (or expiration date), when the option can be exercised.

There are six factors affecting the price of a stock option:

- the current stock price S_0;
- the strike price, K;
- the time to maturity, T;
- the volatility of the stock price, σ;
- the risk-free interest rate, r;
- the dividends that are expected to be paid[12].

Variable	*European call*	*European put*
Current stock price	+	−
Strike price	−	+
Time to expiration	?	?
Volatility	+	+
Risk-free rate	+	−
Amount of future dividends	−	+

Figure 3: *how the six factors cited above affects the price of European options.*
Source: John C. Hull (2011), Options, Futures and Other Derivatives

[12] Jhon C. Hull (2011), Options, Futures and Other Derivatives: Global 8th edition, Financial Times Prentice Hall

Considering a call option on a non-dividend-paying stock, its payoff, c, is determined as follow:

$$c \geq \max\left(S_0 - Ke^{-rT}, 0\right)$$

Which implies that if the continuously compounded discounted strike price is equal or greater than the current underlying price, the option is not exercised.

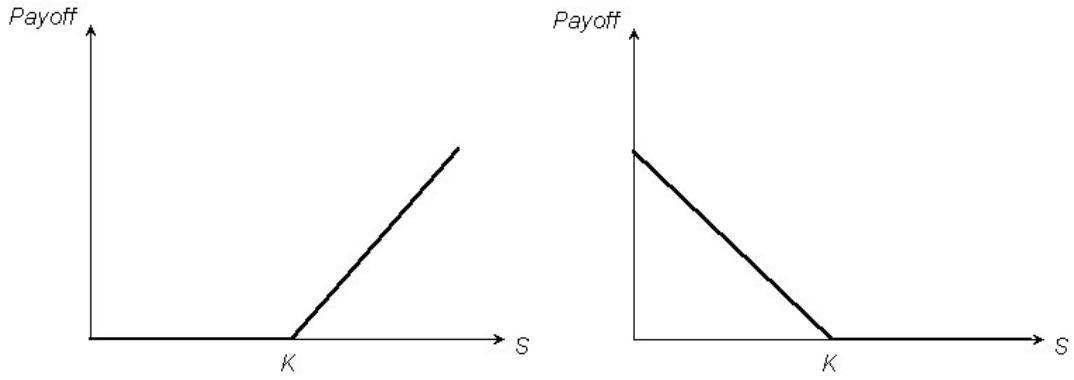

Figure 4: *schematic graphs of the payoff of European Options, without considering the discount factor and the price payed for the option. On the left for Call Option, on the right for Put. Source: Wikipedia*

Similarly, the payoff, p, of a put option on the same underlying asset is given by:

$$p \geq \max\left(Ke^{-rT} - S_0, 0\right).$$

An important relationship between European put and call options (just calls and puts from now on) is called put-call parity. Considering two options having the same exercise price and expiration date, one call and the other one put, their payoffs are linked by the following equation:

$$c + Ke^{-rT} = p + S_0$$

Which shows that the value of a call can be deduced from the value of a put with the same strike price and maturity, and vice versa.

Common underlying instruments for options are: Equity, Bonds, Futures, Indexes, Commodities, Currencies. Other styles of options traded in the markets, or OTC, are:

- American options: similar to European options, but they can be exercised not only at a fixed date but at every moment before the expiration date;
- Binary options: they pay the full amount if the underlying asset reaches a defined price before expiration, nothing otherwise;
- Asian options: in this case the payoff is determined by the average of the underlying price over a fixed time period;
- Exotic options: a large group of options that have more complex structures and/or underlying instruments.

Most of the literature that will be cited in this thesis regards the pricing of European options, because other styles of option do not fall within the scope of this discussion. From now on, with the term option will be indicated a European option.

Volatility

Volatility is defined in this context as the variation of the price of a financial instrument over time. It is generally measured by standard deviation of log-returns. It can be estimated using various methods. The first method, the historical volatility, consists in calculating the standard deviation of the stock log-return over some period:

$$\sigma_H = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T} s_t - \bar{s}}$$

Where σ_H is the historical volatility estimator, T is the total period considered, s_t is the stock log-return in the period t, $\bar{s}$ is the average of all the stock log-returns considered. The returns are calculated as follow:

$$s_t = \ln\left(\frac{S_t}{S_{t-1}}\right)$$

Where S_t is the stock price at the time t. Some problems about this method are highlighted by Figlewski (1994), such as the serial correlation in returns and the variation of volatility over time. Other methods involve GARCH models (Bollerslev, 1986), the simplest of which is GARCH(1,1), in which the unconditional volatility is given by the following relationship:

$$\sigma_t^2 = \omega + \alpha\sigma_{t-1}^2 + \beta\varepsilon_{t-1}^2$$

Where ω, α, β are the estimation parameters and ε_{t-1}^2 are squared residuals. Some studies have shown that using neural networks can improve the forecasting performance of GARCH models for volatility (Kristjanpolle, Fadic, & Minutolo, 2014).

Another method for volatility estimation involves the so-called Implied volatility (IV). It derives directly from the Black-Scholes model as the inverse function of the pricing model:

$$P = f(\sigma, \cdot), \quad \sigma_{IV} = g(P, \cdot)$$

Where P is the price of the option, $f(\cdot)$ is the pricing model, $g(\cdot)$ is its inverse function of f and σ_{IV} is the estimator of the implied volatility. The inversion is possible because $f(\cdot)$ is monotonically increasing in σ, so the inverse function theorem can be applied. However, this method doesn't have a closed form solution and an approximation method must be used, such as Newton's method.

Deep learning models have been developed to approximate the implied volatility and make predictions about future volatility, such as in Malliaris and Salchenberger 1996[13].

[13] Malliaris, Mary and Salchenberger, Linda (2004), Using neural networks to forecast the S&P 100 implied volatility, Elsevier Vol. 10 (2): 183-195

In his *The misbehaviour of markets*, Benoit Mandelbrot (2004)[14] states that saying that "volatility is easier to predict than prices" is an heresy. The statement highlights how is important to consider carefully how the volatility is calculated when pricing an instrument.

[14] Mandelbrot, Benoit (2004), The Misbehavior of Markets: A Fractal View of Financial Turbulence, Basic Books

Black- Scholes-Merton model

The model presented is the most influential model in the field of derivatives pricing, which led Robert Merton and Myron Scholes to be awarded the Nobel Prize for economics in 1997. The following derivation of the Black-Scholes differential equation follows the work of Robert Merton (Merton, 1973), which doesn't rely on the assumptions of the capital asset pricing model (Sharpe, 1964). The hypothesis of the model are discussed in details in the next paragraph.

In Black-Scholes-Merton world, the stock price, S, follows a Wiener process (or Brownian motion) in which the expected stochastic drift rate (the rate at which the mean of the process changes) is μS, for some constant μ. Thus, for a small temporal interval, Δt, the variation of the stock price is:

$$\Delta S = \mu S \Delta t$$

Taking the limit for $\Delta t \to 0$ and rearranging S:

$$\frac{dS}{S} = \mu dt$$

Now, integrating between 0 and T (for instance starting date and expiration date):

$$S_T = S_0 e^{\mu t}$$

Where S_T is the stock price at time T, S_0 is the stock price at time 0. This equation shows that, without uncertainty (μ is constant) the stock price would grow continuously compounded at a μ rate for the entire period considered.

Instead, adding the component of volatility to the equation, before taking the integral, the equation becomes:

$$dS = \mu S dt + \sigma S dz$$

Where dz is the limit of $\Delta z \to 0$ and has the following properties:

P1. Δz is normally distributed;

P2. mean of $\Delta z = 0$;

P3. variance of $\Delta z = \Delta t$.

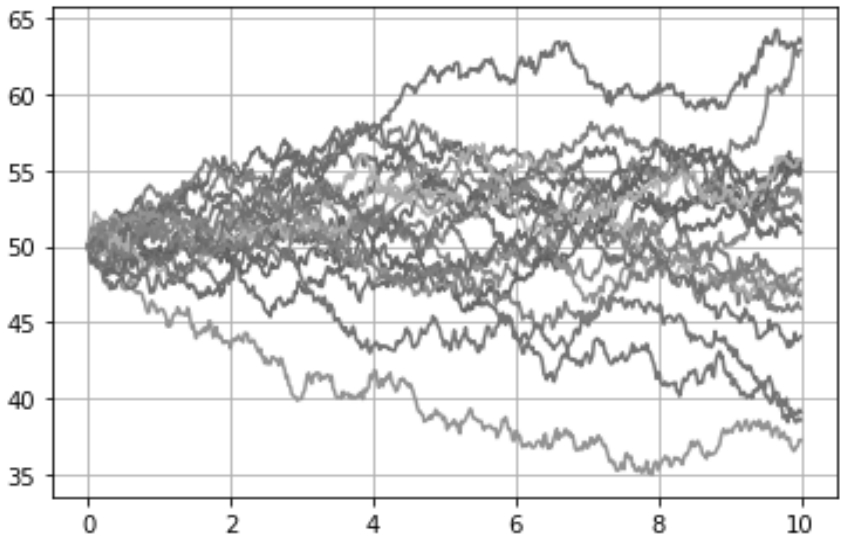

Figure 5: *representation of some of the possible trajectories of a geometric Brownian motion. Source: simulation in Python 3.7*

Now, supposing C the price of a call on the stock considered before. C has to be some function of S. By applying the Itō's Lemma for stochastic calculus (Itō, 1951), it can be demonstrated that the following relationship subsist:

$$dC = \left(\frac{\partial C}{\partial S}\mu S + \frac{\partial C}{\partial t} + \frac{1}{2}\frac{\partial^2 C}{\partial S^2}\sigma^2 S^2\right)dt + \frac{\partial C}{\partial S}\sigma S dz$$

Consider a portfolio built with the option and the underlying asset, such that there is a short position in one option and a long position in $+\frac{\partial C}{\partial S}$ stocks. The value of such portfolio is:

$$\mathcal{P} = -C + \frac{\partial C}{\partial S}S$$

The variation of $\mathcal{P}$ at an infinitesimal time is:

$$d\mathcal{P} = -dC + \frac{\partial C}{\partial S}dS$$

And substituting the equation for dC and dS written before:

$$d\mathcal{P} = \left(-\frac{\partial C}{\partial t} - \frac{1}{2}\frac{\partial^2 C}{\partial S^2}\sigma^2 S^2\right)dt$$

Because this equation doesn't have dz, for the same argument stated for dS without considering dz , the portfolio is riskless during dt and, given r as the constant risk-free rate:

$$d\mathcal{P} = r\mathcal{P}dt$$

Now, substituting in the precedent equation, simplifying for dt, the Black-Scholes-Merton parabolic differential equation is obtained:

$$rC = \frac{\partial C}{\partial t} + \frac{\partial C}{\partial S} rS + \frac{1}{2} \frac{\partial^2 C}{\partial S^2} \sigma^2 S^2$$

To solve the equation the boundary conditions must be specified. In this case is considered:

$$C = \max(S - K, 0), \qquad \text{when } t = T$$

for a call option and:

$$C = \max(K - S, 0), \qquad \text{when } t = T$$

for a put option.

Under these boundary conditions, it is possible to derive a closed form solution both for call, c, and put, p, options:

$$c = S_0 N(d_1) - K\, Ke^{-r\tau} N(d_2)$$

$$p = Ke^{-r\tau} N(-d_2) - S_0 N(-d_1)$$

Where $\tau = T - t$, $N(\cdot)$ is the cumulative probability distribution function of the standardized normal distribution, d_1 and d_2 are defined as:

$$d_1 = \frac{ln\,(S_0/K) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

$$d_1 = d_1 - \sigma\sqrt{\tau} = \frac{ln\,(S_0/K) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

It's important to notice how risk-preferences of individuals are not involved in the above formulas, hence they can't affect the solution. This lead to a Black-Scholes-Merton world where all investors can be considered Risk-Neutral, though this is just an expedient to obtain the solution of the differential equation. In fact, the formulas above can be applied (and are applied) in many real-world scenarios.

From the model presented the implied volatility can be calculated, inverting the function $c(\sigma, \cdot)$ or $p(\sigma, \cdot)$ for σ_{IV} and using numerical methods in order to solve the new equation found. The implied volatility, automatically calculated in this manner for every option in the dataset, will be used in the development and implementation of both the Black-Scholes model and the neural network architecture in the empirical part of the thesis.

Hypothesis

The assumptions implied when deriving the Black–Scholes–Merton differential equation are:

i. the stock price follows a Brownian motion with constant μ and σ;
ii. short selling is allowed;
iii. all instruments are perfectly divisible and infinitely available;
iv. there are no market frictions, such as transactions costs or taxes;
v. no dividends are paid during the life of the derivative;
vi. there are no arbitrage opportunities, neither type I nor type II;
vii. securities trading is in continuous time;
viii. the risk-free rate is constant and the same for all expiration dates.

