Ca' Foscari
University
of Venice

Master's Degree
in Economics and Finance

Final Thesis

Neural Network Models for Option Pricing

Supervisor
Ch. Prof. Antonella Basso

Graduand
Loris Simeoni
Matriculation Number 863724

Academic Year
2021/2022

To my family

Abstract

There exist several different ways to evaluate financial derivatives but, in general, closed-form formulas, such as Black and Scholes, tend to provide unsatisfactory results. Therefore, nowadays, thanks to the increased computational capability of machines numerical methods are commonly used.

The aim of this dissertation is to develop a nonparametric supervised machine learning method, namely a Multilayer Perceptron Feedforward Artificial Neural Network, to price financial options written on the FTSE MIB index. It means we try to implement a data-driven approach which, by exploiting the architectural structure of a multi-level neural network, is able to correctly identify the value of the analyzed derivative. In particular, the function used to train the algorithm is the Levenberg-Marquart backpropagation and the performance is evaluated by relying on the Root Mean Square Error (RMSE).

Contents

List of Tables

List of Figures

Introduction

Since the publication of the famous Black and Scholes model in 1973, there has been an incredible growth in the research and the trading activities regarding financial options. This development was further supported by favorable policies implemented starting from the early 1980s by financial regulators all over the world. Thanks to the rapid succession of these positive conditions, the increase of the importance of this kind of contracts has been so strong that nowadays options are among the most popular components in the portfolios of financial institutions.

Of course, the increasing importance of a financial asset always translates into a series of positive effects; indeed, the efficiency of the market has increased over time, reducing transaction costs, attracting more capital, and partially removing the existing asymmetries. However, it poses also new challenges for all the market participants; for instance, following the subprime crisis, the need of reforming the regulatory structure became evident. Moreover, larger volumes imply a stronger demand for correct pricing models able to provide arbitrage-free estimates. Therefore, we should not be surprised in knowing that following the publication of the Black and Scholes equation, many refined and more sophisticated pricing systems have been developed.

The problem that all these systems have in common is the fact that they are closed-form formulas, meaning they try to present a formalized function describing how much the analyzed options should cost. Of course, the ability to draw up a correct estimate depends on how capable the proposed model is in understanding the existing relationship between the explanatory variables and the dependent output. Moreover, the overall performance is determined by the necessary assumptions of each model. Indeed, simplifying assumptions are always required by deterministic formulas in order to be able to deal with any kind of real-life problem. This is because it is way too complex to try to account for every possible existing variable. However, this approach always determines a huge drawback, i.e., by doing so we are introducing biases, abstractions, and incoherencies in our model, which inevitably make it less precise and distant from a real-world scenario. For instance, many studies have shown that even if the Black and Scholes equation is still able to overperform many other pricing models, it shows a recurrent mispricing when it faces deep-out-of-the-money and deep-in-the-money options, see for instance Yao et al. (2000). Therefore, many alternatives have been proposed trying to smooth the magnitude of these biases by reducing, or even completely removing, the underlying assumptions.

In this sense, an interesting family of alternative approaches, which has now become the state-of-the-art, consists in relying on artificial intelligence and machine learning algorithms. It should be noted that, in the thesis, we chose to address the option pricing task by developing a so-called Artificial Neural Network.

Artificial Neural Networks are data-driven algorithms which try to mimic the functioning of a human brain in order to solve complex problems. In particular, they are extremely good instruments in dealing with high dimensional and complex dataset, since they are capable of detecting patterns and nonlinear connections existing between the inputs and the outputs. However, these algorithms are so good in understanding the structure of the data that quite often there is the risk of "overfitting", meaning the model fits the analyzed observations too well and it is not able to

generalize the results. With this regard, we need to say that we will present useful techniques which help preventing such kind of behaviour. A second "problem" about neural networks is the fact that they are "expensive algorithms" from a computational point of view. Indeed, they require large amount of data to be properly trained, and computers with strong enough softwares that make the required computations feasible both from a time perspective point of view and in terms of use of resources. Consequently, their exploitation was not convenient in the past, mainly due to technical limitations. However, the increasing data availability and the improved performances of computers make it possible to use these tools today in an efficient way. Therefore, since they tend to outperform traditional pricing methods, it is clear why neural networks are nowadays implemented in many different applications such as regression and classification problems.

Having said that, we can now state the objective of the thesis. First of all, we need to highlight that we have a twofold objective which is based on the following hypotheses:

Hypothesis 1. It is possible to develop a multilayer perceptron artificial neural network able to correctly price European call options written on the FTSE MIB index.

Hypothesis 2. Option prices generated by such a model overperform those provided by a traditional pricing formula, i.e., the Black-Scholes-Merton model.

With this respect, we are extremely proud to underline the fact that the model we are going to build is applied to the Italian equity market, meaning that this thesis does something that has never been done before. Indeed, at the time of writing, there exists no paper in which an artificial neural network is applied to the pricing of financial options with the FTSE MIB index as the underlying asset.

In performing this task, we relied on the version R2021b of MATLAB, and we exploited some useful packages which helped us to develop our artificial neural network, namely the Machine Learning Toolbox and the Deep Learning Toolbox.

According to the specific structure followed by the thesis, Chapter 1 introduces the concept of financial options, their features and why it is essential to develop a model able to provide reliable estimates. In doing so, it presents the Black and Scholes formula, its most popular extensions and its most serious drawbacks, explaining at the same time what makes the development of alternative pricing models necessary.

Chapter 2, instead, is focused on machine learning, the theory behind it and the most popular existing approaches. In particular, the Chapter pays a lot of attention to neural networks, their development both from an historical and mathematical point of view, and their main properties.

Then, in the first part of Chapter 3, the Italian equity and option markets are presented, whereas in the second part of the Chapter the structure of the available dataset is analyzed. Moreover, the operations performed on the data, such as the removal of outliers or the applied regularization techniques, are described and deeply explained.

Finally, Chapter 4 shows the hyperparameter optimization procedure that we implemented in developing the best possible architecture for our artificial neural network, and the obtained results are presented. In particular, we focused on the comparison between the forecasts of our model and the ones provided by the BSM model.

Chapter 1 Option pricing

Chapter 1 presents an overview of financial options and the most famous model for pricing such instruments. In particular, the first paragraph provides a definition of these contracts, their main characteristics and the reason why we need efficient pricing techniques. Then, section 1.2 introduces the features and the necessary conditions that each pricing method must respect in order to be coherent and to not allow for arbitrage opportunities. These are derived from the dominance principle, and if a model violates them it would necessarily mean that it is inefficient, and it is possible to detect arbitrages. After that, the third paragraph is focused on deriving the Black and Scholes equation in a formal way, and it concludes by presenting the mathematical formulation found in the original paper of 1973. In section 1.4, the most popular extensions of the original model are presented. These have been developed with the scope of trying to generalize the provided results by relaxing one or more of the founding assumptions of the Black and Scholes formulation. Finally, the last section of the Chapter is about the most serious drawbacks and biases of the former model, and it introduces the reasons why it is necessary to develop new pricing techniques capable of overcoming the problems that can be found in the BS model.

1.1 Options

Contracts similar to options have been used since ancient times. Initially created as a hedging tool for agricultural products, they quickly developed into more complex instruments which may serve a lot of different purposes. However, even if options contracts had been known for decades, it was only in 1973, with the establishment of the Chicago Board Options Exchange (CBOE), that standardized contracts were defined and a guaranteed clearing house was created. Moreover, in the same year, Fischer Black and Myron Scholes published a fundamental paper which became a milestone on the derivatives literature[1].

In the article the two researchers presented a closed-form formula for the pricing of European-style derivatives, which was obtained through a dynamic hedging argument and a no-arbitrage condition. From that moment on, both trading activity and academic research have exponentially increased to the point that nowadays it would be virtually impossible to provide an exhaustive review of the existing literature.

[1] "The Pricing of Options and Corporate Liabilities" by Black, F., and Scholes, M. (1973) The Journal of Political Economy, Vol. 81, pp. 637-654.

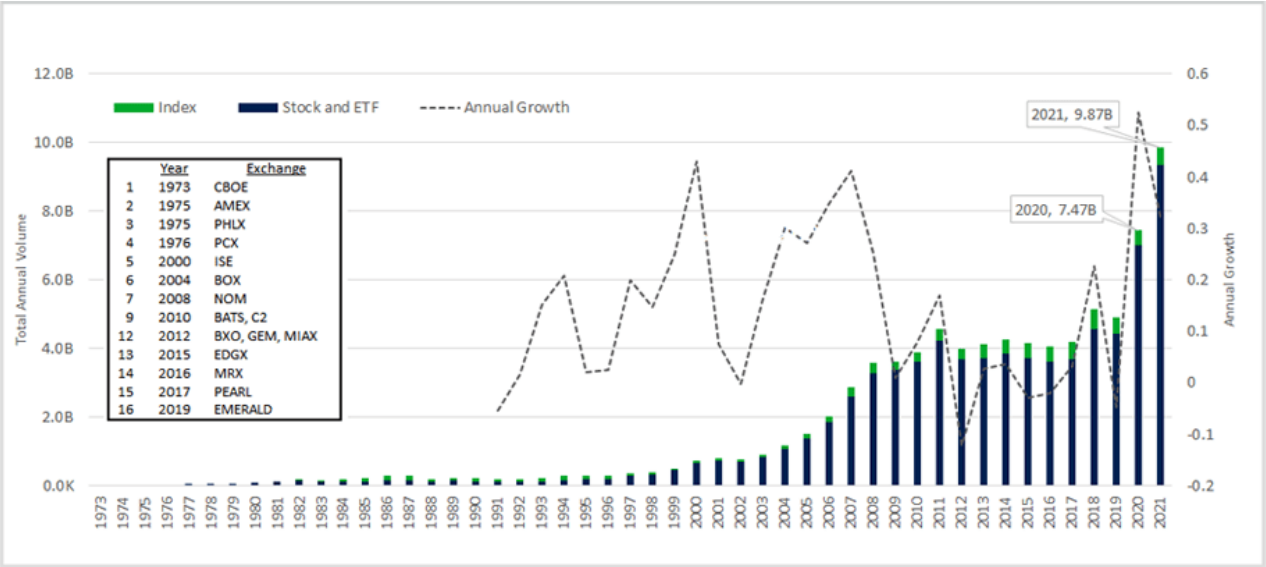

Figure 1 - Annual option volume and annual growth rate (source: SeekingAlpha.com)[2]

As we can see from Figure 1, starting from 1973 option growth has never stopped; on the contrary, it has gained speed in the last years. Just to have an idea, during 2021 a record on the daily total option volume was set 17 times and, overall, about 9.87 billion contracts were traded, meaning a rise of more than 30% when compared with 2020's maximum. Moreover, according to CBOE Global Markets data, the average daily notional value[3] of traded options has risen to more than $450 billion[4]. Therefore, it should be clear that nowadays options are an extremely important part of financial markets, and efficient pricing techniques which allow investors to rapidly evaluate such instruments are essential.

At this point a question naturally arises: what is an option? It should be clear that an option is a flexible contract between two counterparts[5]. Indeed, an option represents the right, not the obligation, for the buyer to buy or sell the underlying asset at, or within, a future point in time for a predetermined price. We have said that an option is not an obligation, however this is true only for the buyer of the contract. The seller of the derivative instrument, indeed, has a contractual, legally binding commitment to sell or to buy the underlying if the buyer chooses to exercise it. Therefore, we can say options are a sort of asymmetric contract, since the right of the buyer is opposed to the obligation of the seller.

Let us clarify the terminology. Being a derivative, an option is an instrument whose value depends on the value of another instrument, which is known as underlying asset. The underlying asset can be of any kind, both financial and real. In the first case, it may be a stock, an index, an interest rate and so on, even another derivative. In the second case, instead, it could be, for example, a course of actions to be taken or to be avoided.

[2] "Options Clearing Corp and Cboe LiveVol, LLC, 12/31/21" from SeekingAlpha.com.
[3] The notional value is computed as the trading volume multiplied by the spot price.
[4] In the same period, stocks were about $405 billion.
[5] On this regard it is possible to consider Hull (2008) or any other book about options.

A primary distinction which can be made regards the type of right the owner of the option is entitled with. In particular, if the buyer has the right to buy the underlying asset, the option is a call option, whereas if the owner has the right to sell the underlying asset to the writer of the option, i.e., the seller, the option is a put option.

Additional distinctions can be made according to the timing of the exercise and the way in which the price is determined.

With respect to the exercise of the contract, namely the style of the option[6], a common but not exhaustive classification is made between European and American options. The difference is that European-style options give the buyer the right to exercise the contract on the expiration date, also called maturity date; it means that the option can be exercised only on the last day of its life. American-style options instead are more flexible, in the sense that they provide a stronger right to the owner of the contract which can, in this case, exercise the option anytime up to the expiration date[7].

Note that the style of the contract is mainly determined by market conventions and operational decisions, which may change from one provider to another; however, since it has a great importance on the pricing of the option, it is crucial for anybody interested in dealing with these instruments to be well aware of the type of contract under consideration.

In regard to how the delivery price is determined, there exist two main approaches which are now the state of the art in the industry. The simplest and most common one for exchange-traded options (ETOs)[8] is to use a fixed, predetermined price, called strike price[9]. However, a quite popular alternative concerns the so-called Asian options and consists in determining the payoff of the contract by comparing the strike price of the option with the average price of the underlying asset computed over a specific period of time, usually the life of the option itself.

These are the main features of a standard, "*vanilla*", option. However, there are many possible alternatives that someone interested in option contracts can come in touch with. In general, options whose payoff and characteristics are different from the ones above mentioned are called "exotic options" and are usually traded over the counter (OTC). They are usually case-specific, meaning that they have peculiar attributes which vary a lot from case to case, and therefore they often embed a significant counterparty risk, a low-price transparency and a lack of data availability[10].

To conclude, we need to explain the meaning of the following terms: in-the-money (ITM), out-of-the-money (OTM), and at-the-money (ATM)[11]. An option is said to be ITM when it has a positive value for its owner. In case of a call option this happens when the current price of the underlying is

[6] The style of an option is sometimes also called "family".

[7] Other less common exercise styles in which the payoff remains the same, but the early exercise may differ are: Bermudan options, Canary options, Capped-style options, Compound options, Shout options, Double options, Swing options, Evergreen options, but it is possible to consider also exotic options with a standard exercise style, like for instance Composite options.

[8] T. Hann: "Option pricing using artificial networks: an Australian perspective", Doctoral thesis (2014).

[9] Other common names are exercise price or contract price.

[10] Note that exotic options can differ not only on the above-mentioned characteristics but also in others such as the lot size, the way in which the underlying is transfer, etc.

[11] It is also possible to add the term "deep" to ITM and OTM options to point out that the strike price is far away from the current price.

above the strike price. For put options, instead, the contract is in-the-money when the current price lies below the strike. OTM options are the opposite case and thus represent a "negative" value for the holder of the contract. Finally, at-the-money options are "neutral", in the sense that they neither provide a profit nor a loss to the option buyer. Therefore, ATM is a term used to refer to options whose strike price is the same as the current price of the underlying[12]. Note that, moneyness is another term used to describe whether a contract is ITM, OTM, or ATM.

1.2 Introduction to European option pricing

Having observed the constantly increasing importance of this kind of instrument in current financial markets, it should be clear the need to develop sophisticated and accurate pricing formulas able to provide reliable estimates, in a time efficient way, regarding option prices. As already mentioned, in 1973 Black and Scholes published a paper presenting an analytical solution for the problem of pricing European-style options. The proposed model rapidly became the foundation of any further research in the field of derivatives evaluation and hedging, and therefore must be correctly explained and understood.

In deriving their model, Black and Scholes relied on the following assumptions:

- There are no penalties for short sales; therefore, it is possible to buy and sell any amount of the asset, even fractional, without any penalizations in terms of market impact, broker restrictions, etc.
- Transaction costs and taxes are excluded from the model computations, meaning we are considering a frictionless market.
- The market operates continuously, there are no interruptions in the trading activity.
- The risk-free interest rate is assumed to be constant over time. Moreover, it is possible to borrow and lend any amount of cash at the riskless rate.
- The stock price is continuous, therefore there are no "jumps" in its trajectory[13].
- The stock does not pay any dividends.
- The considered options are European and can only be exercised on the expiration date.

An important note can be made about these assumptions. In particular, some of them are not strictly necessary and can easily be removed without any loss of generality, while others are more cumbersome to deal with, and impose stronger limitations to the model. They can still be relaxed, but the effects on the pricing formula are more complex and affect the results in deeper ways.

What is truly amazing about the Black and Scholes model is its simplicity combined with its robustness; indeed, Black and Scholes were able to provide a closed-form solution of the problem as a function of only five variables:

[12] Coherently with these definitions, in-the-money options are the only one having an intrinsic value for the holder of the contract, whereas ATM and OTM prices are determined only by the time value of the option.

[13] Roughly speaking, if we consider the time series of the price of a stock, it can be drawn without ever lifting the pen.

- The stock price,
- The strike price,
- The variability of the stock price, expressed as the variance of the price returns,
- The time to maturity of the option, i.e., the life of the contract,
- The risk-free interest rate.

At this point, two important observations can be made. First of all, the solution does not require variables such as the expected return of the stock, or any kind of "risk aversion" parameter[14]. Secondly, the only variable of the solution which is not directly observable is the variance, however it can be estimated by relying on the time series of past prices[15].

Since its first publication the model has been extremely popular, and many adjustments and refinements have been proposed. Moreover, many researchers have tested the goodness and the reliability of the model by relaxing the previous assumptions and have found it to be quite robust, to the point that *"no single assumption seems crucial to the analysis"*, Smith (1976).

Among all the various publications that followed the article, one of the most important was the one provided by Robert Cox Merton (1974). Merton was able to generalize the model to the case of stochastic interest rates and he was also able to develop an adjustment that allows the model to deal with dividends-paying stocks. Furthermore, Merton (1976) and Cox and Ross (1976) were successful in generalizing the Black and Scholes formula to the case in which the stock price movements are discontinuous. They also showed that the solution found was appropriate to evaluate American-style call options.

Before considering the Black-Scholes-Merton model it is necessary to provide some preliminary concepts which define a set of equilibrium conditions on the price of a call option. These were derived by Merton and are independent of distributional assumptions. What does it mean? It simply means that Merton made no previous assumptions on the process generating the data, i.e., the stock prices; therefore, these restrictions simply followed by the concept of dominance and represent a no-arbitrage condition.

The idea is that a rational investor has always to prefer more over less. This translates into the fact that if the return of portfolio A is, over a certain period of time, always[16] greater or equal than the return of portfolio B[17] and they have the same risk, portfolio A is dominant over portfolio B. The problem is that in equilibrium there cannot exist such a situation, otherwise it would be possible to detect an arbitrage opportunity. In our example, this means that everybody would buy portfolio A and sell portfolio B. By doing so, the price of the first portfolio would increase, while the one of the second would decrease, therefore reducing the existing spread, up to the point that the dominance relationship ceases to exist.

[14] With risk aversion parameter I want to denote any variable able to represent the investor attitudes toward risk.
[15] Clifford W. Smith Jr, Option pricing: a review. Journal of Financial Economics (1976).
[16] Always in this case should be read as "in every possible state of the world".
[17] It has to be at least one time higher than the other and never smaller.

Note that this is a general result known as no-arbitrage condition and must be verified in every time instant in order to ensure the efficiency of any economic model.

From the concept of dominance and thus from the no-arbitrage condition it is possible to define a series of boundaries that have to be met, assuming the model being efficient.

At this point it should be clear that the exercise of an option is a voluntary action taken by the owner of the contract. Since it will be pursued only when in the best interests of the option buyer, it defines a first lower bound both for European and American options:

$$c(S, T, X) \geq 0$$

$$C(S, T, X) \geq 0$$

Therefore, call prices cannot be negative.

It should be mentioned that we denote with the capital letter, *C*, the American call option and with the small letter, *c*, the European one. *S* represents the price of the stock at time *t*, *T* is the time to expiration, and it is computed as the difference between the maturity date, $t^{expiration}$, and the current date, *t*, and *X* is the strike price of the call option[18].

Then it is possible to derive the value of the call at maturity. At the expiration, the call will be equal to the maximum of either the difference between the stock price and the exercised price or zero.

$$c(S^*, 0, X) = \max(0, S^* - X)$$

$$C(S^*, 0, X) = \max(0, S^* - X)$$

It is interesting to notice that, in this case, at maturity $T = 0$, since the expiration date and the current date are the same.

For the American case it is also possible to define some additional conditions. Indeed, given the fact that the exercise can be made at any point up to the maturity date, the value of the American call should always be at least equal to:

$$C(S, T, X) \geq \max(0, S - T)$$

Moreover, if we consider two American call options which only differ for the expiration date, the one with the longest time to maturity could never be worth less than the one with the closest expiration:

$$C(S, T_{long}, X) \geq C(S, T_{short}, X)$$

Considering all has been said so far, we should not be surprised in knowing that an American call option could never cost less than the corresponding European option:

$$C(S, T, X) \geq c(S, T, X)$$

Another condition we derive from the no-arbitrage principle is that if two call options are identical except for the strike price, the one with the lower exercise price must value no less than the other.

[18] The notation comes from "Option pricing: a review" by Clifford W. Smith Jr, The Journal of Financial Economics, Vol. 3, pp. 3-51 (1976).

$$C(S,T,X_{low}) \geq C\left(S,T,X_{high}\right)$$

$$c(S,T,X_{low}) \geq c\left(S,T,X_{high}\right)$$

Now we can ask ourselves: what is the relationship existing between the underlying stock value and different calls written on it? If we consider a perpetual call, meaning a call in which the expiration date tends to plus infinite, with a zero-strike price we get:

$$S \geq C(S,\infty,0)$$

The value of a non-dividend paying stock today is at least equal to the value of a call option with an infinitely long maturity and a zero-strike price. Then, from the previous conditions it is also possible to generalize the concept by adding the finite case:

$$S \geq C(S,\infty,0) \geq C(S,T,X)$$

Remember we are considering stocks that do not pay any dividends; if that was not the case the stock value may exceed the value of the perpetual call.

From this relationship immediately follows that a call option on a worth nothing stock, i.e., a stock with a current price equal to zero, must have a zero-value:

$$C(0,T,X) = c(0,T,X) = 0$$

Another interesting result derived by the stochastic dominance principle is that the early exercise, i.e., the exercise before the maturity, of an American option on a non-dividend paying stock is never optimal. To derive this result, we first need to define with $B(\tau)$ the price of a risk-free zero-coupon bond that pays one dollar in τ years. Assuming positive interest rates it is straightforward to understand that bonds of this kind characterized by longer maturities will be valued less than similar bonds with shorter maturities.

Now, suppose we built two different portfolios C and D:

- C: buy one European call $c(S,T,X)$ and buy X bonds for $XB(T)$;
- D: buy the stock, S.

		$Stock\ price\ at\ T=0$	
$Portfolio$	$Current\ value$	$S^{exp} < X$	$S^{exp} \geq X$
C	$c(S,T,X) + XB(T)$	$0 + X$	$(S^{exp} - X) + X$
D	$S(t)$	S^{exp}	S^{exp}
$Final\ value$		$\Pi_C > \Pi_D$	$\Pi_C = \Pi_D$

Table 1 – Proof that the value of a portfolio made of a call and a riskless bond with face value X has a final value greater than or equal to that of the underlying asset if this pays no dividends

S^{exp} is the price of the stock at the expiration date and Π_C and Π_D are respectively the final values of portfolio C and D.

Since, in every possible state of the world, the value of portfolio C at maturity is greater or equal than the one of D, the current value of C has to be greater or equal than the one of D to avoid arbitrage opportunities. Therefore, it is possible to rewrite the value of a European call option at maturity as:

$$c(S,T,X) \geq \max(0, S - XB(T))$$

Recalling the relationship existing between the price of European and American options, we can add another term to the inequality:

$$C(S,T,X) \geq c(S,T,X) \geq \max(0, S - XB(T))$$

Again, this is true for non-dividend paying stocks and in the absence of transaction costs. If the asset provides a positive dividend yield, it is necessary to rely on Merton's adjustment[19].

In case of an early exercise of an American-style option, the payoff would be $\max(0, S - X)$ which, assuming a positive interest rate and a positive time to maturity, is lower than $\max(0, S - XB(T))$[20]. Acknowledging this result, the economically rational buyer of an American call will always choose the most efficient solution, thus, in such a scenario, he or she will sell the option rather than exercising it. The strong implication of this outcome is that for non-dividend paying assets it is never optimal to exercise the contract prior to the maturity date, and therefore American and European call options on such securities will have the same value.

So far, we have focused only on stock not providing any dividend yield, but what changes in case the asset does pay dividends? When the underlying security does pay a dividend to its holder, the early exercise of an American call option may become economically desirable. To understand why, let us consider the following portfolios in which we assume that the stock pays a dividend, D, on the expiration date of the analyzed option:

- E: purchase one European call and $X + D$ bonds.
- F: purchase one stock.

		$Stock\ price\ at\ T = 0$	
$Portfolio$	$Current\ value$	$S^{exp} < X$	$S^{exp} \geq X$
E	$c(S,T,X) + (X + D)B(T)$	$0 + X + D$	$(S^{exp} - X) + X + D$
F	$S(t)$	$S^{exp} + D$	$S^{exp} + D$
$Final\ value$		$\Pi_E > \Pi_F$	$\Pi_E = \Pi_F$

Table 2 - Proof that the value of a portfolio made of a call and a riskless bond with a face value equal to X + D has a final value greater than or equal to that of the underlying asset which pays D dividends at the expiration of the option

[19] "On the pricing of corporate debt: the risk structure of interest rates" by Merton, R. C., the American Finance Association Meeting (1974).

[20] Indeed, when T is greater than 0 and the risk-free rate is positive, the term B(T) will be less than 1. Therefore, the difference $S - XB(T)$ will be greater than $S - X$

Since the final value of portfolio E is never less than the one of portfolio F, in order to not allow arbitrage opportunities, the actual price of E must be not less than F. Therefore, it is possible to write:

$$c(S,T,X) \geq \max(0, S-(X+D)B(T))$$

Since the difference between the stock value and the discounted strike price plus the discounted dividend may be either positive or negative, in case of dividend payments it may be advantageous to early exercise an American call option.

An interesting and useful result that will be exploited in the development of the neural network regards the relationship existing between the option price and the exercise price.

In particular, the price of a call option can be seen as a convex function of the strike price[21]. Suppose we have three identical call options which differ only for the strike price. Specifically, let us assume that $X_1 \geq X_2 \geq X_3$ and let us define a weighting parameter lambda, $0 \leq \lambda \leq 1$.

The convexity condition will imply that $X_2 = \lambda X_1 + (1-\lambda)X_3$ and therefore:

$$C(S,T,X_2) \leq \lambda C(S,T,X_1) + (1-\lambda)C(S,T,X_3)$$

To demonstrate why this restriction has to be true we simply need to form two portfolios G and H.

- G: buy λ calls with strike price X_1 and $1-\lambda$ calls with exercise price X_3;
- H: buy one call option with strike price X_2.

		$Stock\ price\ at\ T=0$			
$Portfolio$	$Current\ value$	$S^{exp} \leq X_3$	$X_3 < S^{exp} < X_2$	$X_2 < S^{exp} < X_1$	$S^{exp} \geq X_1$
G	$\lambda C(S,T,X_1) + (1-\lambda) * C(S,T,X_3)$	$0+0$	$0+(1-\lambda)(S^{exp}-X_3)$	$0+(1-\lambda)(S^{exp}-X_3)$	$\lambda(S^{exp}-X_1) + (1-\lambda) * (S^{exp}-X_3)$
H	$C(S,T,X_2)$	0	0	$S^{exp}-X_2$	$S^{exp}-X_2$
$Final\ value$		$\Pi_G = \Pi_H$	$\Pi_G > \Pi_H$	$\Pi_G > \Pi_H$	$\Pi_G = \Pi_H$

Table 3 - Proof that a convex combination of calls with different strike prices yields a final value not lower than that of a call which is the convex combination of the other two exercise prices

Since the value at maturity of portfolio G is no less than the one of portfolio H, to eliminate stochastic dominance the present value of H must be small than or at least equal to that of G.

[21] In chapter 4, we will see empirically that this relationship holds also in a real-world scenario like the one presented in this dissertation.

1.3 The Black and Scholes model

The fundamental idea in the Black and Scholes model is to form a riskless hedge by constructing a portfolio made of stocks and European call options. In such a portfolio, the only source of uncertainty is given by the evolution of the prices over time. Indeed, since the price of the call option can be expressed as a function of the underlying price and the time to maturity, it follows that changes in the value of the option are determined by changes in the stock price and by the remaining life of the contract.

Black and Scholes observed that, at any point in time up to the expiration date, the portfolio could become a riskless asset by choosing the appropriate mixture of stocks and calls. For instance, if we form a portfolio buying a share of the stock and going short on the corresponding European call, a rise in the stock prices would have a twofold effect. On one side, the stock value would increase. However, on the other side, the profit of the long position would be offset by the proportional decrease of the value of the short position.

Someone could argue that once the price has changed the hedge is no more realized. This is true. Indeed, the model requires that the portfolio is continuously adjusted in an appropriate way, in order to compensate for the changes of the prices over time. If we perform this operation in the right manner, accounting for the value changes of the underlying asset, the portfolio becomes riskless. Therefore, to avoid arbitrage opportunities and to satisfy the stochastic dominance principle, we can conclude that it has to earn the risk-free rate.

In their model, Black and Scholes expressed the value of the hedge portfolio as the sum of two products:

- The number of shares of the stock times the stock price.
- The number of European call options times the call price.

$$\Pi_H = SQ_s + cQ_c$$

where Π_H is the value of the hedge portfolio, S is the stock price, Q_s is the quantity of stock currently held in the portfolio, c is the price of the European call option, and Q_c is the current quantity of calls.

The change in the value of the portfolio, $d\Pi_H$, is given by the first derivative of the previous formula:

$$d\Pi_H = Q_s dS + Q_c dc$$

In order to express the change in the value of the call option, Black and Scholes relied on stochastic calculus. In particular, by assuming that the stock price follows a Geometric Brownian Motion[22], i.e., the instantaneous log-return of stock price is an infinitesimal random walk with drift, Itô's lemma can be exploited to express dc as:

[22] We are assuming that the dynamics of the stock price evolution over time can be described as $\frac{dS}{S} = \mu dt + \sigma dz$, where μ is the instantaneous expected return on S, σ^2 is the instantaneous variance of the return, and dz is a Wiener process (Merton, 1971 and McKean, 1969). Note that in the original BSM (Black and Scholes model) the drift and the volatility are assumed to be constant.

$$dc = \frac{\partial c}{\partial S} dS + \frac{\partial c}{\partial t} dt + \frac{1}{2} \frac{\partial^2 c}{\partial S^2} \sigma^2 S^2 dt$$

What is interesting to notice is that the only stochastic term in this expression is dS since all the other quantities are deterministic. By substituting the expression of dc into the formula expressing the change in the value of the riskless portfolio, we obtain:

$$d\Pi_H = Q_s dS + Q_c \left[\frac{\partial c}{\partial S} dS + \frac{\partial c}{\partial t} dt + \frac{1}{2} \frac{\partial^2 c}{\partial S^2} \sigma^2 S^2 dt \right]$$

Note that by taking arbitrary values of Q_s and Q_c the evolution of the portfolio value is stochastic. However, if we choose in an appropriate way these quantities so that $Q_s dS + Q_c \frac{\partial c}{\partial S} dS = 0$, in other words if we pick Q_s and Q_c ensuring that $\frac{Q_s}{Q_c} = -\frac{\partial c}{\partial S}$ [23], then the portfolio return becomes riskless. By setting $Q_s = 1$ and $Q_c = -\frac{1}{\partial c / \partial S}$ and plugging them into the previous equation, the change in the hedge portfolio can be expressed as:

$$d\Pi_H = -\left(\frac{1}{\partial c / \partial S} \right) \left[\frac{\partial c}{\partial t} + \frac{1}{2} \frac{\partial^2 c}{\partial S^2} \sigma^2 S^2 \right] dt$$

We are well aware that, in equilibrium, two perfect substitutes have to provide the same rate of return to avoid arbitrage opportunities; thus, having constructed a riskless portfolio, to be coherent with stochastic dominance, the return has to be the same as the risk-free one.

$$\frac{\mathrm{d}\Pi_H}{\Pi_H} \rightarrow r \, dt$$ [24]

If we now rewrite this "equality" exploiting the definitions of $\mathrm{d}\Pi_H$ and Π_H we already derived, we obtain:

$$\frac{\partial c}{\partial t} = rc - rS \frac{\partial c}{\partial S} - \frac{1}{2} \frac{\partial^2 c}{\partial S^2} \sigma^2 S^2$$

which is a differential equation for the value of the option.

To solve this analytically, Black and Scholes relied on the equilibrium, no-arbitrage, condition imposing that on the maturity date the cost of the option must be equal to the maximum of either the difference between the stock price and the strike price or zero[25]:

$$c^{expiration} = \max(0, S^{expiration} - X)$$

[23] Note that this condition affects the ratio $\frac{Q_s}{Q_c}$; therefore, it makes no difference which asset is sold and which asset is purchased. If the stock were sold instead of the option, we simply need to adjust the result so that the number of shares sold per each bought call should be $-\frac{\partial c}{\partial S}$.

[24] Note that in this case we prefer to use the symbol → instead of the equal one to highlight the existing difference between the two. Indeed, this is "equal in equilibrium" and has a different meaning.

[25] They also exploited the heat exchange equation from physics.

To avoid overcomplex computations which go behind the scope of this thesis, it is better to rely on a more intuitive and simple solution technique presented by Cox and Ross (1976), which however leads to the same result.

The core idea is that whatever the solution of this differential equation is, it would be a function of only five variables: $r, S, T, \sigma^2, and\ X$[26]. Moreover, in the portfolio construction phase, the only assumption regards the fact that, in equilibrium, two perfect substitutes have to earn the same rate of return to ensure efficiency. This means that no further assumptions are made on the risk aversion of the market participants. Therefore, if a solution can be found assuming a proper preference structure, this should also be the general solution, meaning a valid solution for the differential equation for any different preference structure that ensures the equilibrium.

In other terms, we are allowed to choose the best structure, meaning the one that simplifies the computations the most.

The most useful assumption we can make is to assume a risk-neutral world. Therefore, Cox and Ross assumed all market participants to be risk-neutral investors. In such a case, the equilibrium rate of return is the same for all the assets, r, and the current value of a call option must be equal to the discounted expected value of the contract at maturity:

$$c = e^{-rT} E\left(c^{expiration}\right)$$

$$c = e^{-rT} \int_X^{\infty} \left(S^{expiration} - X\right) L'\left(S^{expiration}\right) dS^{expiration}$$

where $L(S^{expiration})$ is the cumulative log-normal distribution function and $L'(S^{expiration})$ is the log-normal density function for $S^{expiration}$.

Note that it is possible to exploit a useful theorem in solving integrals involving the log-normal distribution. The theorem says that if $L(S^*)$ is a log-normal density function with

$$Q = \begin{cases} \lambda S^* - \gamma X & if\ S^* - \Psi X \geq 0 \\ 0 & if\ S^* - \Psi X < 0 \end{cases}$$

then

$$E(Q) = \int_{\Psi X}^{\infty} (\lambda S^* - \gamma X) L'(S^*) dS^*$$

$$= e^{\rho T} \lambda S * N\left\{ \frac{\ln\left(\frac{S}{X}\right) - ln\Psi + \left[\rho + \left(\frac{\sigma^2}{2}\right)\right] T}{\sigma\sqrt{T}} \right\} - \gamma X * N\left\{ \frac{\ln\left(\frac{S}{X}\right) - ln\Psi + \left[\rho + \left(\frac{\sigma^2}{2}\right)\right] T}{\sigma\sqrt{T}} \right\}$$

[26] Respectively: the risk-free rate of return, the current stock value, the time to maturity, the variance of the returns, and the exercise price of the option.

where $\lambda, \gamma, and\ \Psi$ are arbitrary parameters, ρ is the expected average rate of growth of the stock price[27], and N is the cumulative standard normal distribution function[28].

Therefore, it is possible to solve the equation of the call option value by applying the above-mentioned theorem with $\lambda = \gamma = e^{-rT}$ and $\Psi = 1$. By substituting ρ with the risk-free rate, it is then possible to derive the general solution:

$$c = S * N\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r + (\frac{\sigma^2}{2})\right]T}{\sigma\sqrt{T}}\right\} - e^{-rT}X * N\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - (\frac{\sigma^2}{2})\right]T}{\sigma\sqrt{T}}\right\}$$

This is the Black-Scholes model.

Let us try to provide a more intuitive understanding of the equation. In order to reach this objective, let us suppose there is no uncertainty about the outcome, the future value of the stock. In such a situation, the final value of the call option will be strictly positive, otherwise no one would have ever purchased it. In particular, the cost of the contract will be equal to the difference between the final value of the stock and the exercise price: $c^{expiration} = S^{expiration} - X$.

Since in equilibrium the return of all assets must be the same in order to prevent arbitrage opportunities, it is possible to express the final value of the stock as: $S^{expiration} = Se^{rT}$ considering a continuously compounded rate of return. Therefore, the current option value can be rearranged in the following way:

$$c = (S^{expiration} - X)e^{-rT} = (Se^{rT} - X)e^{-rT} = S - Xe^{-rT}$$

As we can see, this expression differs from the previous one only in the fact that it is not multiplied by the cumulative standard normal distribution. Indeed, if we consider a real-world scenario, characterized by some uncertainty about the possible final result, those two cumulative distributions can be seen as the probabilities reflecting the existing uncertainty.

We already pointed out that the Black and Scholes model is a function of only five variables, all observable except for the variance, which has to be properly estimated. However, in more recent papers, different variables, both observable and unobservable, have been introduced in the effort of developing models capable of providing more accurate predictions and estimates. In any case, these modifications have to respect the restrictions placed on the option value by the stochastic dominance principle. In particular:

- When the underlying asset price increases, so does the option price:

$$\frac{\partial c}{\partial S} = N\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r + \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\} > 0$$

Assuming a log-normal distribution of stock prices, the expected final value is a positive function of the current price.

[27] $e^{\rho T} = E\left(\frac{S^*}{S}\right)$

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

- If the strike price rises, the call value decreases:

$$\frac{\partial c}{\partial X} = -e^{-rT} N\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\} < 0$$

- As the time to expiration expands, the option price increases:

$$\frac{\partial c}{\partial T} = Xe^{-rT}\left[\frac{\sigma}{2\sqrt{T}} N\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\} + rN\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\}\right] > 0$$

This reflects the idea that the present value of the strike price becomes lower if the time to maturity increases.