The first assumption, described in a rigorous way in the above paragraph, can be relaxed assuming that also volatility follows a Brownian motion, like in Hull, White (1987)[15] and Heston (1993). The second hypothesis allows to consider both positive and negative numbers and, combined with the third assumption, permits to use all Real numbers, $\mathbb{R}$, in the development of the model, simplifying the approach. Fourth and fifth assumptions have been relaxed in the literature, leading to more complex models involving, in the case of growing dividends, the rate of growth, q, which is incorporated in the formula. The presence of transaction costs has been addressed by Leland (1985)[16], who started with discrete time and considered a proportional costs model, adjusting the volatility of the Black-Scholes formula as follow:

$$\hat{\sigma}^2 = \sigma^2 \left(1 + \frac{\sqrt{2/\pi}\, k}{\sigma \sqrt{\Delta t}} \right)$$

Where k is the cost proportion of S.

As regards arbitrage opportunities, the work of Kreps (1981) highlights that "the non-existence of 'arbitrage opportunities' is necessary and sufficient for the existence of an economic equilibrium"[17] which makes this assumption difficult to avoid in developing any economic model. The seventh hypothesis allows to work in continuous time, in the domain of $\mathbb{R}^+$, even thought, as presented before, many models start with the assumption of discrete time for simplicity of comprehension. The last hypothesis regards the risk-free rate, which is defined as the rate of return of a riskless investment. This assumption can be relaxed using various risk-free rates, depending on the type of option and the type of underlying asset, and then applying Black-Scholes model using those different rates.

[15] Hull, Jhon, White, Alan (1987), The Pricing of Options on Assets with Stochastic Volatilities, The Journal of Finance Vol. 42 (2)
[16] Leland, H. E. (1985), Option Pricing and Replication with Transactions Costs, The Journal of Finance Vol. 40 (5): 1283-1301
[17] Kreps, D. M. (1981), Arbitrage and equilibrium in economies with infinitely many commodities, Journal of Mathematical Economic Vol. 8 (1):15-35

Deep Learning models

Artificial intelligence (or Computational intelligence) is the study of intelligent agents, namely agents that acts in an environment in an intelligent way, mimicking the human intelligence (Poole, Mackworth, & Goebel, 1998). A subfield of artificial intelligence is machine learning, a set of algorithms and statistical models that perform tasks without explicit instructions, learning by discovering patterns in data by themselves. A particular set of machine learning models is called deep learning, which is based on neural networks structures. The applications of deep learning vary from computer vision (Voulodimos et al., 2018) to speech recognition (Amodei et al., 2016), from natural language processing (Collobert & Weston, 2008) to financial predictions and classifications (Heaton, Polson, & Witte, 2018). All deep learning (and machine learning) models are often referred to as representation learning or feature learning, to state that this type of models allows the machine to automatically discover the representation needed to execute the task. Deep learning models can be supervised or unsupervised. In the first case data are labelled, or the objective function is specified, and the model tries to learn from other features in order to get the required result (the label or the approximation of the function). In the second case no objective is given to the model and it has to figure out patterns between data by itself or modelling the probability distribution of the inputs.

Well-known deep learning models are:

- Perceptron: as discussed before, it takes several inputs and directly gives the output.
- Feedforward neural network: a set of perceptrons, fully interconnected with each other and distributed in layers, generally one input layer, several hidden layers and one final output layer. It is trained via the backpropagation algorithm and gradient-based optimization.
- Recurrent neural network: similar to feedforward networks, but with so called recurrent cells (artificial neurons) in it, which take their own value as an input for themselves.
- Long/Short term memory: these are recurrent neural networks with some cells that account for time gaps (or time lags) between input data. These models are able to “remember” and “forgot” data while training.
- Autoencoder: network with hidden layers smaller than input and output layers, where the inputs are “summarized” and then recreated in the output as similar as possible. These models are an example of unsupervised learning and are used to generalise data or to compress large files.
- Markov chain: this is a relatively old concept (Markov, 1906) that could be express in terms of a neural network with probabilities that determine the activation of the next neuron (or node). These models don’t have a structure of layers and are often used for classifications based on probabilities.
- Convolutional neural network: model typically used for image recognition, in which the layers are 2-dimentional or 3-dimentional. The hidden layers are intended to identify patterns gradually more complex (for instance colour gradient, lines, faces), until the whole picture content is correctly detected.

- Generative adversarial networks: this model consist in two neural networks that compete in a “game" against each other. One network is called the generator, which tries to produce new realistic data, learning from available training data, and the second network is called the discriminant, which has to identify whether the data given from the first network are real or not. Their training is based on the theory of games, until a Nash equilibrium is reached, where the generator must produce perfectly realistic data and the discriminant can’t do better than random choice.

The quintessential deep learning model is the feedforward neural network (Goodfellow, Bengio, & Courville, 2016), which will be described in the next paragraph and will be used for option pricing in the empirical part of the thesis.

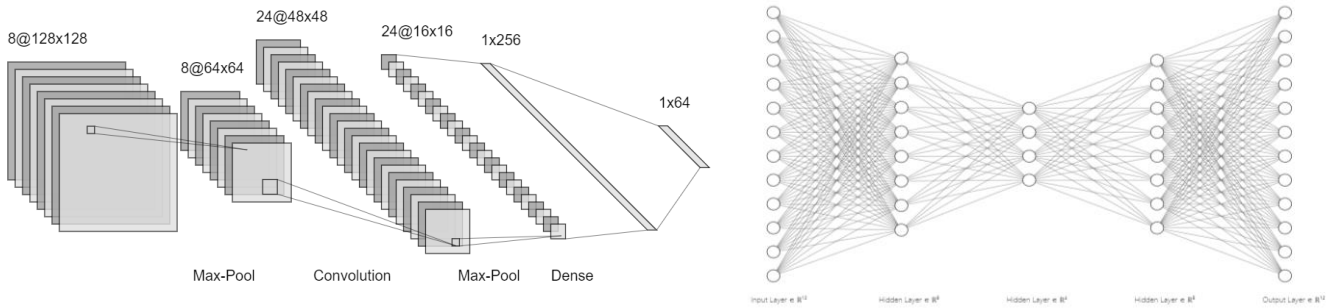

Figure 6: *example of graphic representation of a convolutional neural network (on the left) and an autoencoder (on the right). Source: created in http://alexlenail.me/*

Feedforward Neural Networks

Deep feedforward networks (or multilayer perceptron) are models whose objective is to approximate some function $f(\cdot)$ by utilizing a map $y = \hat{f}(\cdot, \theta)$ and learning the value of the parameters θ that lead to the best approximation. The name "network" is attributed because the function $f(\cdot)$ is approximated using a composition of many functions (the number depends on the size of the network). The adjective "feedforward" is added to highlight that there are no connections in reverse order between the neurons and the information flows always from one layer to the next and not vice versa. The adjective "deep" refers to the depth of the network, expressed by the number of layers of the architecture. An example of the structure of a neural network is the following:

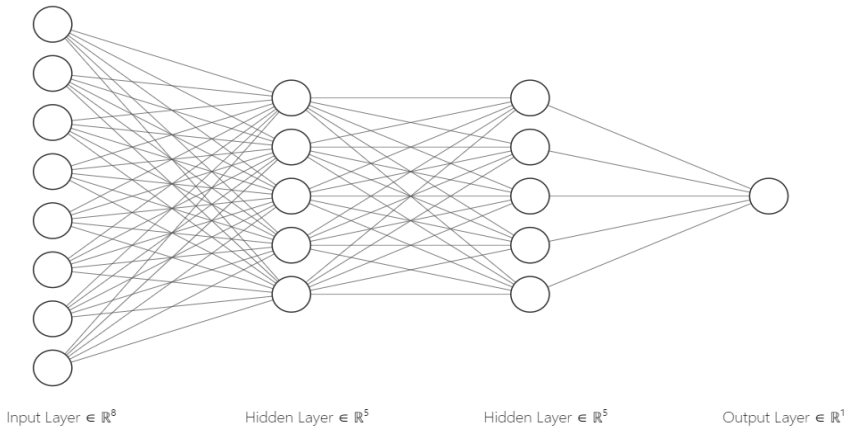

Figure 7: *example of the structure of a feedforward neural network.*
Source: created in http://alexlenail.me/

In the above graphical example, the first layer is referred to as the input layer and has eight neurons, the two middle layers are called hidden layers and have five neurons each, the last layer is the output layer with one neuron. All layers in this case are fully connected. The neurons on the first layer takes the inputs, x_i, and transfer them to every neuron in the second layer with every connection weighted with weights w_i. In matrix form:

$$z = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + b$$

Where $\boldsymbol{x}$ and $\boldsymbol{w}$ are the vectors of input values and weights, b is called bias. To obtain the value for a generic neuron in the second layer, to z is applied a so-called activation function, which is generally non-linear and filters the value of z before passing it to the following layer. Thus, the value that reaches the third layer is:

$$h = f_{(1)}(z) = f_{(1)}(\boldsymbol{x}, \boldsymbol{w}, b)$$

the most used activation functions and their role in the performance of networks will be addressed in a later paragraph. After that, all values produced from the second layer, filtered and, again, weighted, arrive in a generic neuron of the third layer, whose value is given by some function $f_{(2)}(\boldsymbol{h}, \boldsymbol{\omega}, c)$ where $\boldsymbol{h}$ is the vector of

values of the preceding layers, $\boldsymbol{\omega}$ is the new set of weights, c is the new bias. The value of the output layer is the composition of the functions described before:

$$g(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{b}) = f_{(2)}\big(f_{(1)}(\boldsymbol{x}, \boldsymbol{w}, b), \boldsymbol{\omega}, \boldsymbol{c}\big)$$

where $\boldsymbol{W}$ is the matrix obtained using as columns the vectors of $\boldsymbol{w}$ and $\boldsymbol{\omega}$.

Finally, the output layer gives the "guess" of the model, y, which is generally not accurate at first, because the weights are chosen randomly, and they are not optimized for the specific objective. The way to allow the model to learn and optimize itself is via backpropagation (Rumelhart, , Hinton, & Ronald, 1986) and stochastic gradient descent (Robbins & Monro, 1951).

Backprogragation algorithm

Often backpropagation is referred to be the entire process that leads a neural network to learn from data, but backpropagation refers only to the method of computing the gradient, which is used by algorithms, like gradient descent, to perform learning. In order for a neural network to learn, it is necessary to specify a loss (or objective) function $L(\boldsymbol{x}, \boldsymbol{y})$, which depends on the set of input data $\boldsymbol{x}$ and output data $\boldsymbol{y}$. Generally, for a classification problem the loss function used is the Cross Entropy, $H(\cdot)$, which compares the estimated probability distribution, q, with the actual probability distribution, p, of the data:

$$H(p,q) = -\sum_{x\in\mathbb{Z}} p(x)\log q(x) \qquad \text{for discrete time distributions}$$

$$H(p,q) = -\int_{\mathbb{R}} p(x)\log q(x)\, dr(x) \qquad \text{for continuous time distributions}$$

where $r(x)$ is a Lebesgue measure on a Borel σ-algebra.
For a prediction or estimation problem, the mean squared error, MSE, is the most used metric:

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

Where Y_i are actual values and $\hat{Y}_i$ are the values predicted by the model, n is the number of observation in the test set chosen, which has to be different from the training set, to avoid data memorization problems. Sometimes also mean absolute error is used for this purpose.

$$MAE = \frac{\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|}{n}$$

To explain backpropagation, the function which gives the output for a general feedforward neural network is defined as follow:

$$g(\boldsymbol{x}) \coloneqq f_{(n)}(\boldsymbol{W}^n f_{(n-1)}(\boldsymbol{W}^{n-1}\cdots f_{(1)}(\boldsymbol{W}^1\boldsymbol{x})\cdots)$$

A generic loss function is applied for a generic set of inputs and outputs $L(g(\boldsymbol{x}), \boldsymbol{y})$. Backpropagation permits to compute the gradient of the loss function as expressed above. This means computing all partial derivatives of the loss function with respect to weights and biases: $\partial L/\partial w_j^l$ and $\partial L/\partial b_j^l$. Where the apex l indicates the layer and the subscript j indicates the j^{th} neuron in the layer. In order to achieve that, an intermediate quantity, ε_j^l, is introduced and it's called the error of the j^{th} neuron in the layer l [18]. It subsist: $\varepsilon_j^l = \partial L/\partial z_j^l$ by definition, where z_j^l is the weighted input to the activation function for neuron j in layer l. Now $h_j^l = f_{(l)}\left(z_j^l\right)$ is defined, and, merging the notions above:

[18] The absence of the subscript j indicates that the value refers to an entire layer.

$$\varepsilon_j^l = \frac{\partial L}{\partial h_j^l} f_{(l)}{}'(z_j^l)$$

Which, for an entire layer, becomes:

$$\varepsilon^l = \nabla_h L \cdot f_{(l)}{}'(z^l)$$

Where $\nabla_h L$ is a vector whose components are the partial derivatives $\partial L/\partial h_j^l$. $\nabla_h L$ can be interpreted as the rate of change of $L(\cdot)$ with respect to the output h of the activation function. This equation is furnished without proof, but the concept from which it comes is that $\partial L/\partial z_j^l$ can be expressed in terms of h_j^l as follow:

$$\frac{\partial L}{\partial z_j^l} = \frac{\partial L}{\partial h_j^l}\frac{\partial h_j^l}{\partial z_j^l}$$

This permits to rewrite $\nabla_h L$ in terms of ε^l.