- A growth of the risk-free rate determines a growth of the call price:

$$\frac{\partial c}{\partial r} = TXe^{-rT} N\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\} > 0$$

This property is extremely similar to the previous one. By increasing the discount rate, the present value of the sum discounted diminishes.

- Finally, the variance and the price of the option are positively correlated. By increasing the first, also the second follows its path.

$$\frac{\partial c}{\partial \sigma^2} = Xe^{-rT} N'\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\}\frac{\sqrt{T}}{2\sigma} > 0$$

  The idea is that an increase in the variability of the rate of return makes large positive price movements more likely to happen. Of course, also the probability of large negative changes rises but, since the price cannot be negative, this has overall a positive effect on the price of the call.

By merging together all the previous results and ideas, it is possible to derive a theoretical graph of how the Black and Scholes price evolves considering changes of the underlying prices, a fixed interest rate, a constant variance rate, and a given time to maturity.

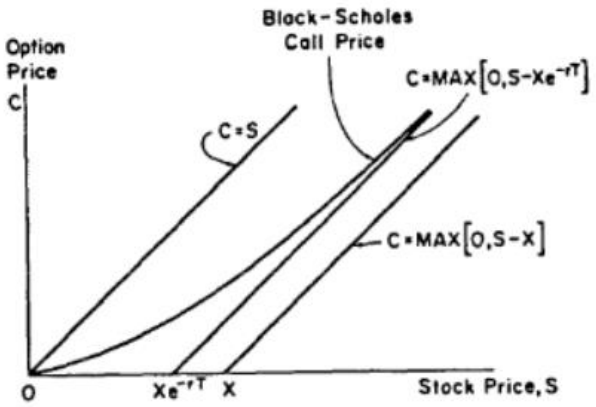

Figure 2 - Evolution of Black and Scholes call option price for different underlying asset values (source: Journal of Financial Economics)[29]

We can observe that the Black-Scholes price lies between two extremes: the maximum possible value, $C = S$[30], and the minimum price, $C = \max(0, S - Xe^{-rT})$.

1.4 Extensions of the model

Having said that, we can now consider the most popular extensions of the model. Specifically, we will focus on Merton's adjustment for dividends, since the model we will use as a benchmark to test the performance of our neural network will be the Black-Scholes-Merton model. First of all, we need to clarify that papers following the first publication of the Black and Scholes article had a twofold objective. On one side, they wanted to assess the goodness and the reliability of the model by testing it both from a theoretical and an empirical point of view. On the other side, they tried to relax one of the several assumptions that Black and Scholes had made, analyzing and testing the robustness of the equation[31].

We already said that Merton was able to prove that if a stock pays no dividends, then it will not make any economic sense to early exercise a call option written on it. Consequently, for non-dividend paying stocks the Black-Scholes pricing model may be applied to evaluate also American options. Merton then moved on with his research by improving the original model. One of the key assumptions of the pricing equation was the fact that the asset did not pay any dividend over the life of the option. However, Merton was able to relax this condition by assuming a special dividend policy. Specifically, he supposed that dividends were paid continuously. With this restriction the dividend yield, δ, became constant over time. So, it was possible to compute the riskless portfolio in the usual way. However, in this case, the stock return was given by two factors: the return itself and the dividends yield. Therefore, the change in the value of the hedge portfolio can now be expressed as:

[29] "Option pricing, a review" by C. W. Smith, Jr, Journal of Financial Economics, Vol. 3, pp. 3-51 (1976).
[30] We are excluding for simplicity the extreme case in which $S = 0$.
[31] A summary discussion about the most important adjustments to the BS equation is provided by Haug (2007).

$$d\Pi_H = Q_s(dS + \delta S dt) + Q_c\left(\frac{\partial c}{\partial S}dS + \frac{\partial c}{\partial t}dt + \frac{1}{2}\frac{\partial^2 c}{\partial S^2}\sigma^2 S^2 dt\right)$$

Again, by taking into consideration a proper number of shares and calls, it is possible to make the hedge riskless. Setting $Q_s = \frac{\partial c}{\partial S}$ and $Q_c = -1$ the change in the value of the portfolio becomes:

$$d\Pi_H = \frac{\partial c}{\partial S}\delta S dt - \left(\frac{\partial c}{\partial t} + \frac{1}{2}\frac{\partial^2 c}{\partial S^2}\sigma^2 S^2\right)dt$$

Since in equilibrium no arbitrage opportunities may be detected, the hedge portfolio must earn the riskless interest rate. Therefore, considering $r = \rho + \delta$ the price of the call becomes:

$$c = e^{-rT}\int_X^{\infty}\left(S^{expiration} - X\right)L'\left(S^{expiration}\right)dS^{expiration}$$

Relying on the theorem and setting $\lambda = \gamma = e^{-rT}$ and $\Psi = 1$ we get:

$$c = e^{-\delta T}SN\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \delta + \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\} - e^{-rT}XN\left\{\frac{\ln\left(\frac{S}{X}\right) + \left[r - \delta - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\}$$

which is the solution for the European-style call option pricing problem in case the stock does pay a continuously compounded dividend yield, δ. Note that, since dividends are paid, this equation is not appropriate in valuing American options, because in that case an early exercise is possible.

By defining two new variables d_1 and d_2:

$$d_1 = \frac{\ln\left(\frac{S}{X}\right) + \left[r - \delta + \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln\left(\frac{S}{X}\right) + \left[r - \delta - \left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}$$

It is possible to express the result in a more compact way:

$$c = Se^{-\delta T}N(d_1) - Xe^{-rT}N(d_2)$$

Other useful extensions of the original model have been proposed over time. It is interesting to notice that they have all been developed following a similar procedure. Once a proper setting was defined, the researcher proceeded by relaxing one of the assumptions made by Black and Scholes. Just to mention a couple of examples, Merton further generalized the model considering a variable interest rate instead of a constant yield, and Ingersoll (1975) modified the model to account for taxes and transaction costs.

By removing the assumption on the stock price dynamics, things become a lot more complicated. In particular, we have already seen that the most common settings are the ones that assume the price movements to be generated by an Arithmetic Brownian motion or a Geometric Brownian motion. The problem is that these models are fine only when applied to continuous data and, in reality, this

is not always the case. An alternative specification which allows for discrete scenarios is to assume the price dynamics to follow a Poisson process. According to it, the stock value will jump to a new level in each period of time with a low probability[32]. Note that the Poisson distribution is a good choice since a corollary to Itô's lemma can be exploited to differentiate such processes.

So far, we have only focused on call options. Therefore, a question should spontaneously arise: what about put options? Following the same procedure adopted for call options, Merton was able to develop some equilibrium conditions on the value of a put. From the stochastic dominance argument Merton derived that the value of a put option at expiration has to be equal to either the difference between the strike price and the underlying price or zero:

$$P\left(S^{expiration}, 0, X\right) = p\left(S^{expiration}, 0, X\right) = \max\left(0, X - S^{expiration}\right)$$

If we suppose borrowing and lending rates to be the same, the cost of a European put option can be expressed as the value of a portfolio made of a European call option with the same characteristic as the put, plus a riskless bond with a face value of X and a short position in the underlying asset. In order to show this relationship, let us consider the following two portfolios I and L:

- I: buy one share of the stock, one European put option, and X dollars borrowed for T time periods.
- L: long one European call with the same underlying asset, expiration, and strike price as the put.

		$Stock\ price\ at\ T = 0$	
$Portfolio$	$Current\ value$	$S^{exp} \leq X$	$S^{exp} > X$
I	$S + p(S,T,X) - XB(T)$	$S^{exp} + X - S^{exp} - X$	$S^{exp} + 0 - X$
L	$c(S,T,X)$	0	$S^{exp} - X$
$Final\ value$		$\Pi_I = \Pi_L$	$\Pi_I = \Pi_L$

Table 4 - Proof that a portfolio made of one share of stock, one European put, and X dollars borrowed for T periods provides a final value equal to that of a European call with the same strike price and time to expiration

To avoid dominance, the call option has to be priced so that:

$$c(S,T,X) \leq S + p(S,T,X) - XB(T)$$

where B(T) is the current value of one dollar payable in T years from now.

Of course, it is possible to rewrite the inequality in terms of the put value by considering two different portfolios M and N:

- M: long a European call, X riskless bonds and short the stock.
- N: long a European put with the same features as the call.

[32] A simple version of a Poisson process can be seen as: $\frac{dS}{S} = \mu dt + (\gamma - 1)$ with probability λdt, and $\frac{dS}{S} = \mu dt$ with probability $(1 - \lambda)dt$. The λ parameter is the intensity of the process while $\gamma - 1$ is the magnitude of the movement.

		$Stock\ price\ at\ T = 0$	
$Portfolio$	$Current\ value$	$S^{exp} \leq X$	$S^{exp} > X$
M	$c(S,T,X) - S + XB(T)$	$0 - S^{exp} + X$	$S^{exp} - X - S^{exp} + X$
N	$p(S,T,X)$	$X - S^{exp}$	0
$Final\ value$		$\Pi_M = \Pi_N$	$\Pi_M = \Pi_N$

Table 5 - Proof that a portfolio made of one European call, a bond with a face value of X, and one share of stock sold short provides a final value equal to that of a European put with the same strike price and time to expiration

According to the stochastic dominance principle, the put has to be worth not less than:

$$p(S,T,X) \leq c(S,T,X) - S + XB(T)$$

This means that, assuming the borrowing and lending rate to be the same, a European put must be priced so that:

$$p(S,T,X) = c(S,T,X) - S + XB(T)$$

This is an extremely important relationship in the derivatives field and, by expressing it in a slightly different way, it becomes the well-known put-call parity.

$$Put - Call\ parity\text{:}\ p(S,T,X) + S = c(S,T,X) + Xe^{-rt}$$

where $B(T) = e^{-rt}$.

From this point, relying on the same restrictions applied for the call option case, it is possible to derive a series of conditions that have to be met in order to ensure efficiency and to avoid arbitrage opportunities. In particular, the price of a European put cannot be greater than the one of a zero-coupon bond with face value X[33]:

$$p(S,T,X) \leq XB(T)$$

Moreover, an American put option cannot be traded at a lower price than the corresponding European counterpart:

$$P(S,T,X) \geq p(S,T,X)$$

Furthermore, since, differently from the call option case, when we consider a put option the probability of an early exercise is never zero, either if the asset does or does not pay any dividend, the American put option price should be strictly greater than the corresponding European one[34].

[33] This is the same as the condition on the call option saying that a call price cannot be greater than the underlying stock price.

[34] Suppose for instance that the underlying asset price falls far below the strike price, the maximum value that the put can deliver is, in any case, X and it will be obtained only if the stock value reaches zero. If the put can be early exercised, a risk-free bond can be purchased allowing the investor to earn the corresponding interest rate. For extreme cases, such as the one above described, the return provided by holding the put can be lower than the one we may obtain through an early exercise and the subsequent purchase of the bond. Therefore, the put may be more valuable if exercised.

Given the Black and Scholes assumptions, the value of the European put, assuming a Geometric Brownian motion, is:

$$p = Xe^{-rT}N\left\{\frac{-\ln\left(\frac{S}{X}\right)-\left[r+\left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\}-SN\left\{\frac{-\ln\left(\frac{S}{X}\right)-\left[r+\left(\frac{\sigma^2}{2}\right)\right]T}{\sigma\sqrt{T}}\right\}$$

All we have to do to modify the model and make it able to deal with dividends, etc. is to substitute the appropriate solution already derived.

Considering the previous expressions of d_1 and d_2 it is then possible to compactly express the price of a European put as:

$$p = Xe^{-rT}N(-d_2)-SN(-d_1)$$

1.5 Drawbacks of the model

Now, we should have a clear understanding of how the Black-Scholes model works, and in theory we should be able to correctly price any European option, once the required inputs have been provided. Remember that this discussion was started after having observed the increasing importance of this derivative instrument in today's financial markets, and consequently the need for a correct and efficient pricing formula. The next step will focus on explaining the most important drawbacks of this equation, which derive from the fact the model is built on non-real life assumptions, and will introduce the reasons why we need to develop more sophisticated, usually numerical, solving methods for addressing this kind of problems. Indeed, it should be obvious that the simplifying assumptions the model is based on often fail to meet real market data.

Let us start by presenting the easiest limitations for which a satisfactory solution has already been derived. Then we will move to the restrictions which are harder to tackle.

We know that the original model presented by Black and Scholes assumes a constant and known interest rate. In particular, given the no-arbitrage condition, the considered rate is the risk-free one. The problem in this case is twofold, since in the real world there is no such thing as a completely riskless asset, and interest rates are not constant over time, but they tend to change following macroeconomic events or changes in market volatility[35]. However, we already know that Merton et. al (1976) were able to introduce an adjustment which allows the model to deal with a variable interest rate.

A problem of the same magnitude is the one set by the no-dividend assumption and, even in this case, Merton (1975) was able to derive an analytical solution which makes the model suitable for securities paying dividends. However, also this new equation relies on some strong assumptions. In particular, it defines a continuously paid dividend yield which is supposed to be fixed over time.

[35] Despite this, it is still possible to detect some assets, like for instance long-term U.S. Government bonds, which are good substitute of theoretically risk-free securities.

Therefore, even this approach lacks the ability to deal with the evolution of dividend policies we usually observe in real world scenarios[36].

The assumption regarding the absence of transaction costs is strictly correlated with what has just been said. The model assumes the market to be frictionless, however in reality things are not that easy, and we have to account for commissions, fees, spreads and other transaction costs, including the time delay for the execution of an order, which may translate in practice into the impossibility of reaching a perfect hedge. About this, it is worth mentioning that Grossman and Zhou (1996) showed that the volatility behaviour depends, to a certain degree, also on the volume of the trade and the transaction costs, and not only on the stock price dynamics. Remember that, as already told, Ingersoll (1975) modified the model to account for taxes and transaction costs.

Other limitations that do compromise the investor's ability to construct the hedge portfolio described by Black and Scholes are the presence of arbitrage opportunities in financial markets and the illiquidity of some securities. On one hand, one prerequisite of the pricing equation is the one derived by Merton relying on the stochastic dominance principle: in equilibrium arbitrage opportunities cannot exist. However, many papers have showed empirically that this assumption is quite often violated. Arbitrages do exist and they can be exploited, see for instance Ambrož (2002). Of course, a model which does not account for this possibility has a strong limit and may lead to abnormal results. On the other hand, the model assumes that an investor can buy and sell any number of stocks and options without any limitations[37]. Again, this is not coherent with what we observe in reality, where one is bounded by various factors: buying power, market liquidity, institutional factors[38], available shares, regulations, and so on.

Note that even if it may not seem so important, this is a crucial assumption when it comes to the real world. Therefore, assuming perfectly liquid markets is not only unrealistic, but can also be fatal[39].

Another problem of the former formulation is the fact it only considers European-style options. While in the case of American-style call options on non-dividend paying assets the model is still valid, for put contracts and calls written on dividend paying stocks this is not true anymore, and extending the result is not always straightforward and can indeed require complex pricing techniques[40].

The last two drawbacks deriving from the Black and Scholes model assumptions regard the process generating the prices and the volatility.

We already saw that the price dynamic is usually determined by exploiting an Arithmetic Brownian motion or a Geometric Brownian motion. According to these models the asset prices rise and fall due to unobservable circumstances. The problem in this case is that with a random walk the price of the underlying can go up or down, at any point in time, with the same probability and magnitude.

[36] Usually, we expect dividend to increase over time. However, in problematic situations they can also decrease or be canceled.

[37] In terms of quantities, trading hours, etc.

[38] Like for instance the "possibility of extension", meaning the potential increasing of the life of the contract. See Longstaff (1990) as an example.

[39] One should take as an example what happened in 2007-2008.

[40] Note that also in these cases the pricing task requires preliminary assumptions concerning the behaviour of the underlying asset.

This is usually not the case, since stock prices are determined by a lot of different factors that affect these probabilities in complex and usually asymmetric ways. Furthermore, with these model specifications we are implicitly saying that the price movement at time $t + 1$ is independent from the movement of the price at time t[41], however in reality price movements tend to exhibit a positive correlation.

At this point, we have to recall that in the Black and Scholes paper they chose to rely on the normal distribution[42]. The problem is that the results provided by the model are strictly dependent on the distributional form chosen for the equation. Therefore, once a distribution has been applied, the model is locked in by that choice. It means that, even assuming the normal distribution to be able to correctly deal with market data, and, in reality, this is often not true, the results will always be bounded by this restriction. Obviously, this is a huge problem, since empirical analyses[43] have clearly showed that prices and returns tend to display "strange" behaviours which cannot be harmonized with a normal distribution, such as:

- heavy-tailed returns[44],
- positive correlations of the squared returns,
- changing volatility,
- volatility clustering.

Note that since most of these limitations are about fundamental aspects of the market, it is necessary, in order to provide accurate and reliable predictions, to come up with more sophisticated models. So far, many different formulas, trying to capture all these features, have been proposed. However, we can only attempt to capture most of the aspects, since it would be virtually impossible to transpose every relevant factor in a closed-form formula which can be analytically solved. As we will see in a moment, this is the reason why we are more and more interested in developing numerical methods, such as neural networks.

The last and most serious problem arising from the Black and Scholes model regards the volatility. We already know that this is the only unobservable variable in the formula; therefore, its estimation is a crucial aspect which has attracted more and more interest over time[45].

We all know that volatility is a measure of how much an asset can be expected to move in a certain period of time. Considering options, it is possible to define a first main distinction between historical volatility and implied volatility:

[41] This is the martingale property of the Brownian motion.

[42] This is already a mistake since according to Hull's research, empirical data have showed that returns tend to be "leptokurtic", meaning they have a much higher probability of exhibiting outliers than would be the case if they were truly normally distributed.

[43] Consider for instance Teneng (2011) who proved that the Black and Scholes model fails to correctly address many aspects of real data.

[44] Securities returns tend to show finite variance and semi-heavy tails which is in contrast to stable distributions like log-normal with tend to display infinite variance and heavy tails (Clark, 1973).

[45] Limiting the volatility modeling to the derivatives pricing field still leaves many different models. Grouping them according to their basic assumptions we can define the following categories: historical volatility, stochastic volatility, volatility term-structure, volatility surface models, and non-parametric volatility.

- *Historical volatility*, sometimes called statistical volatility, is a sort of past-looking metric which is defined as either the observed standard deviation or the observed variance[46] of the price changes of the underlying asset over a specified period of time.
- I*mplied volatility* instead, also called projected volatility, is a forward-looking measure which tries to estimate how volatile the underlying security will be in the near future. It is derived from the quoted option prices. In particular, relying on a pricing model and exploiting the information provided by the current value of an option, it is possible to solve the equation and to compute the volatility the market is currently pricing.

So, while historical volatility refers to the observed past data, implied volatility represents the market expectations. Therefore, there is a huge advantage in using the implied volatility, since it relies on present market data instead of historical observations. Furthermore, many papers have proved that relying on historical volatility can be a huge mistake, since the past conditions may be significantly different from the current ones, see for instance Černy (2008).

The drawback of the Black and Scholes former model is the fact it assumes a constant volatility over time. Empirical studies have indisputably proved that the constant volatility assumption is not met when we consider real financial data. Moreover, it has been showed that it is the violation of this restriction which generates the most significant mistakes in the pricing task.

In any case, we have to make some distinctions. Indeed, even if it is true that volatility changes over time, the magnitude of the changes is not always the same. In particular, if we consider a short-term horizon, volatility seems to be relatively constant. If instead we increase the length of the analyzed period, this behaviour is disrupted. In other words, this measure shows some positive partial correlation: in the short-term, large price changes tend to be followed by large movements, and vice versa. This is a property called volatility clustering[47].