Another step in understanding backpropagation is linking the error in one layer to the next layer and the relationship that subsists is derived from $\varepsilon_j^l = \partial L/\partial z_j^l$ and $\varepsilon_j^{l+1} = \partial L/\partial z_j^{l+1}$. Using the chain rule:

$$\varepsilon_j^l = \frac{\partial L}{\partial z_j^l} = \sum_k \frac{\partial L}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l}\varepsilon_k^{l+1}$$

Recalling $z = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + b$, now expressed in terms of summation, for the layer $l+1$ and the j^{th} neuron:

$$z_j^{l+1} = \sum\nolimits_k w_{kj}^{l+1} h_j^l + b_k^{l+1} = \sum\nolimits_k w_{kj}^{l+1} f_{(l)}(z_j^l) + b_k^{l+1}$$

Differentiating:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f_{(l)}{}'(z_j^l)$$

And, substituting for $\partial z_k^{l+1}/\partial z_j^l$ in the precedent equation of ε_j^l , the formula that links one layer to the next is given:

$$\varepsilon_j^l = \sum_k w_{kj}^{l+1} f_{(l)}{}'(z_j^l)\, \varepsilon_k^{l+1}$$

or, in matrix form:

$$\varepsilon_j^l = \left[(\boldsymbol{w^{l+1}})^T \boldsymbol{\delta^{l+1}}\right] \cdot f_{(l)}{}'(z_j^l)$$

As regards weights and biases, the following relationships are given without proof (they follow from the application of the chain rule):

$$\varepsilon_j^l = \frac{\partial L}{\partial b_j^l}$$

$$\varepsilon_j^l \cdot h_k^{l-1} = \frac{\partial L}{\partial w_{kj}^l}$$

Resuming how the backpropagation algorithm works, we can identify four main steps:

1. Set the output of the first activation function h^1;
2. For each layer $l = 2,3, \dots, \mathcal{L}$, compute z^l ,h^l and ε^l;
3. For each layer $l = \mathcal{L} - 1, \mathcal{L} - 2, \dots, 2$, compute ε^l relating it to its next layer;
4. The gradient of $L(\cdot)$ is given by the following equations for weights and biases:

$$\varepsilon_j^l = \frac{\partial L}{\partial b_j^l}$$

$$\varepsilon_j^l \cdot h_k^{l-1} = \frac{\partial L}{\partial w_{kj}^l}$$

The third step indicates why the algorithm described is called backpropagation.

After the gradient is computed using backpropagation algorithm, to optimize the model (and to train it) a gradient-based optimization algorithm has to be applied. The most used optimization methods are described in detail in the next paragraph.

Gradient-based Optimization

The stochastic gradient descent algorithm is an iterative method to optimize the loss function (or objective function) expressed above. Consider n observations in the dataset and an objective function $L(\boldsymbol{\gamma})$ to be minimized with a set of parameters $\boldsymbol{\gamma}$ that minimize the function and that have to be estimated. The basic algorithm works as follow:

1. An initial vector $\boldsymbol{\gamma}$ is chosen and a learning rate η is defined**;**
2. for $i = 1,2, \dots, n$ do: $\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \eta \nabla L_i(\boldsymbol{\gamma})$[19]**;**
3. repeat until a minimum is reached.

The algorithm is performed by randomly shuffle examples in the training set and can be executed analysing more than one example at the same time. The number of examples executed at the same time is called mini-batch size.

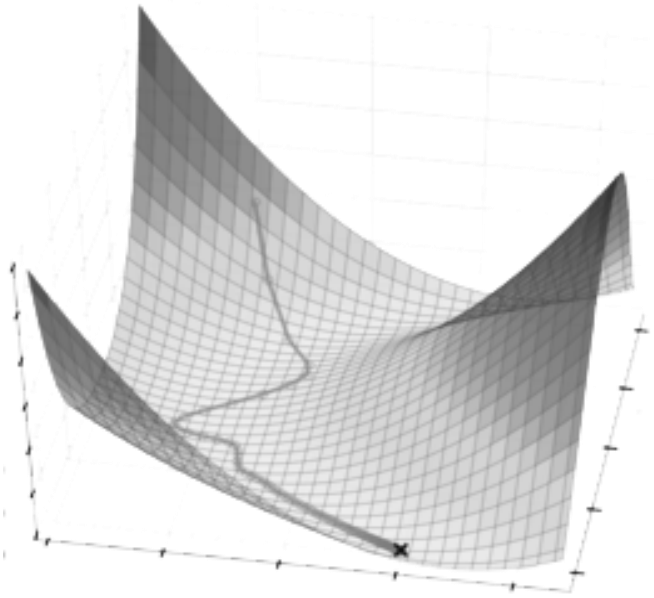

Figure 8: *example of the path followed by gradient descent algorithm to reach the local minima of the gradient (the black X) from its starting point.*
Source: http://dsdeepdive.blogspot.com/

Different optimization methods can be used to tune the learning rate η to train the model. Often the rate is not constant over the entire training, but it changes between iterations by tuning two parameters: decay and momentum. The first one refers to the gradual decrease of the learning rate while approaching the minimum (this is done to avoid jumping back and forth over a minimum). The second is achieved by adding a component in each update, which is a linear combination of the gradient and the previous update:

$$\Delta\boldsymbol{\gamma} \coloneqq \alpha\Delta\boldsymbol{\gamma} - \eta\nabla L_i(\boldsymbol{\gamma})$$

[19] In this case and in the following examples, the symbol ≔ indicates an update of the value, it is not a "definition" or "equality" symbol.

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} + \Delta\boldsymbol{\gamma}$$

Well-known optimization methods are: AdaGrad (Duchi, Hazan, & Singer, 2011), RMSProp (Hinton, Overview of mini-batch gradient descent) and Adam (Kingma & Ba, 2014).

The first method (Adaptive gradient algorithm) consists in updating the parameters as follow:

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \frac{\boldsymbol{\eta}}{\sqrt{\boldsymbol{G}}} \boldsymbol{\nabla L_i(\gamma)}$$

Where:

$$\boldsymbol{G} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{\nabla L_i(\gamma)}\, \boldsymbol{\nabla L_i(\gamma)^T}$$

The second method (Root Mean Square Propagation) was never published by Geoffrey Hinton in a formal academic paper, but it has become one of the most used methods in practice. The update:

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \frac{\boldsymbol{\eta}}{\sqrt{\boldsymbol{v(\gamma, t)}}} \boldsymbol{\nabla L_i(\gamma)}$$

Where:

$$\boldsymbol{v(\gamma, t) = \rho v(\gamma, t-1) + (1-\rho)(\nabla L_i(\gamma))^2}$$

ρ is the so-called forgetting factor, t is the mini-batch size, $v(\boldsymbol{\gamma}, t)$ is called velocity.

Adam (Adaptive Moment Estimation) is an evolution of RMSProp, which uses velocity for both the first and second momentum of the gradient:

$$\boldsymbol{v_1(\gamma, t+1) = \beta_1 v_1(\gamma, t) + (1-\beta_1)\nabla L_i(\gamma)}$$

$$\boldsymbol{v_2(\gamma, t+1) = \beta_2 v_1(\gamma, t) + (1-\beta_2)(\nabla L_i(\gamma))^2}$$

And the parameters are updated as follow:

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \eta \frac{\hat{v}_1(\boldsymbol{\gamma}, t+1)}{\sqrt{\hat{v}_2(\boldsymbol{\gamma}, t+1)} + \boldsymbol{\varepsilon}}$$

Where $\hat{v}_i$ are the estimators of velocities, $\boldsymbol{\varepsilon}$ is a small addend to avoid division by 0, β_i are the forgetting factors.

Activation functions

The activation function of a neuron can be considered as a filter that is applied to the input of the node, which is given from a linear operation (matrix or vector multiplications), to modify it and transfer to the subsequent layer. Based on the type of function, activation assumes different interpretations. Below there is a list and a brief explanation of some well-known activation functions.

The ReLU function (Rectified Linear Unit) is given by:

$$f(x) = \begin{cases} 0 \; for \; x \leq 0 \\ x \; for \; x > 0 \end{cases}$$

It avoids negative values as output of the neuron and is useful in the problems that involves only positive values.

The Sigmoid (or Logistic) function is given by:

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

It varies between 0 and 1 and can be interpreted as transforming real numbers into probabilities. It is often used in classification problems where a probability for the output is needed.

The TanH (Hyperbolic Tangent) function is given by:

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

It varies between -1 and 1 and it is often used in classification problems involving two classes.

The ELU function (Exponential Linear Unit) is given by:

$$f(\alpha, x) = \begin{cases} \alpha(e^{x} - 1) \; for \; x < 0 \\ x \qquad\qquad for \; x \geq 0 \end{cases}$$

It differs from other activation functions for the parameter $\alpha > 0$, which needs to be tuned. It works similarly to ReLU function, but it's smoother and assigns values different from 0 to every real number (except 0).

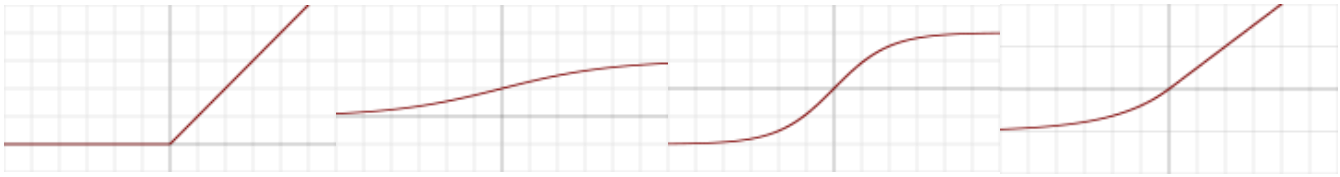

Figure 9: *schematic graphs of most used activation functions. From left to right: ReLU, Sigmoid, TanH, ELU. Source: Wikipedia*

Chapter 4

Empirical Part

Data and environments

The dataset used to implement and compare the two models is the *L3 Historical CSV Data Sample*[20] which includes 928'674 observations of put and call European options traded on the Chicago Board Options Exchange. All data are recorded on 15th August 2019. The following features are included:

Variable	**Class**	**Description**
UnderlyingSymbol	Multinomial	One to four letters, identifying the country of the company in which the option is listed.
OptionSymbol	Multinomial	The ISIN code of the option, which identifies univocally the instrument.
Type	Binomial	It defines whether the option is a call or a put.
Expiration	Date	Settlement date of the contract.
DataDate	Date	It coincides with the observation date of the option, so it can be different from the starting date of the contract.
UnderlyingPrice	Numerical	The price of the underlying asset of the option.
Strike	Numerical	The strike price of the option.
Last	Numerical	The price at which the last trade has been done.
Bid	Numerical	The theoretical maximum price at which a buyer is willing to pay for the option.
Ask	Numerical	The theoretical minimum price at which a seller is willing to take for the option.
Volume	Numerical	The trading volume of the option.
OpenInterest	Numerical	The actual total number of outstanding options that have not been settled yet.
IVMean	Numerical	The average between the bid implied volatility of the option and the ask implied volatility.
IVBid	Numerical	The bid implied volatility.
IVAsk	Numerical	The ask implied volatility.
Delta	Numerical	The derivative of the option price with respect to its underlying price.
Gamma	Numerical	The sensibility of the option price with respect to its delta.
Theta	Numerical	The derivative of the option price with respect to its time to maturity.
Vega	Numerical	The derivative of the option price with respect to its implied volatility.