Note also that volatility measures are usually negative correlated with asset price returns. Indeed, analyses on real data have showed that security volatilities tend to decrease as the stock prices rise, Christie (1982); this is the so-called leverage effect.

Taking into account all what has been said so far, we should not be surprised in hearing that when applying the Black-Scholes model to real data, the results provided by the pricing formula present some systematic differences with respect to the observed prices. In particular, when we plot the implied volatility determined by options with the same time to maturity but different strike prices or different moneyness, the graph that will be created is referred to as the volatility smile. This means that the pricing differential equation is better in estimating the price of at-the-money options than the ones of ITM and OTM options[48].

We can see a graphical representation of this pattern in Figure 3:

[46] Also, other variability measures can be used, but these two are by far the most common ones.
[47] Nowadays, a common way to deal with such a behaviour is thorough the use of generalized autoregressive conditional heteroskedasticity models, i.e., GARCH models.
[48] See for example Campa, Chang and Reider (1998) or Rosenberg (1997). However, it has to be said that ATM options are the most traded ones.

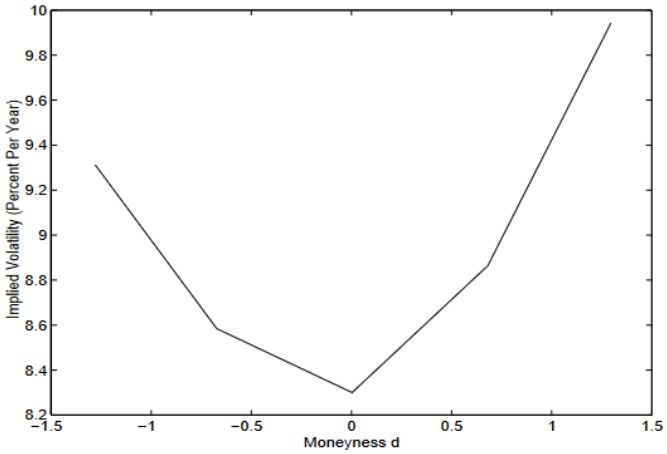

Figure 3 – Example of volatility smile (source: National Bureau of Economic Research) [49]

Note that smiles and other recurrent biases[50] we come in touch with when applying the Black and Scholes model are often attributed to departures from the original assumptions.

To conclude this first chapter, we can say that, regardless of its considerably limiting assumptions, the Black and Scholes model is still widely used by investors to price European-style options, and it is still one of the most fundamental equations in the derivative field.

Over time, many refined versions of the original formula have been proposed to improve the model accuracy by removing some of its key restrictions. Among all the assumptions, we have seen that the most serious and problematic is the one concerning the volatility, both because it is the one impacting the most on the results, and because it is the hardest to tackle. With this regard, a lot of possible solutions have been developed in the attempt to include a non-constant volatility in the model, either in a deterministic or in a stochastic way. Nevertheless, so far there is no such thing as a general formula which is recognized to be universally applicable. However, it is also true that some of the assumptions introduce a small limitation which can be seen as insignificant, and therefore the Black and Scholes model can be considered a good pricing model which provides approximately correct estimates of option prices. Yet, it is characterized by some serious problems and the systematic presence of biases cannot be ignored. Therefore, having observed the constantly increasing importance of this kind of financial instruments, we understand the pressing need for more advanced pricing tools which allow for better estimates both in terms of accuracy and time efficiency.

As already told, it will be virtually impossible to account analytically for all the relevant aspects determining the price of an option, however, relying on the expanding computational ability of computers and the huge amount of available data, nowadays it is possible to take a different path and exploit numerical solving techniques and machine learning algorithms which do not provide analytical closed-form solutions but extremely precise approximations instead. The next chapter will

[49] Note that the figure comes from the paper "Implied exchange rate distributions: evidence from OTC option markets" by Campa et al. (1998) and it regards one-month deutschemark options for five different strikes on April 3, 1996.

[50] For instance, it is possible to identify a systematic bias in the pricing of ATM options with short-time maturities. In particular, Black (1975) said that "Options with less than three months to maturity tend to be overpriced" by the Black and Scholes formula, and other economists confirmed it. See for example Bodurtha and Courtadon (1987).

focus precisely on presenting these new concepts and how they can be applied to the task of option pricing.

Chapter 2 Machine Learning and Artificial Neural Networks

In this Chapter Machine Learning (ML) and Artificial Neural Network (ANN) models are presented. Specifically, the first section introduces the concept of machine learning, the most popular techniques and the classifications that it is possible to apply to such models. Paragraph 2.2 instead is mainly focused on the historical development of ANNs, presenting both the theoretical foundations and the mathematical formulation that have been proposed over time. Finally, in the third part of the Chapter the functioning and the most important characteristics of these algorithms are described, paying, at the same time, close attention to the solutions that can be implemented to deal with the main drawbacks of such models.

2.1 Introduction to Machine Learning

We concluded the previous chapter making clear that closed-form models may not be the most efficient way to follow when trying to develop methods for solving complex problems, such as option pricing. Indeed, analytical solutions always require preliminary assumptions which inevitably make the model unrealistic and, to a certain degree, abstract from reality; therefore, they usually lead to unsatisfactory results. With this regard, an alternative procedure consists in not trying to provide an exact analytical solution but a reliable numerical approximation instead.

In order to achieve this objective, many artificial intelligence (AI) systems and machine learning (ML) algorithms have been developed in the past decades[51]. What do we refer to when using these terms? Answering the question is not an easy task. Indeed, "*The definition of machine learning is inchoate, and it is often context specific*." (Gu, Kelly and Xiu, 2020). According to Arthur Samuel (1959) machine learning is "*the field of study that gives computers the ability to learn without being explicitly programmed*". Therefore, we can summarize by saying that machine learning methods are techniques used to teach computers how to handle data in a more efficient manner exploiting a sort of self-improvement approach.

Providing a more formal definition, these are data-driven models which try to mimic intelligent behaviours existing in nature[52] with the aim of solving problems in a nonparametric way, relying on a collection of high-dimensional formulas, combined with regularization techniques and algorithms for efficient model selection.

The first difference with traditional methods is therefore the high-dimensional nature of machine learning systems, feature that helps explaining the huge degree of flexibility of these kinds of

[51] Preliminary distinction: the term Artificial Intelligence refers to the theory and the development of computers able to perform tasks normally requiring a human intelligence. The concept of Machine learning instead applies to a branch of AI that aims at providing machines the ability to automatically learn from data and past experiences.

[52] For instance, Genetic Algorithms (GA) are inspired from the way in which genes and chromosomes reproduce and try to exploit the concept of natural selection and survival of the fittest. Particle Swarm Optimization (PSO) is a computational technique that aims at optimizing a process by iteratively improving a candidate solution copying the movement of organisms in a bird flock or fish school. Neural networks (NN) take inspirations from the human brain, and so on.

models. As already told, ML algorithms follow a data-driven approach, meaning their structure evolves according to the data we feed them with, in a way that tries to maximize the use of the existing information. Moreover, responding to structural changes in the process generating the observation and not being fixed at priori, these algorithms are able to deal with any kind of process, even highly nonlinear, making them an extremely flexible tool and a "*universal approximator*". Finally, since they do not rely on any previously specified assumption, they tend to be much more robust to specification errors which do plague closed-form formulas instead.

Of course, all these advantages do not come without a cost. Indeed, machine learning techniques require extremely large quantities of data to properly train the algorithms and computers with a sufficiently strong computational ability to perform all the required operations in a time efficient and manageable way. Fortunately, nowadays, thanks to the success of computer science[53] and the huge amount of data availability, these two problems can be partially solved. Therefore, the drawbacks of such methods have to be sought elsewhere. In particular, being so flexible and relying so much on past data, machine learning techniques have a high propensity to overfitting, meaning that they tend to produce results that represents too well a specific set of data and fail to generalize additional observations, i.e., their predictive power is not reliable. However, we will see that regularization techniques may be exploited to prevent this kind of behaviour, reducing the impact of overfitting.

It is possible to classify these systems along many different dimensions. For instance, we can provide a classification based on the operating principle exploited by the learning algorithm. As an example, we can think about decision trees, which try to learn by discriminating among classes of objects, or neural networks which instead learn by adjusting and continuously improving the weighting coefficients they rely on[54]. An alternative is to organize these models according to the task they try to solve. In particular, a macro distinction in this case is between classification and regression problems. In the first case, the algorithm assigns distinct labels to the observed data according to some relevant characteristics, while in the second case it tries to provide correct estimates of the analyzed variables[55].

It should be noted that both problems concern option pricing. For instance, "*one possible application of classification is the decision of whether to exercise an American option early*" (Hahn, 2014), while for regression-type problems, it is sufficient to remember that the aim of this thesis is exactly to provide a model for option pricing.

A third distinction regards the domain for which the knowledge is acquired (Carbonell, Michalski and Mitchell, 1983). Indeed, machine learning is a multidisciplinary concept which comes in touch with a lot of different areas and has many practical applications. Just to mention, nowadays machine learning helps making medical diagnoses and has various uses in a lot of different fields such as agriculture, engineering, and finance. Furthermore, these algorithms are used for several tasks like image and speech recognition, spam identification, self-driving cars and so on. These are just few

[53] The emergence of GPUs leads to significant improvements in the performance of computers and makes working with big data easier.

[54] See for instance Vanstone and Hahn (2010).

[55] Therefore, a classification problem arises when the output is a category while a regression problem is present when the result is a real continuous value.

examples of possible applications of machine learning systems, but of course this is by no means an exhaustive list, since these algorithms affect potentially every aspect of our life by now.

In any case, the most important categorization is the one based on the underlying learning methodology the algorithm uses. Actually, depending on the data availability and the amount of inference the learning system performs, we can identify three basic families of machine learning:

- Supervised learning (SL),
- Unsupervised learning (UL),
- Reinforcement learning (RL).

The goal of supervised learning is to detect a function able to link the input features, namely the explanatory variables, to the output values, also called labels[56], by exploiting the information contained in the available data. Therefore, it tries to infer the unknown connection that may exist between the provided input-output pairs. In particular, supervised learning requires that the data we feed to the algorithm are correctly labelled/classified, meaning we know every useful information about the data except for the relationship that maps the input to the outputs.

Having said that, it should be clear that during a supervised learning procedure the model will always be able to check if either its classification or its prediction is correct. The model learns from the training dataset, i.e., the labeled data, by comparing its predicted output to the true output, and evolving its structure accordingly. Then, once the model has been sufficiently trained, it can be used to predict or classify any new, future, unseen observation.

At this point we can ask ourselves when is it possible to say that the model has been properly trained? Or better, how do we measure the model precision?

There exist several alternatives to define accuracy measures but, in general, machine learning algorithms rely on a so-called loss function, also known as cost function. This function simply quantifies the difference between the true value provided by the available data and the output estimated by the model. Of course, the goal of the algorithm is to minimize this difference making the cost function as small as possible.

We already mentioned that there are many possible machine learning models which can be implemented to solve different kind of problems. With this regard, Mahesh (2019) said that there is "*no single one-size-fits-all type of algorithm that is best to solve a problem*". Therefore, the algorithm we are going to use depends on the nature of the problem we are trying to solve. Critical elements that guide us in selecting a model over another are the number of variables we have to deal with, the quantity and quality of the available data, the level of precision we need, etc.

Now, it is worth mentioning some of the most famous and widespread supervised machine learning algorithms, which are:

- Decision tree: The core idea is to represent the possible choices and their results in form of a tree with nodes and mutually exclusive branches. Each node represents a feature of a group that has to be classified and each branch represents the value that a node can take.

[56] Both input and output variables can be quantitative or categorical quantities, Hastie et al. (2009).

A generalization of such approach is called “random forest” and simply consists in considering together many decision trees.

- Support vector machine (SVM): This is one of the most robust prediction methods currently available. Its functioning is quite simple, provided a set of labeled data the SVM algorithm builds a model that assigns new examples to one category or the other. Relying on the so-called “Kernel trick”, or Kernel adjustment, SVMs are able to perform even highly non-linear classifications.
- Artificial neural network (ANN): We will see it in detail in the next paragraphs[57].

At the extreme opposite of supervised learning there is unsupervised learning. The main differences are that in this second case the input features are not labeled, and the correct output is not known a priori[58]. Therefore, while in SL the algorithm is taught by the provided input-output pairs and there is a sort of external assistance which reduces the amount of inference the model has to perform, in UL this is not true, and the algorithms are left on their own in discovering particular structures and patterns present in the data.

Of course, the different nature of the learning procedure also affects the tasks that these methods are employed to solve. Indeed, unsupervised machine learning models are mainly focused on approximately learn the distribution of the data in the space, meaning they try to recognize recurrent patterns and group together similar observations[59]. Hence, instead of providing a precise estimate of a variable such systems help in developing a better understanding of the environment from which the data are extracted, thus they are primarily used for clustering and feature reduction (Hull, 2021).

The most common unsupervised learning algorithms are:

- K-means clustering. The main idea is to define k centers, where k is the number of clusters we need in our classification task. These centers should be located in an efficient and smart way since their location affects the result. The best choice is to place them as far away one from the other as possible. Then each data point is associated with the nearest center. Once it has been done, k new centroids are computed exploiting the new incorporated information and the process is repeated iteratively.
- Hierarchical clustering. Also known as hierarchical cluster analysis, it is an algorithm that starts by considering each observation as a separate cluster. Then, it identifies the two clusters that are closest together and merges them. The procedure continues until all the clusters are merged or until the result is satisfactory enough[60].
- Principal Component Analysis (PCA). It is a statistical method that uses orthogonal transformations to convert a set of possibly correlated variables into a set of values of

[57] Note that the structure of an ANN makes it a hybrid able to deal with both supervised and unsupervised learnings.
[58] See for instance Hull (2021).
[59] This task is also known as Data Clustering and nowadays it is extremely important in a lot of different applications such as market and consumer analysis and classification.
[60] As for all the others machine learning algorithm, a crucial point is the definition of a stopping criterion which can be based on a satisfactory measures or different rules such as the maximum number of iterations.

linearly uncorrelated measures called principal components. It is used to perform a change of basis on the data, and therefore it is often exploited as a dimension reduction technique[61].

Lastly, reinforcement learning. This is the area of machine learning which is closer to how humans learn since it is based on a sort of "*trial and error*" learning procedure. Indeed, in these kinds of models the learner improves its knowledge by exploiting what it has already learnt and exploring new opportunities with the aim of achieving better and better results[62]. In particular, RL is concerned with how agents should act in a given environment in order to maximize some notion of cumulative reward. Already from this definition we notice some important distinctions with the previous learning paradigms since reinforcement learning is not a static concept but a dynamic one. It differs from supervised learning and unsupervised learning because it is not focused on the present result, but instead it tries to maximize the summation of both current and future rewards.

In practice, reinforcement learning methods try to solve a given problem by taking subsequent actions in a given environment[63]. As consequence of such actions, the agent, meaning the model, receives from the surrounding environment negative or positive rewards which represent informative signals. Then, on the basis of these signals, the ML method improves its knowledge of the environment dynamics[64] and, after a proper number of iterations, approximately learns the optimal policy to follow, i.e., the optimal sequence of actions that maximizes the cumulative rewards.

We can conclude by highlighting the fact that reinforcement learning methods are the most powerful learning procedures among the ones presented, but they are also the most difficult to implement[65]. Therefore, it should not surprise us to know that supervised learning is by far the most widespread learning paradigm.

So far, we have introduced a lot of different machine learning methods to the point that someone could argue "which model should we use?" As already told, a thing such as a general answer applicable in every possible situation does not exist. Indeed, specific problems require specific solutions and the characteristics of the algorithm we aim to implement are a critical aspect to consider with this regard. However, there is some empirical and theoretical evidence that comes to our rescue. In particular, Banko and Brill (2001) proved that the performance of different learning methods on a natural language disambiguation task tends to converge with the increase of the input data[66]. Moreover, they showed that the performances of these algorithms increase all in a sort of monotonical way so that if you pick an "inferior algorithm", meaning a model that tends to display poorer performances than the others, and give it more data, then it looks like it will most likely beat every other "superior algorithm".

[61] From: "Machine learning algorithms – a review" by Batta Mahesh (2019).
[62] See for instance Sutton et al. (2018).
[63] With given environment we simply mean that the environment is well defined. It does not mean that the environment is fixed or constant over time, indeed it can change but following the specified rules.
[64] In other words, thorough these interactions the agent learns the state of the environment.
[65] This is the reason why we do not provide a list of such algorithms. Indeed, they vary a lot from company to company. A couple of examples are the Asynchronous Advantage Actor-Critic (A3C) developed by Google's DeepMind group and the Trust Region Policy Optimization (TRPO) or the Proximal Policy Optimization (PPO) from OpenAI.
[66] "Scaling to very large corpora for natural language disambiguation" by M. Banko and E. Brill (2001).

It is to be noted that since this original paper there have been a lot of different studies showing similar results. Therefore, we can say that what can really drive the performance and the quality of the learning procedure is, in large part, the amount of training data we can provide to the model, as we can see in Figure 4. To show the pervasiveness of this kind of algorithms in different fields and the fact they are multidisciplinary tools, we can mention that Figure 4 comes from a paper of Hanczar et al. (2020) where an artificial neural network is exploited in helping to take clinical decision.

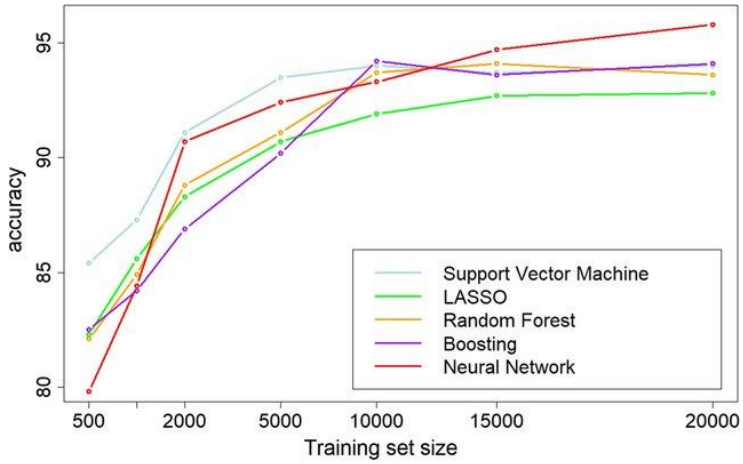

Figure 4 - Accuracy of different learning algorithms as a function of the training set size (source: BMC Bioinformatics)[67]

Which model should we use, then? Even if it is true that by increasing the training set size we can obtain similar results, until now we have not taken into account the specific nature of our problem. In particular, we need to remember that our goal is to develop a machine learning system able to provide correct and reliable estimates for the price of option contracts. Therefore, it is a regression problem. This information alone already enables us to shrink the number of models to consider, indeed it would not be efficient to rely on algorithms such as decision tree or random forest since they are more suitable for classification tasks. Furthermore, in our process we will feed the machine labeled input-output pairs, meaning we will exploit the supervised learning paradigm. Finally, given the particular features of our problem, meaning the multi-dimensionality and the high nonlinearity of the model, we will have to rely on an algorithm able to deal with such characteristics.