Table 1: *a list of all the features in the dataset, with an identification of the type of variable and a brief description of their meaning*.

Not all the features present in the dataset are necessary to the scope of the thesis, and some have been manipulated to be easier to process for the models. The details of the preparation of the dataset are described in the next paragraph.

Another column needed to implement the Black-Scholes model would have been the risk-free rate, but it is missing and no data about the risk-free rate for every observation is available. As a consequence, a risk-free

[20] Source: https://www.historicaloptiondata.com/

rate is chosen arbitrarily, and it will be the same for all the observations. This represents a strong assumption and its implications will be discussed later. The risk-free rate chosen is the US Tree Months Treasury Bill rate at the 15th August 2019 date. The choice has been made because the average time to maturity of all the options in the dataset is 0.4028, which is close to 3 months, and the date is the date at which all options have been observed. The value of the rate discussed is 0.0187 or 1.87%.

The models will be implemented using Python 3.7 programming language. The choice has been made due to the simplicity of the language and all the pre-built libraries for machine learning, which make the workflow faster and the debugging process easier. In particular, the libraries used throughout the empirical part of the thesis are:

- Pandas (McKinney, 2010), used to read, write and manipulate the dataset;
- NumPy (Oliphant, 2006), which uses arrays as the main data structure to perform computations;
- SciPy (Virtanen et al., 2019), a large library, which includes a variety of branches of science, used in this case for statistical tools;
- Time, present in the Python Standard Library[21], used to calculate the time of computation and training for the two models;
- Matplotlib (Hunter, 2007), used to make the graphs and plots to have a visual interpretation of the models;
- TensorFlow (Abadi et al., 2015), an interface from Google made to implement machine learning models, particularly deep learning ones. The name derives from the fact that data are imported in TensorFlow using tensors, which speeds up the process of training the model.

[21] Ufficial documentation: https://docs.python.org/3/library/

Cleaning and preparing the data

The dataset doesn't have missing or wrong recorded cells, thus the process of cleaning and preparing the data is made only by working on columns. In particular, some columns have been eliminated because they don't enter as inputs for the models, such as Volume, OpenInterest, IVBid, IVAsk, UndelyingSymbol, Last, Delta, Gamma, Theta, Vega.

Next, a column called "Average Price" has been created and calculated as the arithmetic average of Bid and Ask prices:

$$Average\ Price = \frac{Ask + Bid}{2}$$

This column will be used as the price label for both the models. The performance metrics will be calculated with respect to this value.

As regards time, the two columns Expiration and DataDate have been converted from date format to real numbers as a fraction of a 252 working days per year. Thus, the new column TimeToMaturity represents the number of days between the observation date and the settlement date of the option.

As discussed before, the risk-free rate will be the same for all options in the dataset, thus a global variable of 1.87% is added as well.

Next all the observations are divided in two sets, the training set and the test set. This operation is necessary for the implementation of the neural network model, which requires data to train, but not for the Black-Scholes model. However, both models must be compared using the same test set. The division is made using 80% of the dataset as train set and 20% as test set, sampled randomly. At the end, the original dataset is split in 742'939 training samples and 185'735 test samples.

The Python code for importing and preparing the dataset is presented below.

```
#Importing Libraries and csv
import pandas as pd
import numpy as np
import sympy as sy
import scipy.stats as si
from scipy.stats import norm
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import math
import seaborn as sns
import time
ds=pd.read_csv('.../L3_options_20190815.csv')
```

```
#Creating the target variable 'Average Price' as the average of Bid and Ask price
ds['Average Price']=(ds['Bid']+ds['Ask'])/2
ds.drop('Bid', axis=1, inplace=True)
ds.drop('Ask', axis=1, inplace=True)

#Creating the variable 'TimeToMaturity' as a fraction of 252 yearly working days
date_format = "%m/%d/%Y"
ds['StartDay'] = pd.to_datetime(ds[' DataDate']).sub(pd.Timestamp('2019-01-01')).dt.days
ds['ExpirationDay'] = pd.to_datetime(ds['Expiration']).sub(pd.Timestamp('2019-01-
01')).dt.days
ds['TimeToMaturity'] = (ds['ExpirationDay'] - ds['StartDay'])/252
ds.drop('StartDay', axis=1, inplace=True)
ds.drop('ExpirationDay', axis=1, inplace=True)

#Risk-Free rate, the US 3 months treasury bill rate
rf = 0.0187

#Deleting the unused columns
ds.drop('Volume', axis=1, inplace=True)
ds.drop('OpenInterest', axis=1, inplace=True)
ds.drop('IVBid', axis=1, inplace=True)
ds.drop('IVAsk', axis=1, inplace=True)
ds.drop('UnderlyingSymbol', axis=1, inplace=True)
ds.drop('Expiration', axis=1, inplace=True)
ds.drop(' DataDate', axis=1, inplace=True)
ds.drop('Last', axis=1, inplace=True)
ds.drop('Delta', axis=1, inplace=True)
ds.drop('Gamma', axis=1, inplace=True)
ds.drop('Theta', axis=1, inplace=True)
ds.drop('Vega', axis=1, inplace=True)
ds.drop('OptionSymbol', axis=1, inplace=True)

#train-test split
train_ds = ds.sample(frac=0.8,random_state=0)
test_ds = ds.drop(train_ds.index)
```

On volatility

As discussed before, implied volatility is provided by the dataset and it is calculated both for bid (IVBid) and ask (IVAsk) prices and, then, their average (IVMean). The IVMean will be used in next paragraphs as a feature for both models and this point, as will be discussed later, is a strong assumption, because it implies that the volatility considered by both models is constant and it is calculated by approximation, without a closed form solution. Furthermore, the volatility calculated as the average of the IVBid and IVAsk doesn't necessarily matches with the "real" implied volatility of the column "Average price" calculated before. Therefore, part of the error in both models could be attributed to forcing the use of volatility without an appropriate calculation, which could have been achieved using one of the methods discussed before, in the volatility paragraph.

Performance metrics

Two types of measures will be used to evaluate the performance of the models, one involving the distance between the predicted price and the actual price, the other one involving the temporal aspect of the computation. Initially the first measure considered was the Root Mean Squared Error (RMSE) or root mean squared deviation, calculated as the square root of the sum of all squared errors, divided by the number of observations:

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

Where n is the number of observations in the dataset, $\hat{y}_i$ is the model estimation for the price of the ith observation, y_i is the actual price of the option. This measure would have been chosen for many reasons, firstly because is a synthetic and easy to interpret measure of performance for the models. It is also easy to calculate and implement and it can be used as a loss function for the deep learning model.

However, in literature some have argued that Mean Absolute Error (MAE) is a better performance metric for machine learning problems like the one addressed in this thesis (Willmott & Matsuura, 2005). The RMSE is more appropriate than the MAE to represent model performance when the error distribution is expected to be Gaussian (Chai & Draxler, 2014), but in this case this assumption does not hold. In fact, the distribution of errors in both models is expected to be leptokurtic.

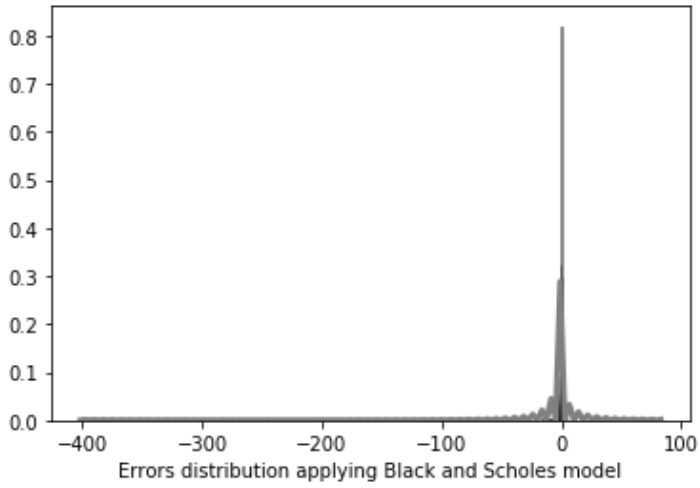

Figure 10: *histogram of the distribution of errors applying the Black and Scholes model to the test set.*

So, the deep learning model will be trained to minimize MAE as loss function, and the same measure will be used as performance metric for both models.

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n}$$

The second performance metric will be the time necessary for the hardware to compute and apply the model to all test's observations. This measure is introduced to compare the models both in spatial and temporal aspect.

The mere “time to compute” (measured in fraction of seconds) is chosen instead of computational complexity (measured using the Big O notation) because it is a measure easy to understand and has a practical implication, while computational complexity requires more sophisticated techniques to be calculated and has less immediate explicative power.

Implementing Black-Scholes model

Using NumPy, which enables to use arrays as inputs, the Black-Scholes formula can be implemented as a function $f(S_0, K, \tau, r, \sigma, Type)$, where, translating in dataset's columns, S_0 indicates the UnderlyingPrice, K indicates Strike, τ indicates TimeToMaturity, r the constant risk-free rate, σ the IVMean and Type indicates whether the option is call or put.

Second step is to define d_1 and d_2 as follow:

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

$$d_1 = \frac{\ln(S_0/K) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

Subsequently, using an if condition to separate call and put options, the formulas are applied:

$$c = S_0 N(d_1) - K\, Ke^{-r\tau} N(d_2)$$

$$p = Ke^{-r\tau} N(-d_2) - S_0 N(-d_1)$$

Then the model can be applied to the test set, simply applying $f(\cdot)$ to each row. Finally, MAE is calculated, creating a column for the difference between "Average price" and the model prediction and taking the average of all the values in that column. The following code has been written to perform the operations described above:

```
#Defining Black and Scholes formula
def BS_model (S, K, T, r, sigma, option = 'call'):

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

    if option == 'call':
        result = (S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2))
    if option == 'put':
        result = (K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1))
    return result
```

```
#Applying Black-Scholes formula to test set and counting time to compute
start = time.time()
test_ds['BS_Price']= np.where(test_ds['Type']=='call',
                        BS_model (test_ds['UnderlyingPrice'],
                                        test_ds['Strike'],
                                        test_ds['TimeToMaturity'],
                                        rf,
                                        test_ds['IVMean'], option = 'call'),
                        BS_model (test_ds['UnderlyingPrice'],
                                        test_ds['Strike'],
                                        test_ds['TimeToMaturity'],
                                        rf, test_ds['IVMean'], option = 'put'))

#Calculating the mean absolute error of the model
test_ds['Absolute_error']= np.absolute(test_ds['Average Price'] - test_ds['BS_Price'])
end = time.time()
MAE = np.mean(test_ds['Absolute_error'])
```

The results of the above implementation are discussed later in this chapter.

Neural network architecture

Before applying the chosen feedforward neural network to evaluate the price of options, some preparing operations have to be performed. First, because the neural network takes as inputs only numerical features, the column "Type" has been transformed using 1 for calls and 0 for puts[22]. Next, normalization of the data is performed, using the mean and standard deviation for each column:

$$z = \frac{x - mean(x)}{st.dv.(x)}$$

Where x represents a vector of original data, z the vector of standardized data, while $mean(\cdot)$ and $st.dv.(\cdot)$ are the functions for mean and standard deviation respectively. After this process is applied the variable representing the risk-free rate disappears, leaving all zeros, therefore it's eliminated and the model will be trained using only UnderlyingPrice, Strike, IVMean, TimeToMaturity, Type_code (the code referring to the Type).