To draw a preliminary conclusion in order to summarize what has been said so far, we can state that while a large number of models exist for regression problems[68], they are not all applicable to the special case of option pricing. Regarding this specific task, the most frequently used machine learning algorithms are artificial neural networks (ANNs). As a matter of fact, neural networks have been found to perform well in a lot of applications where other models have failed. Especially, when it comes to forecasting financial market variables characterized by non-stationarity, non-linearity

[67] The figure comes from the paper "Biological interpretation of deep neural network for phenotype prediction based on gene expression" by Hanczar et al., BMC Bioinformatics, Vol. 21, No. 501 (2020).
[68] See for instance Hastie, Tibshirani, and Friedman (2009).

and high dimensions, such models have been proved to have a significant edge and to embed useful benefits. These are the reasons why from now on we will focus on such algorithms.

2.2 History of Artificial Neural Networks

Since we decided to rely on neural networks, it may be beneficial for us to understand the evolution and the main features characterizing these kinds of models so that we can get a better sense of what we can expect them to do.

First of all, we need to clarify that neural networks (NNs) are actually a pretty old idea, which had fallen out of favor for a while, mostly for hardware and theoretical limitations, but nowadays they undoubtedly have become the state-of-the-art technique for a lot of different ML problems. So, what is the basic idea on which NNs are based? We already said that machine learning algorithms are and have been developed trying to mimic intelligent behaviours which can be observed in nature[69]; also in this case it is the same.

The basic idea of neural networks was to develop an algorithm able to reproduce the brain functioning. We know that our brain is an amazing structure able to learn many different things: it can process images, recognize what we touch and hear, perform abstract thinking, and so on. Therefore, it seems that if we want to replicate it, we have to develop a lot of different software to perform all the required tasks. However, there exists a fascinating hypothesis according to which the brain is able to do all these things exploiting just one single learning algorithm[70]. If this is the case, it means that instead of needing to implement a thousand alternative programs, we just need to develop a system which teaches the machine how to learn by itself and how to process different kinds of data.

Having this in mind and considering that the structure of artificial neural networks is mainly shaped from the anatomy of natural neurons and from the neural system, it makes sense to start analyzing the features of such nerve cells.

[69] Consider for instance footnote number 52 for a short list of nature inspired algorithms.

[70] Many proofs have been proposed over time. For a detailed list see for instance the course of Machine Learning by Andrew Ng, Stanford University.

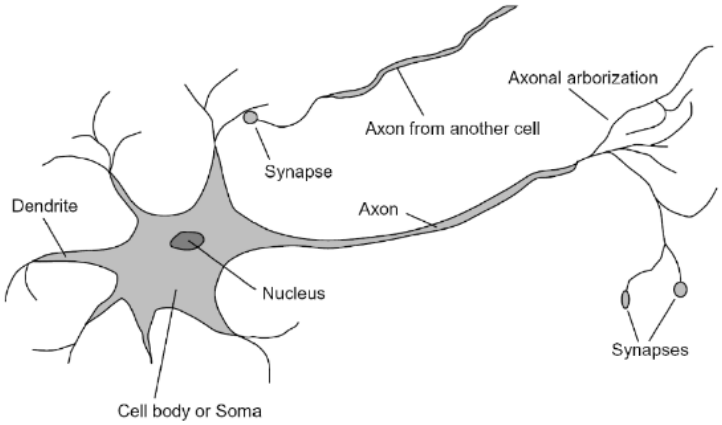

Figure 5 - Sketch of a human neuron (source: International Journal of Plant and Soil Science)[71]

As we can see from Figure 5 a single neuron has a cell body which is composed by three main parts: the nucleus, the dendrites and the axon. The nucleus is where the computations are made, the dendrites are the input wires which receive signals from the external environment, and the axon can be thought of as an output wire used to send messages to other neurons.

So, at a simplistic level a neuron is a computational unit that first gets electrical signals, also called spikes, from the outside thanks to the dendrites. Then this input information is aggregated through some biological procedure that takes place in the nucleus. Note that during this aggregation step the relative importance of these signals can be increased or reduced according to what the neuron "thinks" about it. Finally, the computed output is passed to other neurons through the exploitation of the axiom. Therefore, the way through which neurons communicate with each other is with pulses of electricity.

These continuous computations and transmits of messages are the way in which our brain works. However, single neurons alone are not able to achieve good results, so that in practice most of the strength of our brain is given by the huge number of connections that exist within these cells. The idea according to which it is the network of interconnected neurons to provide the larger part of the computational ability characterizing our brain, is called connectionism, and together with the biological structure of the neurons represents one of the two founding ideas that drove the developing of artificial neural networks.

From an historical point of view, the growth of artificial neural networks has been characterized by different phases. The first phase lasted from the early 1940s to the end of the 1960s. During those nearly 30 years the first rudimental implementations of neural networks were proposed. In particular, the theory for the development of a single artificial neural network, namely a perceptron, was proposed by McCulloch and Pitts (1943) in a paper that quickly became a milestone[72]. That first

[71] Figure from "Artificial neural network model for the prediction of the cotton crop leaf area" by Aboukarima et al., International Journal of Plant and Soil Science, Vol. 8, pp. 1-13 (2015).

[72] "A logical calculus of the ideas immanent in nervous activity" by W. S. McCulloch and W. Pitts (1943).

publication was rapidly followed by other studies among which the most important are the ones from Hebb (1949)[73] and Rosenblatt (1958)[74], which leaded to the creation of the first perceptron.

These researchers took the necessary steps to build the foundations on which the theory concerning the development of ANNs is based. Specifically, McCulloch and Pitts derived the mathematical transposition of the biological neuron defining the following essential properties that underly the structure of an artificial neuron:

- A natural neuron is an "all-or-none" process. This means that a neuron is a binary cell only capable of dealing with positive and negative results. In other words, it is able of classifying only two possible inputs: the presence or the absence of the electrical signal. Therefore, we can represent it, in mathematical terms, relying on logical values and Boolean functions.
- A certain fixed number of synapses must be excited at the same time in order to excite a neuron and produce an output. Moreover, this fixed number has to be independent from the previous activity and the position of the cell.
- The only significant delay within the nervous system is the synaptic delay.
- The activity of any inhibitory synapse absolutely prevents the activation of a neuron at that time. Therefore, if a neuron is excited though the use of an inhibitory synapse it can be deactivated.
- The structure of the network is stable and does not change with time.

We have said that the work of McCulloch and Pitts was essential in defining the mathematical framework of the problem. Indeed, this can be considered as the theoretical birth of the concept of perceptron; however, they neither provided a concrete model nor a structure to further develop, but only the core idea. Moreover, they primarily focused on the definition of a single neuron only partially considering the role of the network. Therefore, there was still a huge lack in the field.

With this regard, the studies of Hebb were extremely useful in clarifying the importance of connectionism. Indeed, Hebb studied the physiology of the nervous system, and he tried to find some sort of "*community-effect*" in how neurons are structured. In particular, he proved that "*some growth process or metabolic change*" in one or both cells takes place as a result of repeated transmissions across synapses. This simply means that the more a neuron "*dialogues*" with another, the more their connection strengthens. So, according to Hebb's model, the exchange of signals among different neurons is an extremely important aspect which cannot be ignored, since it is able to affect the network and its efficiency.

Finally, starting from the above-mentioned theorems and relying on the results of other research[75], Rosenblatt was capable of developing the first "hypothetical nervous system", i.e., a perceptron, in 1958, presenting the first mathematical formulation of such artificial structure.

[73] "The organization of behaviour, a neuropsychological theory".
[74] "The perceptron: a probabilistic model for information storage and organization in the brain".
[75] Other useful references are Kleene (1956), Von Neumann (1956) and Minsky (1960).

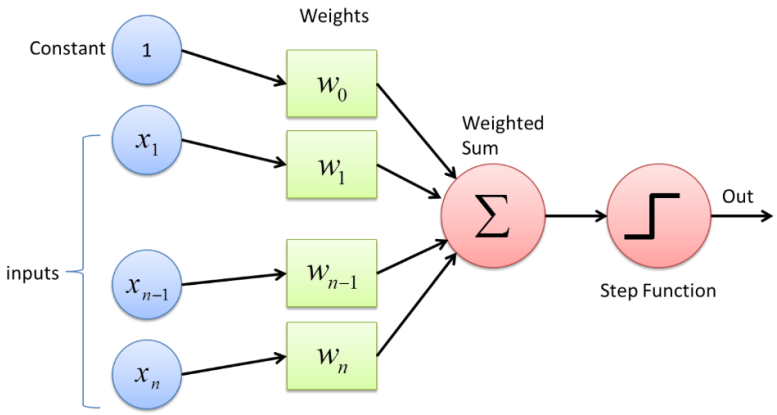

Figure 6 - Diagram of a perceptron (source: DeepAI.org)

Figure 6 represents the structure of the perceptron created by Rosenblatt. As we can see, we have a set of inputs that can be represented by a vector of real values: $(1^{76}, x_1, x_2, \cdots, x_n)$. These are equivalent to the electric signals the biological neuron receives. The information received from the external environment is then weighted. Indeed, we have to remember that also in natural cells the relative importance of each signal can be amplified or reduced. By assigning a greater or a lower weight the perceptron is able to modify the magnitude of the inputs, privileging the information that it considers to be most important. Once all the inputs have been weighted there is the aggregation step, which is the operation that is performed by the nucleus in the biological counterpart and consists in performing a weighted sum of the input values using the vector of real-valued weights. Finally, the output is released by means of a step function.

Note that since the output is provided through the use of a step function which can take only binary values, the perceptron can be seen as a sort of classifier[77].

From a mathematical point of view, it is possible to formulate the previous problem as follows:

$$o(\cdot) = \begin{cases} +1 \; if \; \sum_{i=0}^{n} w_i x_i > 0 \\ 0 \qquad elsewhere \end{cases}$$

Where x_i represents the generic input and x_0 is set by convention equal to 1 and is known as bias. w is the vector of the weights by which it is possible to compute the weighted sum, also called weighted aggregation. Lastly, $o(\cdot)$ is the *transformation/squashing* function.

The functioning of a perceptron is quite straight forward. Once the weighted inputs have been aggregated the result is compared to a given threshold, which is zero in the simplest case[78]. Then, if the result is greater than such limit value the perceptron provides an answer, otherwise it gives

[76] Note that x_0 is usually assumed to be equal to 1 in order to account for the bias. To have a better understanding, suppose we aim to perform a linear regression. If we do not estimate the constant term the result will always be biased. Therefore, the bias term has the same role in a perceptron as the one of the constant in a linear regression.

[77] See for instance Alpaydin (2020).

[78] Note that it can be easily generalized introducing a general value K: $o(\cdot) = \begin{cases} +1 \; if \; \sum_{i=0}^{n} w_i x_i - \mathrm{K} > 0 \\ 0 \qquad elsewhere \end{cases}$

another output. At this point, we can highlight the fact that this former model had some important drawbacks. In particular, the transformation function cannot be differentiated, and it is not continuous[79]. If we keep in mind the task we are trying to tackle, namely the pricing of financial options, we immediately understand that these are huge limitations. In any case, we will see how to cope with them in a moment.

Note that in our formulation the only unknown is the vector of weights. Therefore, the goal of the perceptron can be summarized by saying that it has to estimate in a proper way such variables. Now, from what has been said, a question should follow: how does the model correctly determine these weights? The answer was provided directly from Rosenblatt. Indeed, he developed a learning rule for the artificial neuron known as *Delta rule* or *Training rule*[80].

The Delta rule is an iterative algorithm which represents a key element in the field of machine learning. This is a derivative free procedure which works as follows:

$$Randomly\ initialize\ the\ vector\ of\ the\ weights$$

$$for\ k = 1, \dots, K$$

$$for\ n = 1, \dots, N$$

$$for\ j = 0, \dots, J$$

$$w_n^{k+1} = w_n^k + \Delta w_n^k = w_n^k + \alpha(y_n - o_n)x_{j,n}$$

The first step consists in randomly generating the weights. Then the iterative procedure starts. In particular, the procedure will be repeated K times, where K is the number of iterations chosen by the user. N times, meaning it will be reiterated for every input-output training example[81], and J times, i.e., we keep into account each different weight.

As we can notice, the value of the weights changes in each next iteration and it is determined as the current value plus an increment, Δw_n^k, which represents a percentage of the difference between the true value of the output and the computed one. We say it is a percentage of the difference, indeed we can see that this increment is multiplied by two terms: the input value $x_{j,n}$ and a coefficient α known as learning rate. This learning rate is the value that determines the speed of the learning, and it is a positive, small value.

Now it is possible to make a further observation: thanks to the Delta rule the perceptron is able to modify the weights adjusting them according to the information provided by the data. However, are we sure that there is a convergence? Meaning, can we say that the Delta rule is effectively capable of detecting a vector of weights such that the algorithm is able to learn the analyzed process? Also in this case Rosenblatt was capable of presenting a solution deriving the so-called perceptron convergence theorem: "*let* $\left((x_{1,j}, \dots, x_{n,j}), y_j\right)$ *with* $j = 1, \dots, m$ *and* $y_j \in \{-1, +1\}$ *for all j, be a set*

[79] Indeed, the step function assigns the value of 1 x is greater than 0, and -1 or 0 if x is lower than the origin.

[80] Notice that it took over 15 years to find the proper way to "code" this algorithm. From the theoretical definition of perceptron provided by McCulloch and Pitts (1943) up to its mathematical formulation by Rosenblatt (1958).

[81] Notation: with y_n we mean the generic real output observed in the available data. While o_n represents the generic output originated by the model.

of linearly separable instances, then the Delta learning rule terminates the updating of the weights w_i*, with* $i = 0, \dots, n$*,after a finite number of iterations*" Rosenblatt (1958).

It is possible to prove that the presented training rule does work, and moreover it is robust, meaning that even in presence of noise it is able to correctly provide an answer[82]. See for instance the example provided by Rosenblatt about the classification of the Boolean function AND.

We started this section by saying that the development of neural networks has been characterized by different phases and periods of great innovation were followed by moments of "abandonment" of these instruments. But if the perceptron is such an amazing structure, why has it been forgotten for a long time? The reason is simple, a single perceptron is only able to classify "linearly separable instances".

In 1969 Minsky and Papert published a fundamental paper[83] which proved that if we do not consider the Boolean function AND but for instance another simple logical function, namely the XOR, or Exclusive OR[84], the perceptron is no more able to learn it. In other words, it fails in correctly classifying the outputs since they are not linearly separable. This is the main drawback of the single perceptron and the reason why from 1969 up to 1980 the studies about machine learning and neural network crashed down. This period is now known as "AI winter" to underline the dramatic situation artificial intelligence had to face.

In order to overcome the limitations of these algorithms many possible strategies were proposed. In particular, in the early 1980s an idea had taken over the others: connectionism. Indeed, with the development of artificial neurons, we only addressed half of the problem, and we completely forgot about the importance of the other half: the network. As already said, most of the power of our brain is not given by the specific structure of a single neuron, but it is provided instead by the billions of connections that exist between our neural cells. Therefore, the solution they proposed lied on the idea of combining the concept of perceptron and the concept of connectionism in order to develop a network of perceptron, namely a multilayer perceptron (MLP).

Unlike the perceptron for which it is possible to identify a date of birth, artificial neural networks do not have a precise and unique "birthday". Indeed, it is possible to find mentions of such a concept in a lot of different studies. However, there are some fundamental papers and empirical results which cannot be ignored. In particular, Cybenko (1989) derived a theoretical solution[85] stating the reason why MLPs are nowadays known as universal function approximators: "*[the] networks with one internal layer and an arbitrary continuous sigmoidal function can approximate any continuous function with arbitrary precision providing that no constraints are placed on the number of nodes or the size of the weights*"[86].

[82] Of course, if the noise is too high the perceptron is not able to perform a right classification. However, in normal situations where we use reliable data it works perfectly.

[83] "Perceptron: an introduction to computational geometry" by M. Minsky and S. Papert (1969).

[84] The XOR function is an extension of the function OR. In this case, the output of the logical function is true if and only if one of the two inputs is true and the other is false.

[85] "Approximation by superpositions of a sigmoidal function" by G. Cybenko (1989).

[86] Note that, even if the original Universal Approximation Theorem relied on the concept of sigmoidal functions, Lensho et al. (1993) proved that a huge set of alternative functions can be used, we will see it in the next paragraphs.

Let us clarify it better. First of all, it is necessary to highlight another point which is questionable in the original formulation of the problem: the transformation function. Indeed, the first perceptron relied on a step function. As already mentioned, this function has serious limitations since it is neither differentiable nor continuous and, being binary, it allows us to make only strong classifications, $(0, 1)$. Therefore, alternative squashing functions were proposed, and it turned out that sigmoid functions and s-shaped functions[87] in general were the ones that provided the best results.

Now, it is possible to look at Cybenko convergence theorem from a more formal point of view. It says that if we consider any continuous sigmoidal function σ, the finite sums of the form:

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_i^T x + \theta_j\right)$$

are dense in $C(I_n)$, meaning that given any $f \in C(I_n)\ and\ \epsilon > 0$ there is a sum, $G(x)$, for which:

$$|G(x) - f(x)| < \epsilon\ for\ all\ x \in I_n$$

Therefore, the neural network is able to approximate any unknown continuous function $f(x)$ with any arbitrary desired level of accuracy ϵ. In other words, if a relationship between the input and the output does exist, the ANN properly trained will always be able to detect it.

Another extremely important result was the one found in the same year, 1989, by the Defense Advanced Research Projects Agency (DARPA) which is one of the most important agencies of the U.S. We have to remember that the perceptron was not able to classify the XOR function. However, DARPA proved that relying on MLPs we can make the algorithm able to solve such a problem. In particular, they showed that when increasing the amount of perceptron in the network, which are called layers, the system is capable of separating the space of the problem not only linearly, but also in convex and more complex ways. Therefore, the classification ability of the network improves as the number of neurons grows.

A third milestone was set by the development of the "*error backpropagation algorithm*[88]" by Rumelhart et al. in 1986[89]. Similarly to the Delta rule previously presented, this is a repetitive algorithm used to train the network. Specifically, the error back propagation algorithm is an iterative numerical optimization approach based on the gradient descent method, which is a first-order derivative optimization technique for detecting points of relative minimum of given functions. It works as follows:

[87] This kind of functions are continuous, differentiable and bounded.

[88] It is also called Generalized Delta rule.

[89] "Learning representations by back-propagating errors" D. Rumelhart, G. E. Hinton, and R. J. Williams (1986).

$Randomly\ initialize\ the\ vector\ of\ the\ weights$

$for\ k = 1, \dots, K$

$Compute\ the\ error\colon\ E = \frac{1}{2}\sum_{i=1}^{K}(y_i - o_i)^2;$

$Solve\ the\ minimization\ problem\colon\ min_{w_1,\dots,w_W}\ E$

$for\ n = 1, \dots, N$

$for\ j = 1, \dots, J$

$$w_n^{j+1} = w_n^j + \Delta w_n^{j+1} = w_n^j + \alpha \frac{\partial E}{\partial w_n}$$

As we notice, this process is extremely close to the Delta rule, but it presents some important modifications. In this case indeed we compute also the summation of the squared errors quantified as the difference between the true output and the one predicted by the model, and we try to minimize it before adjusting the weights. We will see it in a more detailed way in the next sections.

Thanks to all these results the development of artificial neural networks experienced a second phase of huge growth which lasted about 15 years until the mid-1990s. Then for various reasons, mainly related to hardware limitations, their popularity diminished in the late 90's, but they quickly bounced back to the point that nowadays they are, as previously mentioned, the state-of-the-art technique for various machine learning applications.