Then the model is built using six layers:

- One input layer with five neurons, one for each of the five features;
- Four hidden layers, each with sixty-four neurons and different activation functions;
- One output layer with one neuron, which gives the price of the option.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 64)                384
_________________________________________________________________
dense_16 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_17 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_18 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 65
=================================================================
Total params: 12,929
Trainable params: 12,929
Non-trainable params: 0
```

Table 2: *a summary of the layers of the neural network, without considering the input layer. The neural network will update multiple times its 12'929 parameters while training.*

As regards the hidden layers, the first one has "linear" activation functions, which leaves the linear combination of the input data that arrives to each neuron unchanged. The second, third and fourth hidden layers have the same activation function, which is the ReLU (Rectified Linear unit):

[22] This choice is arbitrary and shouldn't have any impact on the final result. However, inverting 1 and 0 to indicates calls and puts can lead to a completely different set of weights and biases after the model is trained.

$$f(x) = \begin{cases} 0 \; for \; x \leq 0 \\ x \; for \; x > 0 \end{cases}$$

This choice seems simplistic at first, but it works well in this case. Other functions like ELU and TanH have been previously considered, but ReLU in each of the three layers lead to the best result for the model.
For the output layer, ReLU has been chosen as well and its interpretation is plain to understand: it gives non-negative values, hence non-negative prices.

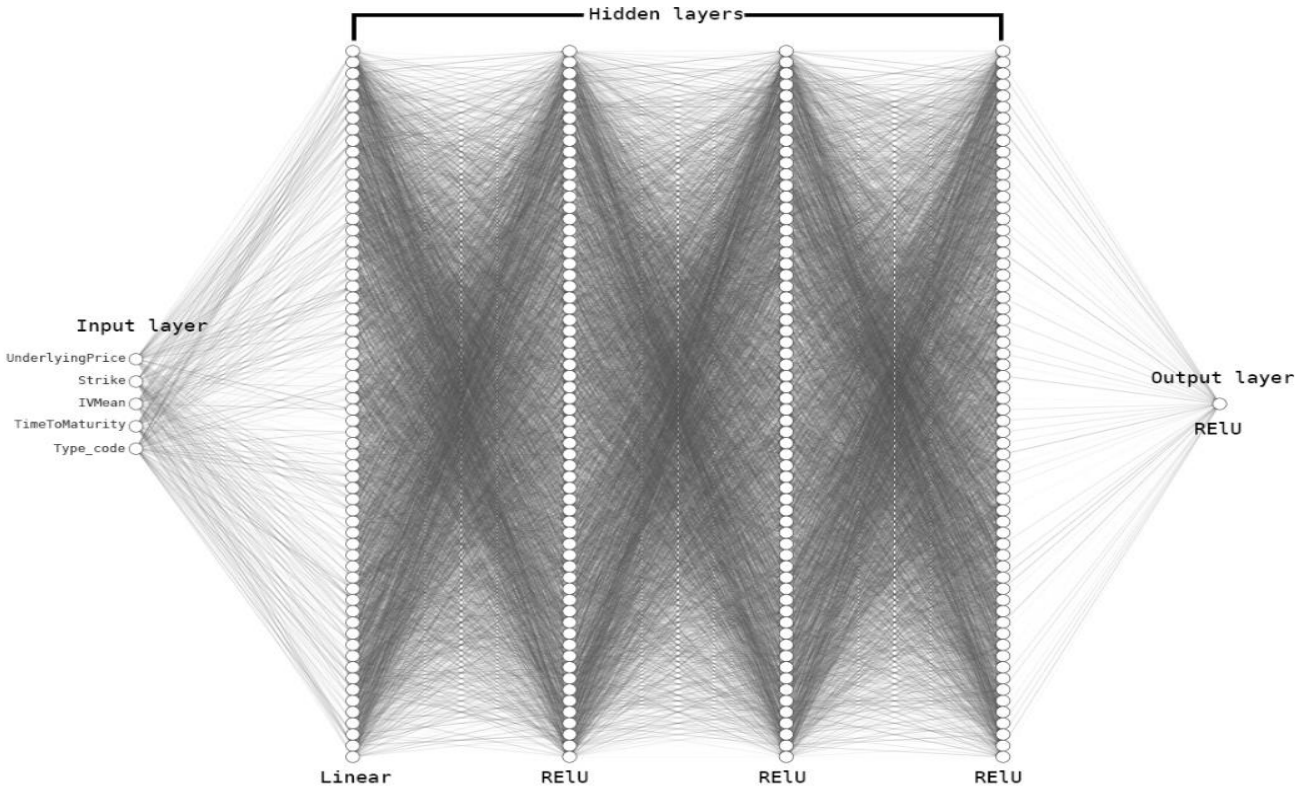

Figure 11: *a synthetic representation of the architecture of the neural network built.*

As discussed before, the loss function used to be minimized is MAE and the optimizer chosen is Adam (Kingma & Ba, 2014) :

$$v_1(\boldsymbol{\gamma}, t+1) = \beta_1 v_1(\boldsymbol{\gamma}, t) + (1-\beta_1)\nabla L_i(\boldsymbol{\gamma})$$

$$v_2(\boldsymbol{\gamma}, t+1) = \beta_2 v_1(\boldsymbol{\gamma}, t) + (1-\beta_2)(\nabla L_i(\boldsymbol{\gamma}))^2$$

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \eta \frac{\hat{v}_1(\boldsymbol{\gamma}, t+1)}{\sqrt{\hat{v}_2(\boldsymbol{\gamma}, t+1)} + \boldsymbol{\varepsilon}}$$

with a learning rate η of 0.004 , and parameters β_1 and β_2 equal to 0.9 and 0.999 respectively. These parameters have been tuned by trial and error.
To train the model a number of fifty-five epochs has been chosen with a batch size equal to one-hundred-twenty-eight and a validation split of 20%. The number of epochs defines how many times the entire dataset

will be analysed by the neural network, while the batch size indicates the number of observations considered before updating the weights of the model (Goodfellow, Bengio, & Courville, 2016). The validation split indicates the number of observations used to test the model during training, it is expressed as a percentage of the training set. The difference between validation set and test set is that the test set is not considered by the model while training, so it gives more information about the generalization power of the model.
In lines of code:

```
#Defining neural network architecture
def NN_model():
    model = keras.Sequential([
    layers.Dense(64, activation='linear', input_shape=[len(train_ds.keys())]),
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1, activation='relu')])

    optimizer = tf.keras.optimizers.Adam(learning_rate=0.004)

    model.compile(loss='mae', optimizer=optimizer, metrics=['mae'])
    return model
model=NN_model()

#training the model using training set

tf.debugging.set_log_device_placement(True)

with tf.device('/device:GPU:0'):
       history = model.fit(normed_train_data, train_labels, batch_size=128,
       epochs=55, validation_split = 0.2, verbose=0)