The major reason for this resurgence is linked to the progress of computer science. Indeed, neural networks are in a sense expensive algorithm when compared to other machine learning models; therefore, they require computers fast enough to deal with large size datasets. This last phase, whose beginning can be traced back to the mid-2000s and which has continued without any interruption ever since, is characterized by the development of new ANN models and improvements that allow for a more efficient implementation. We will present some of them in the next paragraphs and chapters.

2.3 Artificial Neural Networks

Now we should have a clear understanding of the idea behind ANNs and of the historical development that characterized these machine learning algorithms. Let us try to better understand their functioning and the main features that distinguish them from alternative machine learning models.

We know that artificial neural networks are basically massive computational models that, by imitating the structure of a human brain, try to map the input values, also called features, into one or multiple outputs[90]. Therefore, we can say that the main goal of such models is to approximately learn the function that connects the vector of the inputs to the corresponding outputs. Note that

[90] See for instance the paper by Xu et al. (2019).

we will focus on supervised neural networks, meaning we will make available for the system a training set composed of labeled input-output pairs.

Relying on a bit of notation, it is possible to provide a more formal representation of such processes. Indeed, we want to detect a function f such that $f: X \rightarrow Y$ where X is the vector of the features and Y is the related output. In order to perform this task, we will rely on a training set which is composed of all the observable features $(x_1, x_2, \dots, x_N)$ and the true output $(y_1, y_2, \dots, y_N)$. We have to highlight that both the inputs and the outputs may be multidimensional. Therefore, the generic feature x_i or the generic result y_j can be seen respectively as a vector of inputs or a vector of outputs.

Once the network is fed with a new, additional, and unobserved value x, the output that will be produced by the model will be the one ensuring the lowest prediction error, or, in other words, the one that maximizes the coherence with the relationship already detect by the model starting from the training set. This can be expressed in two alternative ways:

- $f(x) = argmin_{y \in Y} E$, where E is the error metric we choose to exploit[91].
- $f(x) = argmax_{y \in Y} S$, where S is a satisfaction measure which represents the similarity between the predicted output and the true one.

We already presented the functioning of a single artificial neuron[92], therefore at this point we just need to mention that in case of neural network each perceptron is known as “node”.

We know that a single neuron performs a dual task. On one side, it receives the inputs from the external environment or from other neurons and does some computations. On the other side, it transmits this weighted sum by means of a so-called activation function, which we have to remember is usually a s-shaped function such as a Sigmoid. So, each node receives a set of inputs, that always include the bias term, and performs some sort of operations, usually a linear transformation, in which it exploits the weights associated with every input.

Then the result is transformed by means of the activation function. Once all the outputs have been collected, it is possible to evaluate the goodness of the model by comparing the predicted results with the true available output, and then the network can adjust the weights rely on this information. The process is iteratively repeated until a satisfactory result is obtained or another stopping criterion is met.

Question: when a result is considered to be satisfactory enough? Or what can we say about such stopping criteria? To answer these doubts, we need to introduce some preliminary notation and a few additional concepts.

First of all, we need to clarify that in order to properly assess the performance of a machine learning algorithm, we cannot usually rely on traditional evaluation methods, but instead we have to define new specific metrics. A primary distinction that it is possible to make regards the available data we

[91] Note that the most common error measure is the Mean Square Error or better its square root, i.e., the Root Mean Square Error.

[92] See the Perceptron proposed by Rosenblatt and described in paragraph 2.2.

provide the model with. Indeed, in case of machine learning systems the dataset has to be divided into three[93] different and mutually exclusive subsets which are respectively:

- The training set is the sample of data used to fit the model, meaning the actual dataset we use to teach the model about the relationship existing between inputs and outputs. For this reason, it is a good practice to make it the largest among the three subgroups. Having said that, it should be clear that the training set affects the model directly.
- The validation set has a particular function since it is used to provide an unbiased evaluation of how the model fits the training dataset while we are tuning the hyperparameters, i.e., during the process in which we select the best characteristics of the machine learning algorithm[94]. Thus, the purpose of such dataset is to understand how well the model behaves when exploiting different data never seen before. It affects the model, but only indirectly.
- The test set is the sample of data used to provide a correct evaluation of the final proposed model, of its ability to properly fit the data, and so on. It is usually the smallest part of the available data which is exploited to perform such a task. Note that this is only used at the end of the training process and therefore it does not affect the model specification. We can rely on it when comparing competing models.

According to the performance of the system and how well the model fits the data both in in-sample and out-of-sample analyses, meaning in the analyses made in the training set and in the test set, it is possible to draw interesting conclusions which help us to understand the behaviour of the network and the steps we can take to improve it. In particular, we can face three alternative scenarios, two negatives and one positive. Underfitting and overfitting are the two sides of a coin when it comes to negative scenarios, while a positive outcome means the presence of a good and balanced fitting of the data.

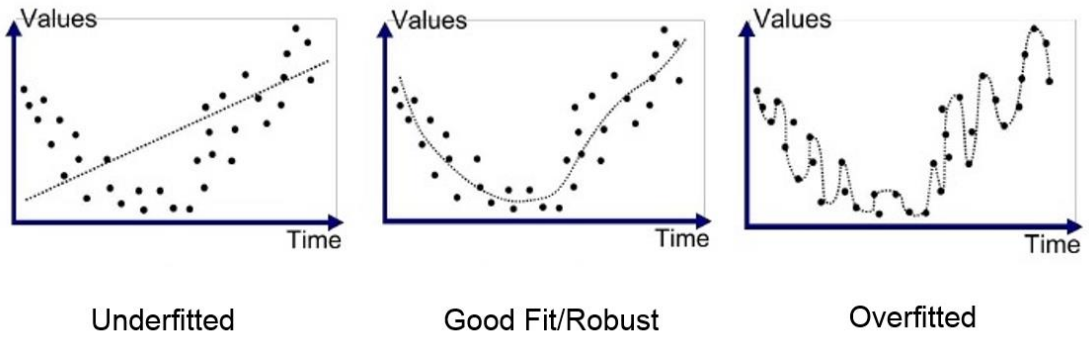

Figure 7 – Overfitting versus Underfitting (source: TheStartup.com)[95]

As we can derive from Figure 7, a model will be said to be overfitted when it has learned the training set too well. In such a case, it may mistake the noise presented in the data for actual information.

[93] Or at least two in case we are considering unsupervised or reinforcement learning paradigms.
[94] For instance, if we consider a decision tree the hyperparameters to adjust are the number of nodes and branches.
[95] Figure by K. Hoffman (2021) "Machine learning: how to prevent overfitting" published by The Startup.com.

Therefore, in case of overfitting, the in-sample performance will be extremely good while the out-of-sample-performance will be poor, meaning the model does not generalize well when additional unseen data are provided.

Underfitting is the opposite case. When the model has not been sufficiently trained it usually cannot understand the structure that underlies the data, and it fails to capture relevant patterns that a properly trained model would have detected. Therefore, if the network underfits both the in-sample and the out-of-sample analyses will provide unsatisfactory results. In both cases, the system cannot effectively adapt to new data and, considering that our main objective is to perform some future analysis, we have to solve these drawbacks. According to Bishop (1995)[96] "*The goal of network training is not to learn an exact representation of the training data itself, but rather to build a […] model of the process which generates the data*." Then he continued by saying that it is important for the applied method "*to exhibit good generalization, that is, to make good predictions for new inputs.*". So, we do not want to learn the data but the process generating them, however since in case of both underfitting and overfitting generalization is not possible, we have to understand how to cope with these two kinds of problems.

In order for us to do this, it is sufficient to think that if we detect overfitting, we are probably including too much information in our system which has become too complex; in a similar situation simplifying the model structure may be the best way out. This can be done removing some features, i.e., decreasing the training set size, or some explanatory variables which in reality do not add useful information (maybe because they are highly correlated with the others).

In presence of underfitting the solutions we have to pursue are the opposite. The model is indeed too simple, meaning we can try to solve such a problem providing additional observations (if possible) or adding some hyperparameters, i.e., we both try to train the network better and to specify a more complex and hopefully more efficient model.

Related to the concepts of overfitting and underfitting there is the so-called *bias-variance trade-off*. The idea is the following: when a model is too simple it is not able to learn the data structure well. This means that when underfitting occurs the model's predictions will be characterized by a small variability, but at the same time some of them will be more biased. In case of overfitting instead the variance associated with each prediction is very high, but the bias tends to be extremely low. A well-defined machine learning model should achieve both a low variance and a low bias since this ensures the predictions to be corrected and the performance high.

By adjusting the hyperparameters and the splits of the dataset between training, validation, and test sets, in other words by modifying the complexity of the model, we can reduce the variance and/or the bias of the network. The problem is that this is a trade-off, therefore when we try to minimize the variance the bias rises, and vice versa[97]. Of course, our goal is to detect the best possible balance between these two errors.

In Figure 8 it is possible to see a graphical representation of the existing relationship between bias and variance and how they contribute to the total prediction error of the model.

[96] "Neural networks for pattern recognition" by C. M. Bishop published by Oxford University Press (1995).

[97] With this regard it is possible to consult, for instance, the works of Fortmann-Roe (2012) or Gèron (2019).

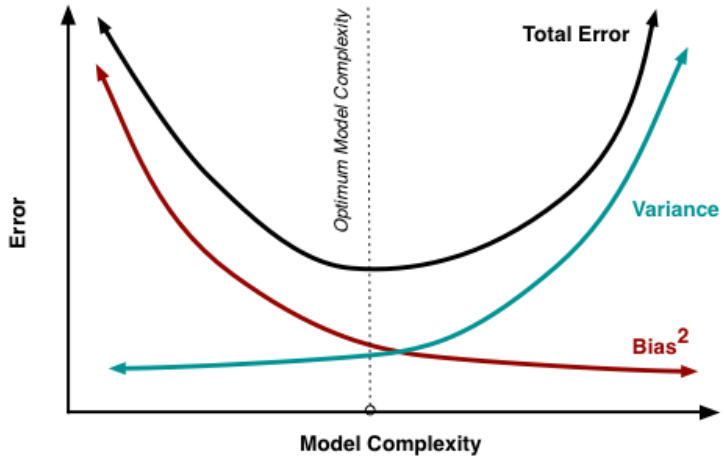

Figure 8 - Bias and Variance contribution to total error (source: Scott.Fortmann-Roe.com)[98]

From the definition of the bias-variance trade-off and the concepts of overfitting and underfitting one could ask: how is it possible to train the network in such a way that it exhibits good generalization capabilities? To achieve such a result, it is possible to rely on a lot of different techniques which will be seen in more detail later on during the thesis. However, a short list of these methods includes cross-validation, early stopping criterions, and regularization techniques.

Going back to neural networks, we can say that what is truly interesting about such models is the fact they combine the structure of the perceptron with the idea of connectionism. In order for us to better understand how these two concepts merge together, we need to define a new "unit" in the structure of ANNs, the so-called layer. Indeed, a set of perceptrons, here called nodes, on the same "level", can be grouped together to become a layer. It is then possible to define three different types of layers according to their position in the network:

- Input layer. It is the first level of the network and consists in the perceptrons that receive signals from the external environment.
- Hidden layer. First of all, we need to say that, according to the number of hidden layers, it is possible to distinguish between machine learning and deep learning systems. Indeed, neural networks exploiting multiple hidden layers are part of the so-called deep learning, a branch of ML characterized by powerful algorithms. Note it is the presence of hidden layers that makes the system intelligence meaning that if we do not consider this intermediate level we end up with a single artificial neural network, which is a simple perceptron, and therefore we end up with a structure unable to learn complex functions, for example the XOR.
  Regarding the existing connections, we can highlight that the nodes of the hidden layers receive signals from neurons and release their output to other different neurons, so that they have no interactions with the external environment.
- Output layer. It is the last level of the network, and it provides the final result to the "outside".

[98] Figure from Fortmann-Roe.com "Understanding the Bias-Variance tradeoff" (2012).

Depending on the architectural structure, also called topology, that is exploited by the net, in other words depending on the way in which different nodes are connected, many alternative kinds of networks can be defined. The most common ones are the following:

- Auto associative ANN. In this structure, there are usually two levels of nodes, and each single node is linked only to nodes of the other level. However, there is not a precise direction in the flow of the information.
- Fully connected ANN. Each node is connected to all the other neurons.
- Unstructured ANN. There are more or less random links between the various perceptrons.
- Feedforward ANN[99]. It is similar to the auto associative neural network, in the sense that different nodes are organized on different levels. However, in this scenario, the nodes of a specifical level are connected only with the nodes of a subsequent level and there is a sort of linear progression, meaning the information can only flow in one direction, i.e., there are no nodes coming back.

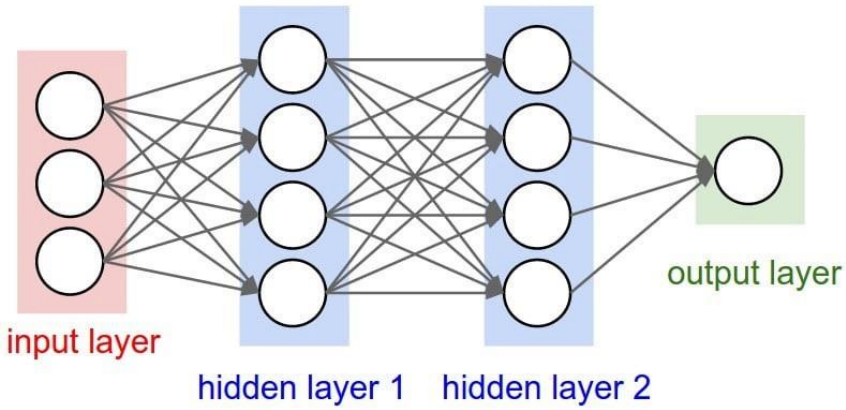

Figure 9 - Example of a feedforward multilayer neural network characterized by two hidden layers (TowardsDataScience.com)[100]

Since the feedforward ANN is the architecture that we will employ in the practical implementation presented along the thesis, let us focus a bit more on this specific structure, which can be observed in Figure 9.

A MLP feedforward ANN has an input layer made of $n + 1$ neurons, with $n \geq 1$ being the number of input features we are providing to the network. Note that we add 1 to account for the bias unit. Then there are one or more hidden layers constituted by what are usually called "thinking neurons". In particular, each of these hidden layers contains $h_j \in N^+$ hidden nodes, with $j = 1, \dots, H$, where H is the number of hidden layers. Observe that the number of neurons of the *j-th* hidden layer is expressed as $h_j + 1$.

[99] It is worth mentioning that deep learning methods directly derive from an extension of feedforward neural networks. See for instance Zhang et al. (2017).

[100] Figure from "Coding Neural Network – Feedforward propagation and backpropagation" by Dabbura, I. published by TowardsDataScience.com (2018).

It is interesting to notice that both the number of hidden layers (H) and the number of neurons for each hidden layer (h_j) are chosen by the user, therefore these are two hyperparameters we can adjust in our optimization process in order to maximize the performance and to find the best balance in the bias-variance trade-off.

Finally, the output layer is the one constituted by the artificial neurons which provide the result to the external environment. In particular, there are q of such neurons where $q \geq 1$ and equal to the number of outputs we need. The feedforward structure can then be seen in the fact that each layer, except for the output layer, is fully connected only with the next layer, as we can see in Figure 9, and the flow of information is forward, meaning it moves on starting from the input neurons and arriving at the output without ever coming back.

To highlight the complexity of such models, we have to understand that to each "arrow" of the network, which represents the flow of information from one neuron to the others, a weight, the model has to compute and improve over time, is associated. Therefore, a MLP feedforward neural network is characterized by a number of weights equal to:

$$W = (n+1)h_1 + \sum_{i=1}^{H-1}(h_i + 1) + (h_H + 1)q$$

If we consider the simple network presented in Figure 9, we can compute the number of weights the model has to estimate which is equal to $(3+1)4 + (4+1)4 + (4+1)1 = 41$.

It should be clear that, with respect to the design of the network, there are many possible parameters to tune. In particular, we have to choose the number of hidden layers, the quantity of input and output neurons, which are given by the nature of the investigated problem, and the number of hidden nodes. Of course, the higher the number of the hidden units, the greater the "thinking" ability of the model but, at the same time, by increasing these hyperparameters we make the required computational capability grows exponentially. Some of these problems can be easily fixed, for instance we already said that the number of output nodes depends on the characteristics of the specific work we are performing. However, the determination of the number of hidden layers and hidden neurons is a more complex task to deal with.

We have to say that, in reality, there is no such a thing as a general answer or a universal procedure to follow and instead most of the times we will have to rely on a trial-and-error sequence of actions. In any case, as a rule of thumb, it is possible to fix an upper limit for the number of the total weights, and consequently for the number of neurons, which is a good practice to respect. This is called Training to Weights Ratio (TWR) and it is defined as follows:

$$TWR = \frac{input - output}{w} \geq 10 \quad from\ which\ it\ follows\ that \quad w \leq \frac{input - output}{10}$$

Now, considering more in detail the structure of each neuron we can say that in general the nodes of both input and hidden layers are characterized by a logistic, usually sigmoidal, transformation function while the ones of the output layer rely on a linear transformation function. The idea is that the computations have to be performed in the first steps of the network, whereas the last layer has only the task of releasing the result already computed without modifying it.

Notice that it is possible to demonstrate that the activation function plays an essential role in the ability of the network to learn a relationship, indeed, if we consider for instance a constant function, the network becomes only able to perform simple linear regression, losing most of its predictive power. Therefore, it is essential to spend few more words regarding this concept which is a key part of the design of neural networks.

Activation functions, also known as transfer functions or squashing functions, are applied to the aggregated result computed by each node. Their main goal is to transform this output into a more useful and manageable result, that is then sent to the subsequent neuron. Theoretically, it is possible to use any kind of function, both linear and nonlinear, but as already mentioned, in order to ensure the full computational capability of the network, linear functions have to be exploited only in the output node.

Indeed, on one hand, the choice of the activation function in the hidden layers will determine how well the network is able to learn the data. "*In order to access to a much richer hypothesis space that would benefit from deep representations, you need non-linearly activation functions*"[101] (Chollet, 2017). On the other hand, the choice of the activation function in the output layer will define the types of predictions that the model can make. Therefore, if we want our system to be able to provide an unbounded, real, scalar result, the choice of a linear function is perfect. That is why they are a so popular alternative for the output layer.

Nowadays there are some functions that have been proved to provide the best results when used as activation functions in machine learning systems and have become the state-of-the-art techniques. These are the *Sigmoid function*, the *Rectified Linear Unit Function* (ReLU), and the *Hyperbolic Tangent function* (Tanh) which can be seen in Figure 10:

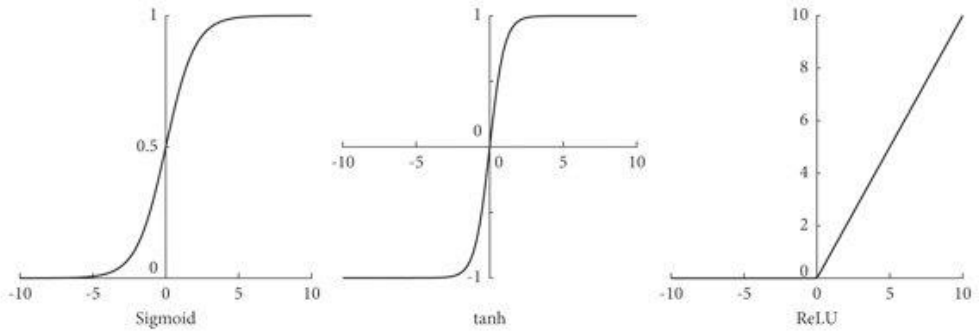

Figure 10 - Most common activation functions

We have to highlight that activation functions are typically defined on the line of real numbers, i.e., their domain is $\mathbb{R}$, they are all nonlinear and usually differentiable, meaning that their first-order derivative can be computed. This is a fundamental condition since it is necessary to apply the error backpropagation algorithm, which requires to compute the derivative of the error in order to update

[101] From "Deep learning with Python" by F. Chollet (2017).

the weights of the model; we will see it in detail in a moment. However, it can be noticed from the graph that the ReLU function is not differentiable when $x = 0$, so we have to use a little trick by "manually" including that point.

The mathematical formulations of these functions are the following:

- $Sigmoid = \frac{1}{1+e^{-x}}$
- $Tanh = \frac{1-e^{-2x}}{1+e^{-2x}}$
- $ReLU = \max(0, x) = \begin{cases} x \ \ if \ x \geq 0 \\ 0 \ \ if \ x < 0 \end{cases}$

The Sigmoid function, also called logistic function, can take any real value as input, while the output values range in the $(0, 1)$ interval. In particular, the larger the positive input provided, the closer the result will be to 1. Alternatively, the smaller the value, i.e., the more negative it is, the closer the output will be to 0.

The Tanh is extremely close to the Sigmoid: they are both s-shaped, they behave in a similar way, and they can take any real number as input. However, the output lies in a different range, $(-1, 1)$, which makes the Tanh function less steep than the Sigmoid.

Lastly, the ReLU activation function is probably the most used one thanks to its simplicity and effectiveness. Indeed, despite the non-differentiability of the zero point, which can be easily solved, it is possible to prove that such a function tends to provide robust results, less susceptible to other problems that plague alternative activation functions[102].

The question that rises now is about how we can choose which activation function to exploit. To begin, we have to point out that a neural network will almost always rely on the same activation function for all its hidden layers. Indeed, it is very rare to change it from one layer to the others. In any case, there is not a general consensus about which function we should use, so that the choice is mostly determined by empirical analyses, experiments, and depends on the specific nature of the problem we are dealing with. However, it is possible to mention some previous studies and the empirical results they derived.

Traditionally, the Sigmoid was the default activation function but through the mid to late 2000s the Tanh function took over. With this regard, Karlik and Olgac (2011) proved that the Tanh function is a proper choice for achieving high accuracy and good performance in many MLP implementations. On the other side, different studies showed that ReLU was able to outperform other activation functions in a lot of applications[103]. In any case, the fact that there are not too significant differences between these functions make them all a suitable choice, therefore it is better to empirically test which one provides the best performance for the specific case we are considering.

The last important elements we have to understand to conclude the definition of the structure of artificial neural networks are the concepts of gradient descent and backpropagation. These are two strictly connected ideas that refer to how machine learning algorithms, and neural networks specifically, learn. Therefore, we have to exploit such techniques in the optimization process of any

[102] Like for instance the vanishing gradients that compromise the ability of a model of being trained.
[103] See for instance Sharma et al. (2020).

ANN. In particular, we already know that the gradient descent method is a first-order derivative optimization technique for detecting points of relative minimum, while the error back propagation algorithm is an iterative numerical optimization approach which relies on the gradient descend[104]. Let us try to focus a bit more on them.

First of all, we need to derive a so-called loss function, also known as objective function, cost function or error function. This is a function representing the difference between the output produced by our model and the true output which is observable in the data. Therefore, they are used to evaluate candidate solutions and to compare and rank competing models by quantifying the errors produced. Moreover, "*It is important […] that the function faithfully represents our design goals. if we choose a poor error function and obtain unsatisfactory results, the fault is ours for badly specifying the objective of the search*." (Reed, 1999)[105]. Of course, our goal is to try to minimize such values relying on an efficient network design made of a proper hyperparameters optimization and a good and effective training procedure.

As we can imagine, there are many possible alternative loss functions among which we can choose, and since identifying the best objective function is a hard task, usually the best way to proceed is to rely on many different functions at the same time. The most common error functions are the Mean Squared Error (MSE), the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE), the Mean Absolute Percentage Error (MAPE), and the R^2. We will define them during the practical implementation presented in Chapter 4.

Going back to gradient descent and backpropagation we can say that since our goal is to minimize the loss function, we need to detect the global minimum; by relying on one of the above-mentioned error functions we are sure that such a point exists because these are all convex functions.

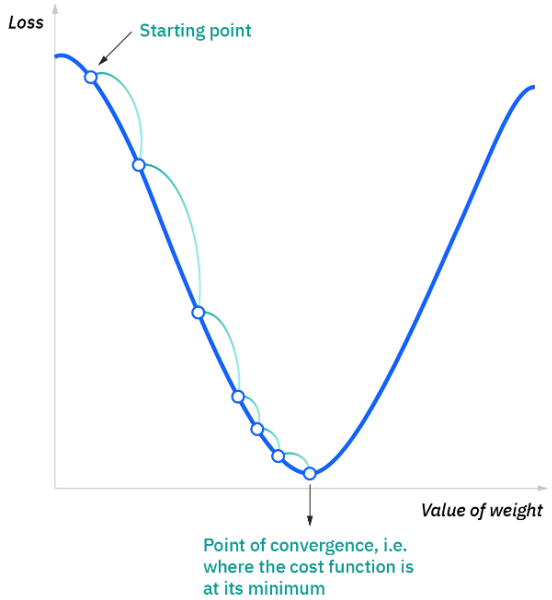

Figure 11 - Example of Gradient Descent (source: IBM Cloud Education (2020))

[104] Remember it was developed by Rumelhart, Hinton and Williams in 1986 and it is also called Generalized Delta Rule.
[105] "Neural smithing: supervised learning feedforward artificial neural networks" by R. Reed (1999).

If we plot a properly defined[106] loss function, we get something similar to Figure 11[107]. As we know, at the beginning our network will assign randomly generated weights to the features. Since this is a randomly assignment, we will usually not obtain good results at the beginning; as we can notice in our example, the starting point is indeed far away from the minimum. Now we clearly see that if, in our case, we increase the value of the weights, we can minimize the loss function. But how will the machine be able to do so? We need to optimize the weights to reduce the error, meaning we simply need to define a way to measure how the error modifies itself according to changes in the weights so that the network can understand what useful changes are.

To get this result all we have to do is to compute the derivative of the loss with respect to each weight, which will be called gradient:

$$Gradiend = \frac{dE}{dw}$$

Where E is the error and w is the specific weight considered.

Notice that it will always be the "negative gradient" to show the directions along which the weights should be moved to optimize the objective function. Therefore, it is this quantity that guides the network learning procedure and "tells" the model if it has to increase or reduce a specific weight. So, relying on this concept, the model learns the direction in which it has to move. However, it also needs to know by how much it should move, meaning how much it should modify the weights.

This is determined by the so-called learning rate, α. In particular, if this value is too large the parameters will change too much each time and they may fail in detecting the optimal values that allow the loss function to converge to its minimum. At the same time, if the learning rate is too low the model will only be able to take tiny steps and the learning procedure will require a huge amount of time to reach the minimum. Obviously, neither the first case nor the second scenario is desirable. Therefore, we need to fix a proper learning rate in order to ensure the model to be effective and efficient[108].

Having said that, it is possible to express the way in which each weight is updated over time, i.e., the method we use to teach our feedforward neural network, as:

$$w_i(t) = w_i(t-1) - \alpha * \left|\frac{dE}{dw}\right|$$

We have to remember that this is exactly the Generalized Delta rule, namely the error backpropagation algorithm, and it is an efficient way to train ANNs. As we see, the *i-th* weight at time t is a function of the previous weight, time $(t-1)$, and the gradient weighted by the learning parameter α. This means that with each iteration, called epoch, the model adjusts the weights according to the information provided by the gradient value. Note that the number of epochs is another parameter which is chosen by the user and therefore it is an hyperparameter whose optimization can help improving the general performance of the network.

[106] Meaning an objective function which is convex in shape. So that we are sure a global minimum exists.

[107] Of course, according to the specific problem we are considering and the available dataset we can observe more complex situations, but the basic idea is the same.

[108] See as an example Gèron (2019).

In particular, the backpropagation technique works by computing the gradient of the error function with respect to each weight and it does exploit the chain rule in this process; the chain rule is a mathematical technique used to express the derivative of the composing of two differentiable functions in terms of their single derivatives. Therefore, the network computes the gradient one layer at a time, and it iterates it backward starting from the last layer in order to avoid redundant calculations. In other words, this is a dynamic process.

The problem is that there are some drawbacks associated with such a technique, the most notable being the fact that there are no guarantees that the backpropagation algorithm will be able to find the global minimum of the loss function. Indeed, relying on derivative computations, the model may end up being stuck in points of relative minimums, since also these kinds of points are characterized by a null first-order derivative, i.e., the learning procedure stops when they are reached. However, it has been proved that in practical implementations this is not a major problem, since it is possible to exploit random initialization of the weights and to perform multiple optimizations[109].

This concludes the introduction of machine learning and neural networks. The following chapters will focus on presenting the analyzed dataset, its main features, and the proposed model to solve the option pricing task we want to deal with.

[109] See for instance "Efficient backprop" by Yann LeCun et al. (1998).

Chapter 3 Data analysis

Chapter 3 has two objectives; on one side, it aims at presenting the structure of the Italian equity market and the main characteristics of option contracts having as underlying the FTSE MIB Index. On the other side, it wants to present the dataset that has been used in our experiment. In doing so, it adopts the following form: section 3.1 describes both the Italian stock market and the Italian option market, focusing, in particular, on their composition and how they are computed and managed. Paragraphs 3.2 and 3.3, instead, explain the structure of the dataset, what features are available and which ones will be used, and in general the data manipulation techniques that have been performed to make the learning process easier and more efficient for the network. Moreover, in section 3.3, it is empirically proved that our data satisfy the put-call parity relationship.

3.1 Italian equity and option markets

As already stated, the present dissertation has the objective of applying a machine learning algorithm to the pricing of financial options written on one of the most important Italian stock indices: the FTSE MIB[110]. Therefore, it may be useful to start by briefly presenting such index and its main characteristics.

From an historical point of view, the FTSE MIB index replaced the MIB-30 in September 2004. It used to be run by the Standard & Poor until June 2009, when it was bought by the FTSE Group and its responsibility was consequently placed under the control of the London Stock Exchange Group. The index is the main reference for the Italian equity market since it tracks the stock performance of the 40 largest and most traded Italian stocks[111], which together account for more than 80% of the domestic market capitalization and represent about 90% of the yearly exchanged value.

Given its importance, it should not surprise us knowing that the index is also a sort of benchmark for how the Italian economy develops over time[112].

The Italian equity market shares many similarities with the equity markets of other developed countries; however, it is relatively small[113] and shows in a certain sense a high degree of concentration in specific industries, meaning it lacks some sort of sector diversification. In particular, there is a considerable preponderance of the financial sector, as we can see from Table 6.

[110] FTSE MIB is an acronym for Financial Times Stock Exchange Milano Indice di Borsa.
[111] Keep in mind that they are Italian stocks which however may have their registered office abroad.
[112] Just like CAC 40 is used to measure the performance of the French economy, DAX is a benchmark for Germany, and IBEX 35 reflects the trend of the Spanish economy.
[113] Relying on Borsa Italiana's report about historical statistics of August 2022, the FTSE MIB has a total capitalization of around 520 billion which represents the 80.04% of the total Italian stock exchange value. This means that the exchange is excluded by the top 20 largest markets in the world (considering the capitalization).

ICB Code	ICB Supersector	No. of Cons	Net MCap (EURm)	Wgt %
1010	Technology	1	22,676	6.46
1510	Telecommunications	2	5,464	1.56
2010	Health Care	3	10,251	2.92
3010	Banks	6	62,010	17.67
3020	Financial Services	4	14,734	4.20
3030	Insurance	3	24,708	7.04
4010	Automobiles and Parts	3	52,113	14.85
4020	Consumer Products and Services	1	9,625	2.74
4510	Food Beverage and Tobacco	1	4,876	1.39
5020	Industrial Goods and Services	7	44,886	12.79
6010	Energy	4	46,867	13.36
6510	Utilities	5	52,636	15.00
Totals		**40**	**350,847**	**100.00**

Table 6 - FTSE MIB composition (source: FTSE RUSSELL: FTSE MIB Index (31 August 2022))

As briefly mentioned, the most relevant component of the market is the financial sector; indeed, banks, financial services and insurances account for almost one third of the total capitalization. However, we have to highlight that, following the subprime crisis of 2008, the financial industry has undergone a major downsizing. In particular, the decline was evident for the banking sector, which went from a value of over 93 billion euros in 2009 to about 62 billion euros today, corresponding to a decrease of almost 9 percentage points, from 26,4% to 17,67%. The void left has been filled mainly by an increasing importance of natural resources (utilities and energy), and industrial and automotive companies[114].

With respect to the structure and the regulatory framework of the FTSE MIB, we can say that they are not very different from those of most other industrialized countries and, for those who are interested, they can be found either in the official website of Borsa Italiana or in the relative section of the FTSE Group's material.

At this point we can ask ourselves how is the value of the FTSE MIB index computed. We know that indices are synthetic measures of how the value of a basket of assets changes over time, and we should also know that, according to how the weight of each security is computed, there are three main alternative methods which can be employed:

- Equally Weighted Indices assume a constant weighting factor which is identical for all the securities that make up the index. In such a case, the specific value of each company is not important since all the stocks have the same weight. Therefore, this is a sort of weighted average. Of course, the main drawback of such a method is the "constant assumption", which ignores the different capitalizations of the stocks.
- Price Weighted Indices, instead, change the relative importance associated with each equity according to its particular price. It means that if the price of an asset increases relatively more than the others, its weight within the index increases as well. Thus, the computations

[114] This data comes from various sources, one interested can see for instance the website of Borsa Italiana or various reports (e.g. FTSE Russell: "Per la gestione dell'indice FTSE MIB").

are quite straightforward since they are simply given by the sum of the prices of all the index components. The problem in this case is that the index does not correctly reflect the portfolio performance since the most expensive stocks are the heaviest, regardless of the size of the companies.

- Finally, Value Weighted Indices are nowadays considered the best available choice because they overcome the limitations of the previous approaches. Indeed, in this case, the weight of each stock is proportional to its market value. Therefore, it should not surprise us knowing that most of the main world indices, such as the S&P500, are value weighted indices. The only disadvantage of this method is the fact that it is the most difficult to compute since it needs to be continuously adjusted to account for corporate events (stock splits, extraordinary dividend payments, mergers, and so on).

For its part, the FTSE MIB is a value weighted index, meaning that it is computed in real time keeping in mind the specific capitalization of each constituent stock and its evolution with respect to a particular base period. The total market value of a company is obtained by multiplying the price of the stock by the issued shares. Therefore, the FTSE MIB index value computed at time t, I_t, is derived using the following formula:

$$I_t = \frac{M_t}{D_t}$$

where D_t is the value of the index divisor at time t[115], and M_t, the total market capitalization at time t, is equal to:

$$M_t = \sum p_{It} * q_{It} * IWF_{It}$$[116]

where

- p_{It} is the last traded price, at time t, of the i-th share.
- q_{It} is the number of shares of the i-th stock in the index at time t[117].
- IWF_{It} is the investable weighting factor for the i-th component[118].

This was a short introduction regarding the Italian equity market, but what can we say with respect to the relative option contracts? Since the aim of the thesis is pricing financial options that consider as underlying asset the presented index, we will focus only on describing them. In particular, an option written on the FTSE MIB index is called MIBO, which means MIB Option.

MIBOs were introduced in 1995 and they are traded on the IDEM[119] derivatives market since then. Remember that the FTSE MIB was created in 2004, meaning that, from 1995 up to that point, it was the MIB-30 index to be MIBOs' underlying. These kinds of options have a European style, and they

[115] This is a value given by the specific corporate events that took place during the considered period and it is used to adjust the index value in order to account for this relevant aspect.
[116] From "Ground Rules: FTSE MIB Index" by FTSERussell.com, Vol. 4.5 (August 2022).
[117] This is equal to the number of shares issued for the i-th security net of the treasury shares.
[118] $IWF = 100\% - sum\ of\ the\ \%\ of\ shareholdings\ held\ by\ restricted\ shareholders$.
[119] It is the acronym of Italian Derivatives Market.

are cash settled. What does it mean? Option contracts can be settled in two alternative ways depending on the underlying:

- *Cash settlement*. It means that the position is closed by a transferring of money from the debtor to the creditor. This is usually used when the underlying asset is a derivative or an index.
- *Physical settlement*. In this case, the contract is terminated with the physical delivery of the underlying asset. It is very common for stock options.

Therefore, since MIBOs are settled in cash they involve a monetary transaction to close the contract. This exchange of money is made on the first open Euronext Clearing calendar day following the expiration of the contract, and it is automatic.

To conclude this brief introduction, it is interesting to notice that these contracts are quoted in "*index points*" and not euros. In particular, each index point is worth 2,5 euros and this number is called multiplier. Therefore, for instance, if we want to compute the premium of the option in euros, we need to multiply the quoted value by this amount. With respect to the strike and expiration structures, we can say that there are many possible alternatives simultaneously available, but we will see it better when we will be presenting our dataset in the next sections.

3.2 Structure of the dataset

The analyzed data comes from OptionMetrics, a data provider specialized in financial options. In particular, we will rely on the IvyDB Europe database, which was launched in 2008 and rapidly became the industry standard for historical option prices and implied volatility data in the European financial markets. Indeed, it is nowadays used by a huge number of institutional investors and other market participants thanks to its accuracy, reliability and the amount of valuable information it provides[120].

Let us begin by presenting the structure of the original dataset. It should be immediately noted that this is only the starting point; indeed, once we get the data, we always have to pursue a preliminary analysis which then will help us in performing further data manipulations. In particular, since we are considering machine learning algorithms, we will implement the so-called feature engineering, also known as feature extraction. This is a technique which consists in exploiting the domain of knowledge we have about the problem we are dealing with, in order for us to extract the most from the raw data. It can be done in various ways such as numerical transformations or clustering of some features, and it helps achieving a more parsimonious representation or a better fit of the model.

The data we will consider range from 1st January 2019 to 1st January 2021, and, even if, at first glance, it may seem a small period, the original dataset includes over 400.000 observations relative to 30 different features. The available features, provided by the so-called Option_Price File, are:

[120] Just to mention, the data includes daily option pricing information, dividend projections, historical corporate actions, volatility surface estimates, and so on.