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

#evaluating the model using test set and counting time to compute

start = time.time()
loss, mae = model.evaluate(normed_test_data, test_labels, verbose=2)
end = time.time()
```

The results of the above implementation are discussed in the following paragraphs.

Results

The Black-Scholes model didn't perform as well as expected in terms of error, with a mean absolute error of 1.557018. The model predicts values that are, in general, slightly above the average between bid and ask price.

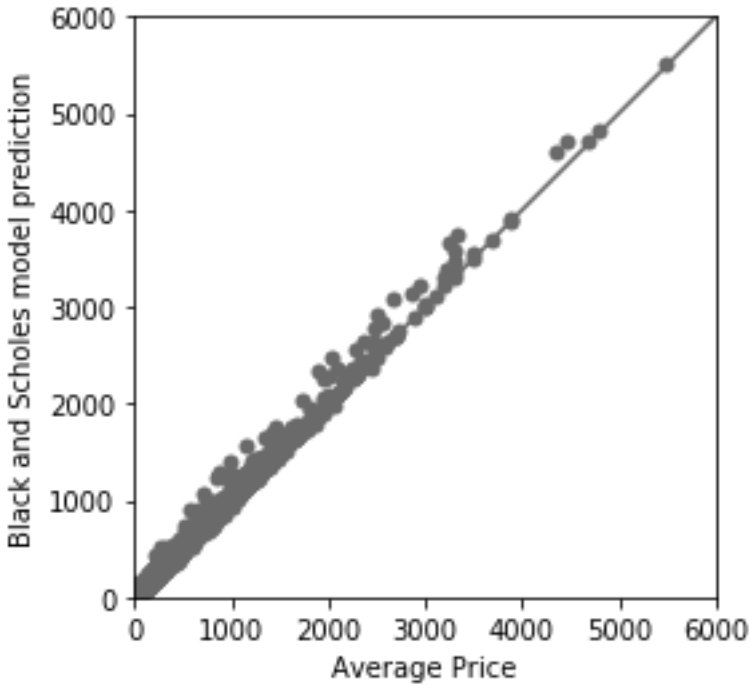

Figure 12: *representation of actual prices vs. Black-Scholes prices. The predictions are biased upward.*

As regards the temporal aspect, it performed fairly well with a time to price all the 185'735 test observations equal to 0.116884 seconds[23].

As regards the neural network model, it performed better in terms of MAE, but worse in terms of time to compute. The learning curve showed a fast convergence:

[23] This computation has been performed on an Intel Core i7-8750H CPU.

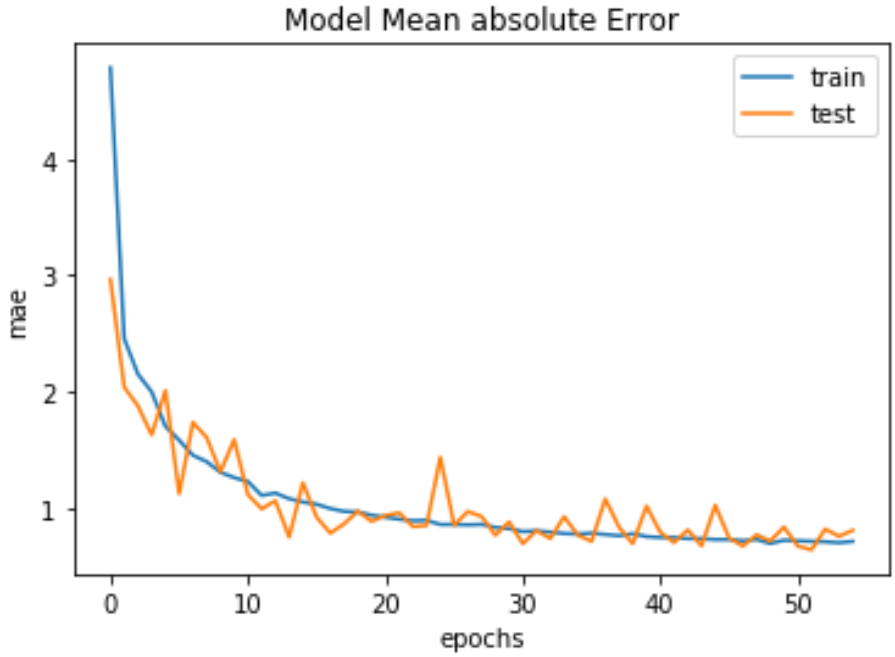

Figure 13: *a graph of the learning curve of the deep learning model, showing the MAE per Epoch.*

The test set's MAE is approximately near the train set's MAE, indicating a good generalization power of the model and the final test set's MAE is 0.802675, which is lower than the one achieved by the Black-Scholes model. Although, the time to evaluate the model is 5.666841 seconds, significantly higher than the time scored by the Black-Scholes model[24]. Furthermore, the deep learning model took nine minutes and fifteen seconds to be trained[25]. The neural network built showed a tendency to under-valuate prices, but with less difference in mean absolute terms than Black-Scholes method.

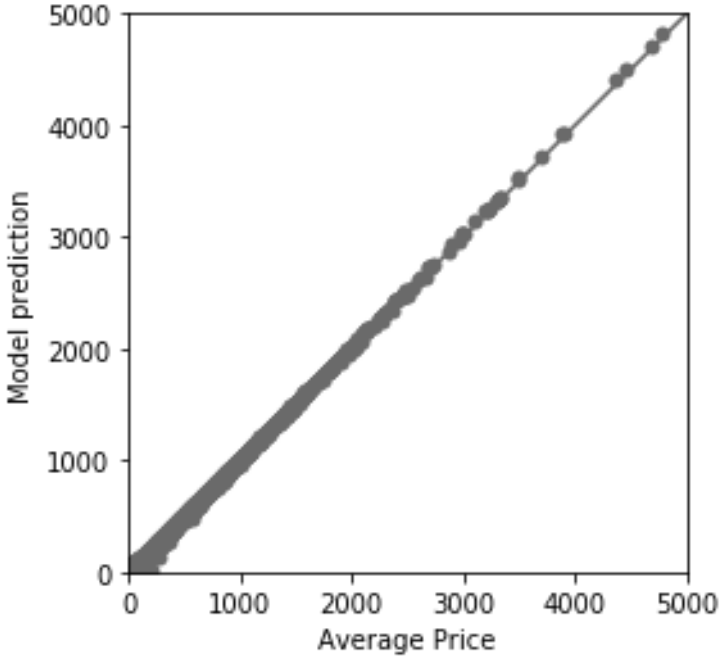

Figure 14: *representation of actual prices vs. neural network prices. The predictions are biased downward.*

[24] The evaluation has been performed on the same Intel Core i7-8750H CPU.
[25] This computation was performed in parallel on an NVIDIA GeForge GTX 1050 Ti GPU and an Intel Core i7-8750H CPU.

In general, given the same features for both models, the neural network model performed better than Black-Scholes at the expenses of time to compute, which is approximately fifty-six times higher for the neural network. However, the result achieved by the deep learning model has no underlying assumptions and could work with different data without the necessity to explicitly make hypothesis about volatility, arbitrage opportunities and stock prices movements.

The result is anyway quite surprising, because the deep learning model was not only able to approximate the Black-Scholes function, but also outperform it, using the same exact variables.

In practical field the use of a more complex deep learning model, with a larger dataset for training and evaluation and a more powerful hardware component, could lead to an even better performance, both in terms of errors and time (both to compute and train). On the other hand, the Black-Scholes model has fixed formulas and, even improving the hardware component, it cannot show better results in terms of MAE.

It can be argued that the result of the neural network has been achieved at the expenses of the explicatory power of the model. That's because, without any assumption and without the knowledge of what happened inside the hidden layers while training, a decision in a bargaining environment, based on the result of the model, couldn't be fully explained.

In fact, the Black-Scholes formula is fully understood and universally shared in the finance field, while deep learning models offer only results and not full explanation of why those results were achieved.

Critical issues

As regards the dataset used, it has been collected using only twenty-two working days, which could be a short temporal interval to fully evaluate the performance of the models. Other market tendencies may have been raised in the period considered, which could have deviated the prices from their theoretical value and could have been captured in the dataset available, leading to a biased performance of the models.

As regards volatility, the comparison between the models have been done assuming that the IVMean described in previous chapter was representative of the average between bid and ask prices. This assumption not always holds and could have caused biases for both the models.

Another aspect is the risk-free rate considered. As stated before, the risk-free rate should have been different for each option, based on the market, the timing and the geographical position of the option. The simplification adopted could have caused deviations in the Black-Scholes model and the neural network architecture built should have been different to account for different risk-free rates (for instance one neuron should have been added in the input layer). This aspect can be considered for further improvements of this work.

Other criticisms regard the models compared. The neural network outperformed Black-Scholes model, but is not sure if, considering more complex models such as Heston model (Heston, 1993) or Hull-White model (Hull & White, 1990), this tendency would be confirmed or not. On the other side, using more complex deep learning models may also lead to better results.

Therefore, this work represents a tiny step in comparing mathematical models and deep learning approaches in option pricing, using the same features for both.

Chapter 5

Conclusions

Option pricing has been a prolific topic in literature. A variety of different models have been built to achieve the goal of a reasonable option pricing, involving mathematics, statistics and probability. The most influential model is the Black-Scholes model (Black & Scholes, 1973), which legitimized the activities of the Chicago Board Options Exchange and other options markets around the world (MacKenzie, 2006).

In the last decade, the rise of Big Data and machine learning led to the spread of new models, based on linear algebra and vector calculus, which have found a wide range of applications.

In this thesis, a deep learning model was built for option pricing and compared to the Black-Scholes model, using the same features: Underlying price, Strike price, Time to maturity, Risk-Free rate, Implied Volatility.

The feedforward neural network built outperformed the Black-Scholes model in terms of mean absolute error, but it took more time for computation.

The aim of this thesis was to make a step in the favour of use of deep learning models in the field of pricing in finance and the results achieved go in the desired direction.

Nevertheless, some critical issues are present, such as the temporal range of options considered in the dataset, the price label considered, the use of implied volatility, the simplification made about risk-free rate, and, finally, the poor hardware environment in which both model have been implemented.

Further improvements can solve these issues by using different labels for prices, more accurately chosen, different methods for approximating the volatility and more specific data, which includes different risk-free rates for different options.

Next step could be to compare deep learning models with more advanced mathematical models and see if the same result holds.

Practical implications of this result can influence trading and portfolio management. The deep learning model presented, in fact, can be used to price options and can be re-trained with new options traded every day, making it more precise over time. Also, this process can be automated and, perhaps, co-adjuvanted with a specific trading algorithm to improve performance and returns. The usefulness of the model presented is that it only requires the same features of the Black-Scholes model, which are information easy to acquire in practice.

Finally, is helpful to consider that in some cases, where a complete and understandable explanation is needed, using Black-Scholes or other mathematical methods for option pricing is advisable, because the “black-box” of deep learning models could negatively impact in those situations, or in cases where the model fails without explanation.

References

Abadi et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous system. *tensorflow.org*.

Amodei et al. (2016). Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. *Proceedings of Machine Learning Research Vol. 48*, 173-182.

Angelini et al. (2008). A neural network approach for credit risk evaluation. *Elsevier, Vol 48 (4)*, 735-755.

Bachelier, L. (1900a). Théorie de la spéculation . *Annales Scientifiques de l'École Normale Supérieure*.

Baruníkova, M., & Baruník, J. (2011). Neural networks as a semiparametric option pricing tool. *Bulletin of the Czech Econometric Society*.

Berner et al. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *https://openai.com/*.

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy 81 (3)*.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics 31 (3)*, 307–327.

Boyle, P. P. (1977). Options: A Monte Carlo Approach. *Journal of Financial Economics, 4 (3)*.

Carr, P., & Madan, D. (1999). Option valuation using the fast fourier transform. *Journal of computational finance 2 (4)*, 61-73.

Chai, T., & Draxler, R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? - Arguments against avoiding RMSE in the literature. *Geoscientific model development*.

Chance, D. (2008). A Synthesis of Binomial Option Pricing Models for Lognormally Distributed Assets . *Journal of Applied Finance, Vol. 18*.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. *Proceeding*, 160-167.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research 12*.

Figlewski, S. (1994). Forecasting Volatility Using Historical Data. *NYU Working Paper No. FIN-94-032*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. *MIT Press, http://www.deeplearningbook.org*.

Heaton, J., Polson, N., & Witte, J. (2018). Deep Learning in Finance. *Cornell University, arXiv:1602.06561*.

Heston, S. L. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies, 6 (2)*.

High, R. (2012). The era of cognitive systems: An inside look at IBM Watson and how it works. *IBM Corporation, Redbooks*.

Hinton, G. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation Vol. 18 (7)*.

Hinton, G. (s.d.). Overview of mini-batch gradient descent . *Neural Network for Machine Learning course - Coursera*.

Hull, J., & White, A. (1990). Pricing interest-rate derivative securities. *The Review of Financial Studies, Vol 3 (4)* , 573–592.

Hunter, J. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering, vol. 9, (3)*, 90-95.

Itō, K. (1951). On Stochastic Differential Equations. *Memoirs of the American Mathematical Society 4*, 1-51.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *ARS Journal. 30 (10)*, 947–954.

Kingma, D., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *International conference on Learning Representations*.

Kou, S. G. (2002). A jump-diffusion model for option pricing. *Management science 48 (8)*, 1086-1101.

Kristjanpolle, W., Fadic, A., & Minutolo, M. (2014). Volatility forecast using hybrid Neural Network models. *Elsevier Vol. 41*, 2437-2442.

Leisen, D., & Reimer, M. (2006). Binomial models for option valuation - examining and improving convergence. *Applied Mathematical Finance Vol. 3 (4)*.

Liu et al. (2011). A one-layer recurrent neural network for constrained pseudoconvex optimization and its application for dynamic portfolio optimization. *Elsevier, Vol 26*, 99-109.

MacKenzie, D. (2006). An Engine, Not a Camera: How Financial Models Shape Markets. *Cambridge, MIT Press*.

Markov, A. A. (1906). Extension of the law of large numbers to dependent quantities. *Izvestiia Fiz.-Matem. Obsch. Kazan Univ Vol. 2 (15)*, 135-156.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 51-56.

Merton, R. (1973). Theory of Rational Option Pricing,. *Bell Journal of Economics and Management Science 4 (1)*, 141-183.

Oliphant, T. (2006). A guide to NumPy. *USA: Trelgol*.

Poole, D., Mackworth, A., & Goebel, R. (1998). Computational Intelligence: A Logical Approach. *Oxford University Press*.

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics Vol. 22 (3)*.

Rumelhart, , D., Hinton, G., & Ronald, J. (1986). Learning representations by back-propagating errors. *Nature 323* , 533-536.

Şchiopu, D. (2010). Applying TwoStep cluster analysis for identifying bank customers' profile. *UniversităŃii Petrol – Gaze din Ploieşti, Vol. LXII*.

Scott, L. O. (1987). Option pricing when the variance changes randomly. *Journal of Financial and Quantitative analysis, 22 (4)*.

Sharpe, W. (1964). Capital Asset Prices: a Theory of Market Equilibrium Under Conditions of Risk,. *Journal of Finance 19 (3)*, 425-442.

Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 484–489.

Virtanen et al. (2019). SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*.

Voulodimos et al. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*.

Willmott, C., & Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Center for Climatic Research, Department of Geography, University of Delaware.*

Summary

Introduction

This thesis aims to compare the Black-Scholes model with a deep learning approach in the field of option pricing, using the same variables as inputs for both models: S, the underlying price; K, the strike price; T, the time to maturity; σ, the volatility; r, the risk free interest rate.

The first part of the thesis is dedicated to a review of the literature and to the explanation of the main concepts involving options, mathematical models for pricing and deep learning. In the second part the study is explained, giving attention on the structure of the dataset used, the implementation of the two models, the results and the critical issues found.

Basic concepts of options

A European option is a contract which gives the owner the right, but not the duty, to buy an underlying asset at a specified price and at specified date. The other part involved in the contract has the obligation to sell the underlying asset if the buyer exercises his right. Buyers are referred to as having long positions, sellers as having short positions. In the case the holder has the right to buy the underlying instrument, the option is referred to as Call option, in the opposite case it's called Put option. Considering a call option on a non-dividend-paying stock, its payoff, c, is determined as follow:

$$c \geq \max\left(S_0 - Ke^{-rT}, 0\right)$$

Similarly, the payoff, p, of a put option on the same underlying asset is given by:

$$p \geq \max\left(Ke^{-rT} - S_0, 0\right)$$

An important relationship between European puts and calls is called put-call parity. Considering a call and a put having the same exercise price and expiration, their payoffs are linked by the following equation:

$$c + Ke^{-rT} = p + S_0$$

Mathematical option pricing models

The history of mathematical option pricing models can be brought back to the work of Louis Bachelier (1900), *Théorie de la speculation*[26], in which is introduced the stochastic process called Brownian motion and its use for stock options pricing. Later, in 1973, starting from the assumption that stock prices follow a

[26] Bachelier, L. (1900a), Théorie de la spéculation, Annales Scientifiques de l'École Normale Supérieure, 3.

geometric Brownian motion, Fischer Black and Myron Scholes published their famous and influential paper *The Pricing of Options and Corporate Liabilities*[27], where a closed formula for European option pricing is given, solving a parabolic partial differential equation, with initial conditions based on continuous "Delta-hedging". In particular, in Black-Scholes model, the stock price, S, follows a Brownian motion in which the expected stochastic drift rate is μS, for some constant μ. For a small temporal interval, $\Delta t \to 0$, the variation of the stock price is:

$$dS = \mu S dt$$

Adding the component of volatility to the equation, it becomes:

$$dS = \mu S dt + \sigma S dz$$

Where dz has the following properties:

P1. dz is normally distributed;

P2. mean of $dz = 0$;

P3. variance of $dz = dt$.

Starting from this formulation, using Itō's Lemma for stochastic calculus (Itō, 1951), the Black-Scholes-Merton parabolic differential equation is obtained (presented without proof):

$$rC = \frac{\partial C}{\partial t} + \frac{\partial C}{\partial S} rS + \frac{1}{2}\frac{\partial^2 C}{\partial S^2}\sigma^2 S^2$$

To solve the equation the boundary conditions are:

$$C = \max(S - K, 0), \qquad \text{when } t = T$$

for a call and:

$$C = \max(K - S, 0), \qquad \text{when } t = T$$

for a put.

Under these conditions, it is possible to derive a closed form solution both for call, c, and put, p, options:

$$c = S_0 N(d_1) - K\, K e^{-r\tau} N(d_2)$$

$$p = K e^{-r\tau} N(-d_2) - S_0 N(-d_1)$$

Where $\tau = T - t$, $N(\cdot)$ is the cumulative probability distribution function of the standardized normal

[27] Black, F.; Scholes, M. (1973), The Pricing of Options and Corporate Liabilities, Journal of Political Economy, 81.

distribution, d_1 and d_2 are defined as:

$$d_1 = \frac{ln\,(S_0/K) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

$$d_1 = d_1 - \sigma\sqrt{\tau} = \frac{ln\,(S_0/K) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

It's important to notice how risk-preferences of individuals are not involved in the above formulas, hence they can't affect the solution.

As regards the assumptions of the model, they are listed below:

i. the stock price follows a Brownian motion with constant μ and σ;
ii. short selling is allowed;
iii. all instruments are perfectly divisible and infinitely available;
iv. there are no market frictions, such as transactions costs or taxes;
v. no dividends are paid during the life of the derivative;
vi. there are no arbitrage opportunities, neither type I nor type II;
vii. securities trading is in continuous time;
viii. the risk-free rate is constant and the same for all expiration dates.

In the first hypothesis, σ indicates the volatility, which is defined in this context as the variation of the price of a financial instrument over time and it is generally measured by standard deviation of log-returns. The assumption about volatility can be relaxed assuming that also volatility follows a Brownian motion, like in Hull, White (1987)[28] and Heston (1993)[29].

Deep learning models

As regards deep learning, the quintessential model is the feedforward neural network (Goodfellow, Bengio, & Courville, 2016). This is a model whose objective is to approximate some function $f(\cdot)$ by utilizing a map $y = \hat{f}(\cdot, \theta)$ and learning the value of the parameters θ that lead to the best approximation. This purpose is achieved using a structure called artificial neural network:

[28] Hull, John, White, Alan (1987), The Pricing of Options on Assets with Stochastic Volatilities, The Journal of Finance Vol. 42 (2)
[29] Heston, S. L. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The Review of Financial Studies, 6 (2).

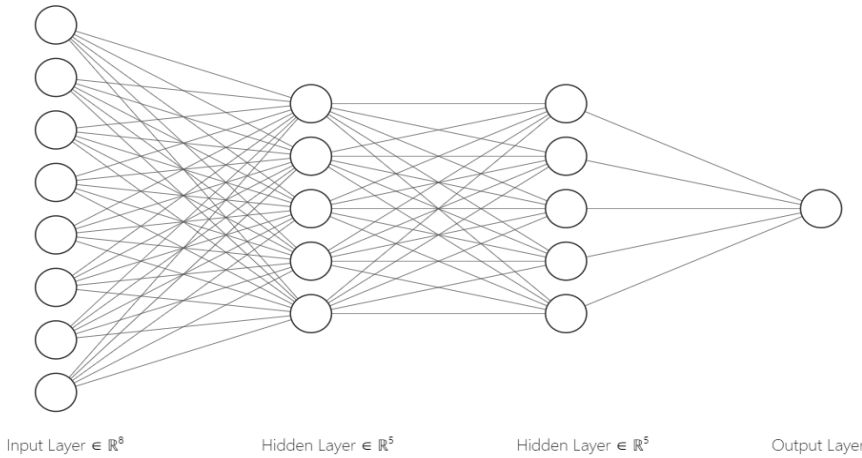

Figure 1: *example of the structure of a feedforward neural network. Source: created in http://alexlenail.me/*

In the above graphical example, circles are referred to as perceptrons (or simply neurons) and they take as inputs the value from their preceding neurons via synapses. These are connections between neurons and contains values called weights. The neurons are distributed in layers: the first is referred to as the input layer and in the example above has eight neurons, the two middle layers are called hidden layers and have five neurons each, the last layer is the output layer with one neuron. All layers in this case are fully connected. The neurons on the first layer takes the inputs, x_i, and transfer them to every neuron in the second layer with every connection weighted with weights w_i. In matrix form:

$$z = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + b$$

Where $\boldsymbol{x}$ and $\boldsymbol{w}$ are the vectors of input values and weights, b is called bias. To obtain the value for a generic neuron in the second layer, to z is applied a so-called activation function, which is generally non-linear and filters the value of z before passing it to the following layer. Thus, the value that reaches the third layer is:

$$h = f_{(1)}(z) = f_{(1)}(\boldsymbol{x}, \boldsymbol{w}, b)$$

After that, all values produced from the second layer, filtered and, again, weighted, arrive in a generic neuron of the third layer, whose value is given by some function $f_{(2)}(\boldsymbol{h}, \boldsymbol{\omega}, c)$ where $\boldsymbol{h}$ is the vector of values of the preceding layers, $\boldsymbol{\omega}$ is the new set of weights, c is the new bias. The value of the output layer is the composition of the functions described before:

$$g(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{b}) = f_{(2)}\big(f_{(1)}(\boldsymbol{x}, \boldsymbol{w}, b), \boldsymbol{\omega}, \boldsymbol{c}\big)$$

where $\boldsymbol{W}$ is the matrix obtained using as columns the vectors of $\boldsymbol{w}$ and $\boldsymbol{\omega}$.

Weights and biases in the feedforward neural network are initially chosen randomly, then they are optimized via backpropagation algorithm and a gradient-based optimization method, such as stochastic gradient descent. To perform the optimization process a loss function $L(\cdot)$ must be specified, which will be the objective function

to be minimized.

The backpropagation algorithm works with the following four main steps:

1. Set the output of the first activation function h^1;
2. For each layer $l = 2,3, \dots, \mathcal{L}$, compute z^l ,h^l and ε^l;
3. For each layer $l = \mathcal{L} - 1, \mathcal{L} - 2, \dots, 2$, compute ε^l relating it to its next layer;
4. The gradient of $L(\cdot)$ is given by the following equations for weights and biases:

$$\varepsilon_j^l = \frac{\partial L}{\partial b_j^l}$$

$$\varepsilon_j^l \cdot h_k^{l-1} = \frac{\partial L}{\partial w_{kj}^l}$$

Where ε^l is called the error of the j^{th} neuron in the layer l and $\varepsilon_j^l = \partial L / \partial z_j^l$ by definition.

The third step indicates why the algorithm described is called backpropagation.

Now, considering n observations in a dataset and an objective function $L(\boldsymbol{\gamma})$ to be minimized, with a set of parameters $\boldsymbol{\gamma}$ that minimize the function and which have to be estimated, the basic algorithm of gradient descent works as follow:

1. An initial vector $\boldsymbol{\gamma}$ is chosen and a learning rate η is defined**;**
2. for $i = 1,2, \dots, n$ do: $\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \eta \nabla L_i(\boldsymbol{\gamma})$[30]**;**
3. repeat until a minimum is reached.

A method called RMSProp (Root Mean Square Propagation), never published by Geoffrey Hinton in a formal academic paper, has become one of the most used methods in practice. In this case, the update (step 2 above) is made as follows:

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \frac{\boldsymbol{\eta}}{\sqrt{\boldsymbol{v}(\boldsymbol{\gamma}, \boldsymbol{t})}} \boldsymbol{\nabla L_i}(\boldsymbol{\gamma})$$

Where:

$$\boldsymbol{v}(\boldsymbol{\gamma}, \boldsymbol{t}) = \boldsymbol{\rho v}(\boldsymbol{\gamma}, \boldsymbol{t} - \mathbf{1}) + (\mathbf{1} - \boldsymbol{\rho})(\boldsymbol{\nabla L_i}(\boldsymbol{\gamma}))^{\mathbf{2}}$$

ρ is the so-called forgetting factor, t is the mini-batch size, $v(\boldsymbol{\gamma}, t)$ is called velocity.

In the neural network built in this thesis, Adam (Adaptive Moment Estimation) optimizer is used, which is an evolution of RMSProp, which uses velocity for both the first and second momentum of the gradient:

$$\boldsymbol{v_1}(\boldsymbol{\gamma}, \boldsymbol{t} + \mathbf{1}) = \boldsymbol{\beta_1 v_1}(\boldsymbol{\gamma}, \boldsymbol{t}) + (\mathbf{1} - \boldsymbol{\beta_1}) \boldsymbol{\nabla L_i}(\boldsymbol{\gamma})$$

[30] In this case and in the following examples, the symbol ≔ indicates an update of the value, it is not a "definition" or "equality" symbol.

$$\boldsymbol{v_2(\gamma, t+1) = \beta_2 v_1(\gamma, t) + (1-\beta_2)(\nabla L_i(\gamma))^2}$$

And the parameters are updated as follow:

$$\boldsymbol{\gamma} \coloneqq \boldsymbol{\gamma} - \eta \frac{\hat{v}_1(\boldsymbol{\gamma}, t+1)}{\sqrt{\hat{v}_2(\boldsymbol{\gamma}, t+1)} + \boldsymbol{\varepsilon}}$$

Where $\hat{v}_i$ are the estimators of velocities, $\boldsymbol{\varepsilon}$ is a small addend to avoid division by 0, β_i are the forgetting factors.

As regards activation functions, the ReLU function (Rectified Linear Unit) will be used:

$$f(x) = \begin{cases} 0 \; for \; x \leq 0 \\ x \; for \; x > 0 \end{cases}$$

It avoids negative values as output of the neuron and is useful in the problems that involves only positive values.

Finally, in the deep learning model presented, a loss function must be specified. Generally, the most used metric for this purpose is RMSE, although it is appropriate to represent model performance when the error distribution is expected to be Gaussian (Chai & Draxler, 2014). However, in this case this assumption does not hold. In fact, the distribution of errors in both models is expected to be leptokurtic.

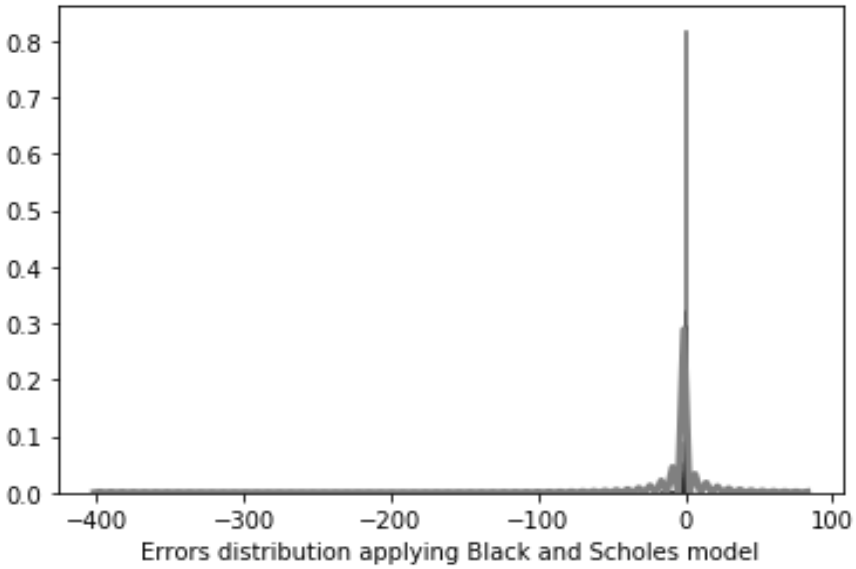

Figure 10: *histogram of the distribution of errors applying the Black and Scholes model to the test set.*

So, the deep learning model will be trained to minimize Mean Absolute Error as loss function:

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n}$$

Data preparation and implementation of the models

For the same reason stated before, MAE will be used as performance metric for both models. The second performance measure with which Black-Scholes model and the deep learning approach will be compared is the time necessary for the hardware to compute and apply the model to all test's observations. This measure is introduced to compare the models both in spatial and temporal aspect.

The dataset used to implement and compare the two models is the L3 Historical CSV Data Sample[31] which includes 928'674 observations of put and call European options traded on the Chicago Board Options Exchange. All data are recorded on 15th August 2019. The following features are included:

Variable	**Class**	**Description**
UnderlyingSymbol	Multinomial	One to four letters, identifying the country of the company in which the option is listed.
OptionSymbol	Multinomial	The ISIN code of the option, which identifies univocally the instrument.
Type	Binomial	It defines whether the option is a call or a put.
Expiration	Date	Settlement date of the contract.
DataDate	Date	It coincides with the observation date of the option, so it can be different from the starting date of the contract.
UnderlyingPrice	Numerical	The price of the underlying asset of the option.
Strike	Numerical	The strike price of the option.
Last	Numerical	The price at which the last trade has been done.
Bid	Numerical	The theoretical maximum price at which a buyer is willing to pay for the option.
Ask	Numerical	The theoretical minimum price at which a seller is willing to take for the option.
Volume	Numerical	The trading volume of the option.
OpenInterest	Numerical	The actual total number of outstanding options that have not been settled yet.
IVMean	Numerical	The average between the bid implied volatility of the option and the ask implied volatility.
IVBid	Numerical	The bid implied volatility.
IVAsk	Numerical	The ask implied volatility.
Delta	Numerical	The derivative of the option price with respect to its underlying price.
Gamma	Numerical	The sensibility of the option price with respect to its delta.
Theta	Numerical	The derivative of the option price with respect to its time to maturity.
Vega	Numerical	The derivative of the option price with respect to its implied volatility.

Table 1: *a list of all the features in the dataset, with an identification of the type of variable and a brief description of their meaning*.

As regards the risk-free rate, the US Tree Months Treasury Bill rate at the 15th August 2019 date has been chosen. The choice has been made because the average time to maturity of all the options in the dataset is 0.4028, which is close to 3 months, and the date is the date at which all options have been observed. The value of the rate discussed is 0.0187.

[31] Source: https://www.historicaloptiondata.com/

Next all the observations are divided in two sets, the training set and the test set. The division is made using 80% of the dataset as train set and 20% as test set, sampled randomly. At the end, the original dataset is split in 742'939 training samples and 185'735 test samples. Both models will be evaluated in the same test set.
Using NumPy, which enables to use arrays as inputs, the Black-Scholes formula can be implemented as a function $f(S_0, K, \tau, r, \sigma, Type)$, where, translating in dataset's columns, S_0 indicates the UnderlyingPrice, K indicates Strike, τ indicates TimeToMaturity, r the constant risk-free rate, σ the IVMean and Type indicates whether the option is call or put.
Second step is to define d_1 and d_2 as follow:

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

$$d_1 = \frac{\ln(S_0/K) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

Subsequently, using an if condition to separate call and put options, the formulas are applied:

$$c = S_0 N(d_1) - K\, Ke^{-r\tau} N(d_2)$$

$$p = Ke^{-r\tau} N(-d_2) - S_0 N(-d_1)$$

Then the model can be applied to the test set, simply applying $f(\cdot)$ to each row. Finally, MAE is calculated.
To implement the chosen feedforward neural network to evaluate the price of options, data normalization is needed and it is performed, using the mean and standard deviation for each column:

$$z = \frac{x - mean(x)}{st.dv.(x)}$$

Where x represents a vector of original data, z the vector of standardized data, while $mean(\cdot)$ and $st.dv.(\cdot)$ are the functions for mean and standard deviation respectively. After this process is applied the variable representing the risk-free rate disappears, leaving all zeros, therefore it's eliminated and the model will be trained using only UnderlyingPrice, Strike, IVMean, TimeToMaturity, Type_code (the code referring to the Type).
Then the model is built using six layers:

- One input layer with five neurons, one for each of the five features;
- Four hidden layers, each with sixty-four neurons and different activation functions;
- One output layer with one neuron, which gives the price of the option.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 64)                384
_________________________________________________________________
dense_16 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_17 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_18 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 65
=================================================================
Total params: 12,929
Trainable params: 12,929
Non-trainable params: 0
```

Table 2: *a summary of the layers of the neural network, without considering the input layer. The neural network will update multiple times its 12'929 parameters while training.*

As regards the hidden layers, the first one has "linear" activation functions, which leaves the linear combination of the input data that arrives to each neuron unchanged. The second, third and fourth hidden layers have the same activation function, which is the ReLU. For the output layer, ReLU has been chosen as well and its interpretation is plain to understand: it gives non-negative values, hence non-negative prices. As discussed before, the loss function used to be minimized is MAE and the optimizer chosen is Adam (Kingma & Ba, 2014) with a learning rate η of 0.004 , and parameters β_1 and β_2 equal to 0.9 and 0.999 respectively. These parameters have been tuned by trial and error.

To train the model a number of fifty-five epochs has been chosen with a batch size equal to one-hundred-twenty-eight and a validation split of 20%. The number of epochs defines how many times the entire dataset will be analysed by the neural network, while the batch size indicates the number of observations considered before updating the weights of the model (Goodfellow, Bengio, & Courville, 2016). The validation split indicates the number of observations used to test the model during training, it is expressed as a percentage of the training set. The difference between validation set and test set is that the test set is not considered by the model while training, so it gives information about generalization power of the model.

Results and critical issues

The Black-Scholes model didn't perform as well as expected in terms of error, with a MAE of 1.557018. The model predicts values that are, in general, slightly above the average between bid and ask price.

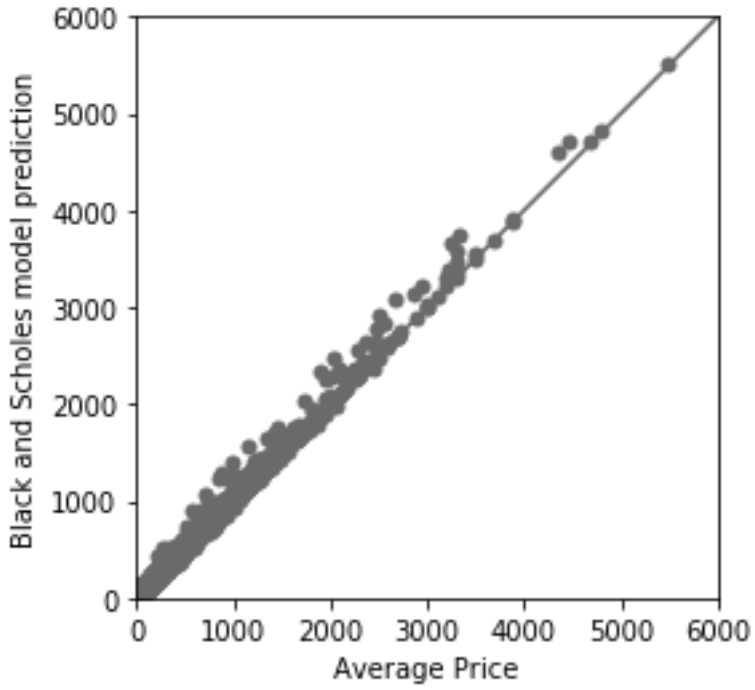

Figure 12: *representation of actual prices vs. Black-Scholes prices. The predictions are biased upward.*

As regards the temporal aspect, it performed fairly well with a time to price all the 185'735 test observations equal to 0.116884 seconds[32].

As regards the neural network model, it performed better in terms of MAE, but worse in terms of time to compute. The learning curve showed a fast convergence:

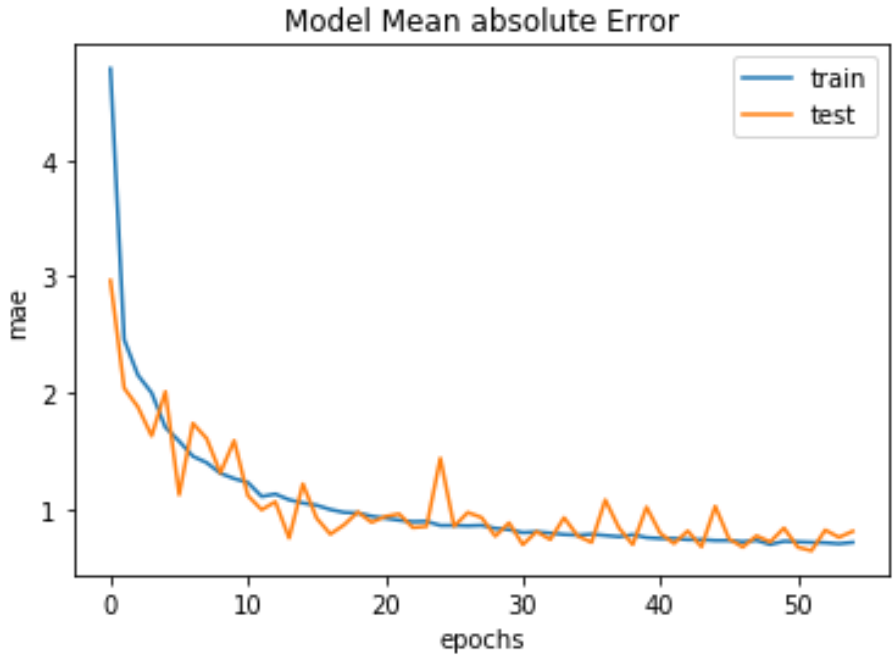

Figure 13: *a graph of the learning curve of the deep learning model, showing the MAE per Epoch.*

The test set's MAE is approximately near the train set's MAE, indicating a good generalization power of the model and the final test set's MAE is 0.802675, which is lower than the one achieved by the Black-Scholes

[32] This computation has been performed on an Intel Core i7-8750H CPU.

model. Although, the time to evaluate the model is 5.666841 seconds, significantly higher than the time scored by the Black-Scholes model[33]. Furthermore, the deep learning model took nine minutes and fifteen seconds to be trained[34].

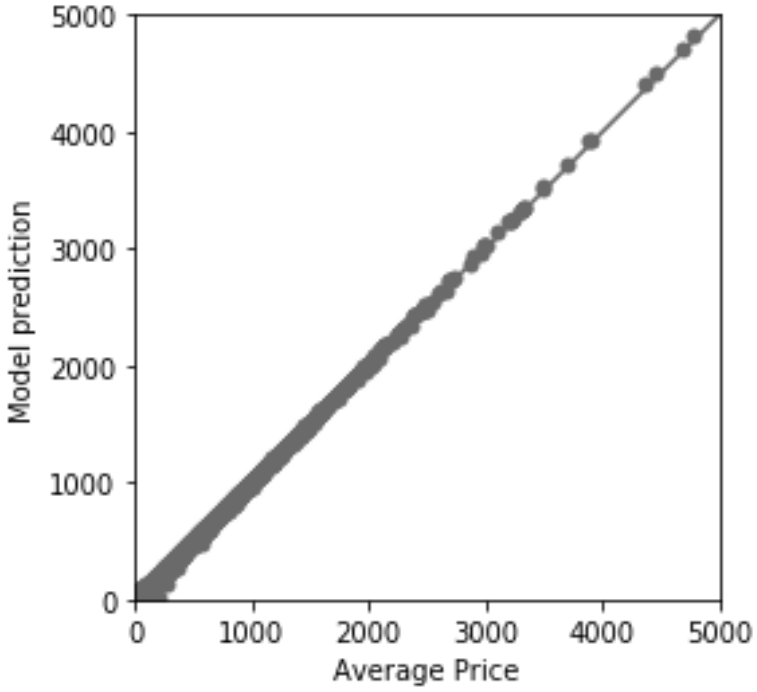

Figure 14: *representation of actual prices vs. neural network prices. The predictions are biased downward.*

In general, given the same features for both models, the neural network model performed better than Black-Scholes at the expenses of time to compute, which is approximately fifty-six times higher for the neural network. However, the result achieved by the deep learning model has no underlying assumptions and could work with different data without the necessity to explicitly make hypothesis about volatility, arbitrage opportunities and stock prices movements.

The result is quite surprising, because the deep learning model was not only able to approximate the Black-Scholes function, as intended at the beginning, but also outperform it, using the same exact variables.

It can be argued that the result of the neural network has been achieved at the expenses of the explicatory power of the model. That's because, without any assumption and without the knowledge of what happened inside the hidden layers while training, a decision in a bargaining environment, based on the result of the model, couldn't be fully explained.

In fact, the Black-Scholes formula is fully understood and universally shared in the finance field, while deep learning models offer only results and not full explanation of why those results were achieved.

Some critical issues can be brought about the dataset used, which has been collected using only twenty-two working days, that could be a short temporal interval (even though 928'674 observations are considered).

As regards volatility, the comparison between the models have been done assuming that the IVMean described

[33] The evaluation has been performed on the same Intel Core i7-8750H CPU.

[34] This computation was performed in parallel on an NVIDIA GeForce GTX 1050 Ti GPU and an Intel Core i7-8750H CPU.

in previous chapter was representative of the average between bid and ask prices. This assumption not always holds and could have caused biases for both the models.

Another aspect is the risk-free rate considered, the same for all options in the dataset. The risk-free rate should have been different for each option, based on the market, the timing and the geographical position of the option. The simplification adopted could have caused deviations in the Black-Scholes model and the neural network architecture built should have been different to account for different risk-free rates (for instance one neuron should have been added in the input layer). This aspect can be considered for further improvements of this work.

This work represents a tiny step in comparing mathematical models and deep learning approaches in option pricing, using the same features for both.

Conclusions

Option pricing has been a prolific topic in literature. The most influential model is the Black-Scholes model (Black & Scholes, 1973), but, in the last decade, the rise of Big Data and machine learning led to the spread of new models, which have found a wide range of applications. In this thesis, a deep learning model was built for option pricing and compared to the Black-Scholes model, using the same features for both.

The feedforward neural network built outperformed the Black-Scholes model in terms of mean absolute error, but it took more time for computation. The aim of this thesis was to make a step in the favour of use of deep learning models in the field of pricing in finance and the results achieved go in the desired direction.

Nevertheless, some critical issues are present, such as the temporal range of options considered in the dataset, the price label considered, the use of implied volatility, the simplification made about risk-free rate, and, finally, the poor hardware environment in which both model have been implemented. Further improvements can solve these issues by using different labels for prices, more accurately chosen, different methods for approximating the volatility and more specific data, which includes different risk-free rates for different options.

Practical implications of this result can influence trading and portfolio management. The deep learning model presented, in fact, can be used to price options and can be re-trained with new options traded every day, making it more precise over time. The usefulness of the model presented is that it only requires the same features of the Black-Scholes model, which are information easy to acquire in practice.