- Security ID: it is the unique identifier given by OptionMetrics to the specific asset;
- Date: it represents the date in which the observation has been collected;
- Option ID: it is a unique integer identifier for a particular option; it can be used to track specific option contracts over time;
- Exchange: the code representing the exchange where the option is traded[121];
- Currency: the code indicating the denomination currency[122];
- Bid: it is the bid price for the analyzed option contract;
- Bid Time: the trade time for the bid price;
- Underlying Bid: this number represents the bid value available for the underlying security when the bid time was taken;
- Ask: it is the ask price for the analyzed option contract;
- Ask Time: the trade time for the ask price;
- Underlying Ask: this number represents the ask value available for the underlying security when the ask time was taken;
- Last: the last traded price for the option contract[123];
- Last Time: the trade time of the last price;
- Underlying Last: the price of the underlying security synchronized with the last time, i.e., available in that specific moment[124];
- Implied Volatility: the calculated implied volatility of the option contract;
- Delta: it is the Delta of the option[125];
- Gamma: it is the Gamma of the option;
- Vega: it is the Vega of the option;
- Theta: it is the Theta of the option;
- Calculation: this character value represents the calculation principle exploited to compute the security price implied in the option pricing model[126];
- Volume: the volume on the exchange where the option is traded on this specific date;
- Open Interest: the total number of contracts traded for this option contract, up to the date before the observation was collected, i.e., this value is lagged by one day;

[121] Note that the number 42 stands for MDD which is "Mercato Dei Derivati" (Milan, Italy).
[122] In this case, 814 represents the euro.
[123] We have to highlight the fact that this value is already the premium expressed in monetary terms, therefore we do not need to adjust it exploiting the multiplier.
[124] It should be noted that this value and the one of the "Last" field are used in the computation of the implied volatility.
[125] Attention: we do not present the concept of Greeks since it goes beyond the scope of the thesis; furthermore, they are not exploited in the presented model. However, they are a fundamental tool which helps option traders in evaluating and managing the risk associated with this kind of contracts. In particular, they are sensitivity measures which represent how the price of an option changes according to a modification of a relevant variable such as the price of the underlying (Delta), the time to maturity (Theta), the volatility (Vega) and so on. Gamma is in a sense the only exception, since it represents the sensitivity of Delta (not the price of the option) to changes in the underlying value.
[126] Indeed, the implied volatility computation relies on the option price by exploiting the following principle. Firstly, it uses the settlement option price when available. The second-best solution is to rely on the last traded price. If even this value is not available, it uses a midpoint computed as an average between bid and ask prices. If this is not possible, it uses the bid price or lastly the ask price.

- Special Settlement: this is a dummy variable where 0 means the option has a standard settlement and 1 means the option has a non-standard settlement[127];
- Reference Exchange: the ID of the exchange where the underlying price is taken for implied volatility calculations;
- AM Settlement: a binary variable which distinguishes whether the options expire at the market closing of the last trading day (0) or at the market opening of the last trading day (1);
- Contract Size: the deliverable quantity of underlying entities. The standardized value is 100 shares;
- Expiry Indicator: this character value indicates if the option is a regular[128] (blank), daily (d), weekly (w) or monthly option (m);
- Strike: the exercise price of the option multiplied by 1000;
- Expiration: the maturity date of the contract;
- Call Put: it is a binary character variable which represents if the derivative is a call (C) or a put (P) option;
- Option Style: it is an additional code expressing the type of the option (weekly, daily, etc.);
- Exercise Style: a variable defining the style of the option. American (A), European €, Bermudan (B), unknown or not classified (?);
- Rate: the dividend yield.

After having gone through this list, we should immediately realize we do not need all this information in order to develop our model, both because some of the items are, in a sense, repeated, meaning they provide the same knowledge of other items, and because we can suppose some of them will not have much predictive power. Therefore, to be sure to include only relevant variables, it may be wise to rely on previous studies, which help us understanding what tends to be truly useful.

As already said, following the publication of the Black and Scholes paper in 1973, the research about financial options has continuously increased over time. Moreover, thanks to the results presented in Chapter 2 regarding the development of artificial neural networks, also the application of this kind of tool for option pricing and hedging has steadily grown. Figure 12 shows exactly this phenomenon.

[127] The option may have a non-standard settlement in terms of number of shares to deliver, size, additional requested guarantees, and so on.

[128] Regular options are options which expire on the third Friday of the specified month.

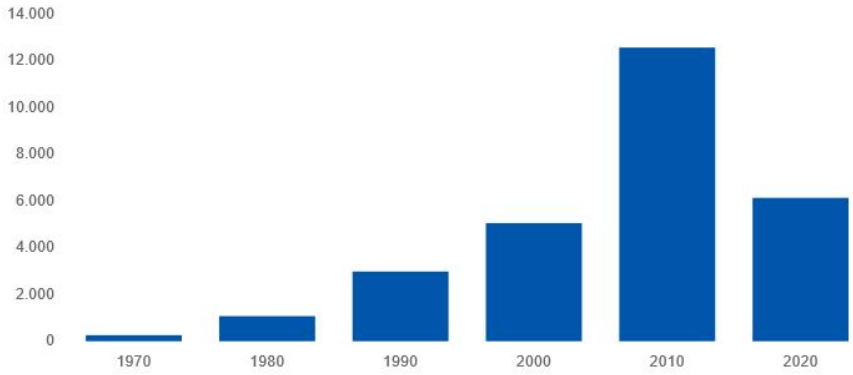

Figure 12 - Number of publications regarding the application of ANNs to option pricing per decade (source: constellate.org)

As we can see, the number of publications grew a lot in the early '90s, consider for instance Malliaris and Salchenberger (1993)[129] or Hutchinson et al. (1994)[130], but this has started exploding only from the second half of the 2000s[131], to the point that nowadays a fully complete literature review is not possible. However, the attempt to provide an extensive literature survey has been made various times. In particular, it is worth mentioning the work of Bennell and Sutcliffe (2004)[132], Chen and Sutcliffe (2012)[133], Hahn (2013)[134] and Ruf and Wang (2020)[135].

By relying on the results presented in these studies, it is possible to draw some preliminary conclusions regarding for instance which features it is better to exploit. First of all, we should remember that the aim of this thesis is to develop an artificial neural network able to correctly estimate the price of a European option[136] with the FTSE MIB index as underlying asset. In particular, we want to show that this machine learning algorithm is capable of providing better results than the Black, Scholes and Merton model even when European options are considered. Moreover, the choice of the underlying is not accidental, but it was determined by the lack of research on this specific subject. Indeed, in the over 150 presented papers, no one ever proposed the application of a neural network to the Italian index.

With respect to the input variables we have to give to the model in order to estimate the option value, the underlying price and the strike price are absolutely indispensable. In particular, there are two alternative approaches according to which the neural network can be fed. On one hand, it is possible to use these two variables separately. On the other hand, we can rely on their ratio, i.e., the moneyness. Several reasons have been proposed proving that it is less efficient to use the two separate inputs instead of exploiting the moneyness. Therefore, it should be clear why "in the

[129] "A neural network model for estimating option prices" M. Malliaris and L. Salchenberger (1993).
[130] "A nonparametric approach to pricing and hedging derivative securities via learning networks" M. Hutchinson, A. W. Lo, and T. Poggio (1994).
[131] We should be careful of the data. Indeed, from the plot it seems that the growth has stopped in the last years. However, this is not true. Note in this regard that, in 2022, with only 2 years passed, we have already reached almost half the publications of the previous decade.
[132] "Black-Scholes versus artificial neural networks in pricing FTSE 100 options" J. Bennell and C. Sutcliffe (2004).
[133] "Pricing and hedging short sterling options using neural networks" F. Chen and C. Sutcliffe (2012).
[134] "Option pricing using artificial neural networks: an Australian perspective" J. T. Hanh (2013).
[135] "Neural networks for option pricing and hedging: a literature review" J. Ruf and W. Wang (2020).
[136] Note that the option price is the most common output when we consider artificial neural networks applied to financial options. Alternative outputs include to estimate the implied volatility or a sort of sensitivity measure such as a hedging ratio.

previous ten years, the second approach is used more often" (Ruf and Wang, 2020). The strongest arguments supporting the use of moneyness are the following:

- it reduces the number of inputs so that it makes the training procedure of the model easier[137];
- it helps reducing the possibility of incurring in overfitting[138];
- moneyness is a stationary variable, in contrast to stock and strike prices. It follows that by using it we can improve the generalization capability of the network[139].

Therefore, we will also rely on the use of moneyness in our model.

From what we already know about option contracts, it should be clear that volatility is another crucial input feature whose use cannot be avoided. We have to say that it can be computed in a lot of alternative ways, among which the most popular ones consist in:

- using historical volatility;
- using implied volatility;
- using volatility indices such as the VIX[140];
- using more complex models, such as GARCH, to estimate the realized or the implied volatility.

It is interesting to notice that there is no general consensus on which is the best measure to use. For instance, Andreou et al. (2008) proved that by relying on the implied volatility the performance of the model improves a lot, when compared with historical volatility-based models. However, Blynski and Faseruk (2006) showed that an ANN overperforms Black and Scholes equation when using historical volatility as input, but it tends to underperform it when using implied volatility instead.

Regarding our practical application, we will use the volatility surface computed by OptionMetrics. This is the implied volatility computed relying on the exploitation of a kernel smoothing technique[141].

Other useful and commonly exploited inputs are the time to maturity, the dividend yield and the risk-free interest rate. All these data are available in the Option_price File except for the riskless rate. However, this feature can be obtained by the so-called Zero_Curve file from the IvyDB Europe database, which provides the current zero-coupon interest rate curves[142]. The file structure includes three columns which are:

[137] See for instance Hutchinson et al. (1994).
[138] See for instance Anders et al. (1998).
[139] Consider as an example the work of Garcia et al. (1998).
[140] VIX is the acronym for Volatility Index and it is a real-time instrument which represents the market's expectation for the short term volatility of the S&P500.
[141] For those interested, the complete description of this procedure is available on the IvyDB Europe Reference Manual (see for instance Version 3.1 rev. 2021).
[142] The standard interest rates used by the IvyDB Europe option model are derived from the BBA LIBOR rates. However, the curve available in the presented data is the one calculated for the euro.

- Date: the date of the zero curve;
- Days: the number of days to maturity;
- Rate: the continuously compounded zero-coupon interest rate.

It should be noted that this yield is computed from "a collection of continuously compounded zero-coupon interest rates at various maturities, collectively referred to as the zero curve[143]". Moreover, we have to say that, for a specific option, the interest rate input we need to use is the one that corresponds to the zero-coupon rate having the same expiration of the contract. This can be obtained with a linear interpolation between the rates of the two closest maturities, which works as follows:

- The Euribor rates for the available maturities, namely 7, 31, 59, 90, 181, and 365[144] days, i.e., they go from 1 week to 12 months, are converted to discount factors using the following equation:
$$DF = \left(1 + r * \frac{d}{360}\right)^{-1}$$
where r is the Euribor rate and d is the actual number of days to maturity.
- The discount factors are transformed into continuous zero rates by means of the next formula:
$$L = -\frac{365}{d} * \ln(DF)$$

In case the considered option has an expiration greater than the longest available maturity, the model uses the interest rate associated with the longest maturity. In the opposite case, meaning when the option has an expiration lower than the lowest available maturity, i.e., 7 days, the data is removed from the model. There are various reasons to exclude such observations, among which the most relevant one is that options characterized by short-time maturities are usually traded at their intrinsic value and therefore they have a small predictive power[145].

To conclude we need to mention that, over time, many papers proposing new alternative input features have been published. In particular, Ghaziri et al. (2000) and Healy et al. (2002) included open interest in their models. Montesdeoca and Niranjan (2016) investigated the predictive power of many explanatory variables such as trading volume, and Cao et al. (2019) analyzed the usefulness of adding the underlying return to the set of input variables. However, in developing our neural network we will rely on the most common features; therefore, we will provide to the model 5 inputs, namely moneyness, volatility, risk-free rate, dividend yield and time to maturity[146].

[143] It comes from the IvyDB Europe Reference Manual.
[144] These are the most common values; however, depending on the specific month they may vary. For instance, 28, 29, and 30 are numbers presented in the dataset.
[145] See for instance "Option pricing via regime switching models and multilayer perceptrons: a comparative approach" by M. Billio, M. Corazza, and M. Gobbo, Rendiconti per gli Studi Economici Quantitativi, Vol. 2002, pp. 39-59 (2002).
[146] Note that we feed the model with only 4 variables since it turned out to be more efficient to consider the risk-free rate and the dividend yield together.

3.3 Data manipulation

So far, we have briefly introduced the dataset. Now, let us focus a bit more on the specific structure of the observations and on the preliminary operations we have to perform in order to clean the raw data. We said we have more than 400.000 observations, to be more specific they are 424.138. In particular, for every trading day about 390 observations are collected, each of which can be distinguished from the others according to its maturity date and its specific strike price[147].

Of all these contracts, 210.138 are call options and 214.000 are put options.

Already from this circumstance a problem arises: indeed, we have to build two distinct models if we want to be able to correctly price calls and puts. Of course, this is not efficient, but, if we cannot find an alternative solution, it is the only way we can follow. Fortunately, since we are trying to price European-style options, we can rely on the useful relationship defined by the put-call parity. Still, we should remember this is just a theoretical result, i.e., there are no guarantees the equation will hold in our data. Therefore, the only thing left for us to do it is to test if the relationship is verified in our dataset[148].

70	670	172	422	672	257	758	1257	264	764	1264	1765	83	184	283	465	35	785	1785
68	668	166	416	666	161	661	1161	157	657	1157	1657	82	182	282	469	31	781	1781

Table 7 - Put - Call parity relationship

Table 7 is an approximation[149] of a sample of results extracted from our dataset and showing that the put-call parity holds. Indeed, the first row has been computed as:

$$S_0 - \frac{X}{(1+r)^t}$$

where S_0 is the stock price at time 0, X is the strike price, r is the interest rate, and t is the time to maturity. The second row is instead computed as:

$$c - p$$

where c is the call option price, and p is the cost of the corresponding put option. In other words, we are exactly calculating the two hand-sides of the put call parity. Then, it is possible to perform a comparative analysis which aims in proving if the two terms are identical. It should be noted that, since we are considering real financial data, it would be virtually impossible to obtain a perfect equality. Therefore, we will be satisfied if the relationship is approximately verified.

[147] As an example, there are options traded in 2019 which will expire in 2023. Of course, the closer the maturity the more strike prices are quoted. For instance, for option contracts traded on the 02/01/2019 and expiring on Friday 04/01/2019 there are about 30 strikes available whereas less than 20 exercise prices are listed for options quoted the same day but expiring on 15/12/2023.

[148] The code is available in Appendix A.

[149] Indeed, as we see, decimal digits are not expressed.

As we notice, ignoring what can be considered a sampling error, the results are extremely similar to each other and, since this is true for all the analyzed data, we can assume that the put-call parity holds within our dataset. Moreover, it is possible to compute the mean absolute deviation and the mean relative deviation which are respectively 30.54 and 0.076. This means that on average the difference between the two values will be equal to 7.6%. Considering the large size of the sample, this is an extremely good result, also because it can be observed that it is mainly determined by the most expensive contracts, i.e., ITM options, which we know being the most difficult to correctly price.

Thanks to this result, we can now proceed by developing only one neural network. In particular, we will build a model capable of providing accurate estimates for the value of European call options. Why call options? First of all, since we are relying on the put-call parity equation, it does not make much difference the type of option we choose to price. Secondly, call option pricing models are the most widespread in the existing literature[150].

At this point, we are left with about half of the data. However, these are raw data that include outliers, missing values, and in general "weird" observations with no predictive powers. It means that we have to perform some sort of feature engineering to deal with these kinds of problems. Indeed, to be able to develop an optimal structure for our neural network, it is essential to select a proper subset of the original data which is able to capture the model generating the data, and to detect relevant existing input-output patterns. Moreover, if we succeed in such a task, we can exclude from the learning procedure uninformative observations, therefore boosting the accuracy of the model and reducing the required computational costs.

What is feature engineering? It is a data manipulation technique which involves different approaches such as data deletion, combination or mutation with the aim of improving the performance of our machine learning model. Obviously, to be able to perform an effective manipulation, we need to exploit our previous knowledge about the nature of the problem and the structure of the dataset. Therefore, we can start by excluding those option contracts which have not been traded, since we can suppose they will be characterized by an extremely small predictive power in determining the price of the option. It is interesting to notice that this operation alone reduces the number of available data to less than 50.000, meaning that more than three quarters of our initial dataset has been cleared at once. Following the same logic, we can remove also the options that have a volume equal to one. In fact, they will be "case-specific", i.e., these are single operations which can take place at prices far from those of the market that are not representative of the general process generating the data.

Relying on the previous literature, we can then delete the observations with a short time to maturity. In particular, we will remove all the data characterized by a residual life of less than 7 days. This because these kinds of contracts are usually traded at their intrinsic value, therefore adding little information about the true relationship linking inputs and outputs.

It should be noted that already with these two operations, we shrank down our dataset size to little more than 40.000 input-output pairs. However, we still have to deal with missing values and

[150] See for instance "Neural networks for option pricing and hedging: a literature review" by Ruf and Wang (2020).

outliers. Indeed, the management of such observations is a crucial aspect we necessarily need to deal with in order to not compromise the quality of the estimates of our model[151].

The problem in this case lies in the definition of one of these two terms. If, on one hand, a missing value can be easily defined as a missing observation, meaning a situation in which one or more pieces of information about a variable have not been stored; on the other hand, the definition of outliers is neither unique nor clear. The statistician Hawkins (1980)[152] defined an outlier as "An observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism". Therefore, these are observations that lie at an "abnormal distance" from the other values collected in our sample. However, there is still a huge degree of vagueness regarding what can be considered an abnormal distance.

Of course, since these two concepts have different drawbacks, also the way in which we deal with them changes. In particular, the hard task with missing observations is to find them and to understand how they are represented within the dataset. In our case, the OptionMetrics database has different notations for missing values; for instance, when the information about the volatility of one piece of data has not been recorded, we do not find a blank space or a character string such as NA, but a numerical value instead, namely -99.99. Obviously, not being aware of these kinds of circumstances can lead to unsatisfactory results because we can risk including in the model meaningless values like for example a negative volatility.

The problem with outliers is different. As previously stated, these can be seen as data generated by a different process other than the one we are trying to analyze. Therefore, they can be misleading. However, they are in a sense difficult to be recognized and, in this regard, there exist several techniques we can employ in order to try to detect these strange observations. The most famous approach is the statistical one based on the so-called Z-score. This simply represents how far from the sample mean a certain data point is, in terms of standard deviations. Despite its simplicity, this method has some serious limitations. First of all, we need to know the distribution of the dataset. Secondly, this distribution has to be well-shaped, meaning it needs to be, at least within a certain degree, symmetric.

The most common assumption is to rely on the normal distribution[153]. In such a case, we usually consider as outliers all the observations that lie at a distance greater than 1.96 standard deviations, in absolute value, from the mean[154]. Therefore:

$$If\ Z = \frac{x - \mu}{\sigma} \geq |1.96|\ we\ can\ suppose\ the\ x^{th} observation\ to\ be\ an\ outlier$$

where μ is the sample mean and σ is the sample standard deviation of our data.

[151] The process of identifying valid, useful patterns in data is known with various different names. For instance, it is called "knowledge discovery in databases" in the paper "Optics-of: identifying local outliers" by M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander, PKDD, LNAI, Vol. 1704, pp. 262-270 (1999).

[152] "Identification of outliers" (vol. 11) D. M. Hawkins (1980).

[153] However, we can rely on different distributions such as the student-t.

[154] Note that this value is derived by the specific structure of the standard normal distribution. Indeed, observations that deviate from the mean more than 1.96 standard deviations have a maximum probability of 5% of being observed.

Since we are considering a multidimensional, nonlinear problem it is hard to think that our features will comply with the normality assumption[155], i.e., even in this case it seems better to exploit our knowledge about the problem and to rely on previous studies. In particular, a lot of different papers have clearly shown the improvement in the performance of those neural networks that exclude deep-out-of-the-money and deep-in-the-money options from their learning procedure[156]. In our case, the selection was made according to both the moneyness and the premium of the options; as an example, among the 43.510 input-output pairs obtained by removing the zero-volume contracts, only 41 observations where so deep ITM that they showed a premium above 8.000 euros. This means that by removing these data we were deleting only the 0.094% of the total information.

Now, after having managed the outliers and the missing values, all that is left to do is to understand if it may be helpful to perform some sort of normalization or other kinds of transformations on our input features and output variable.

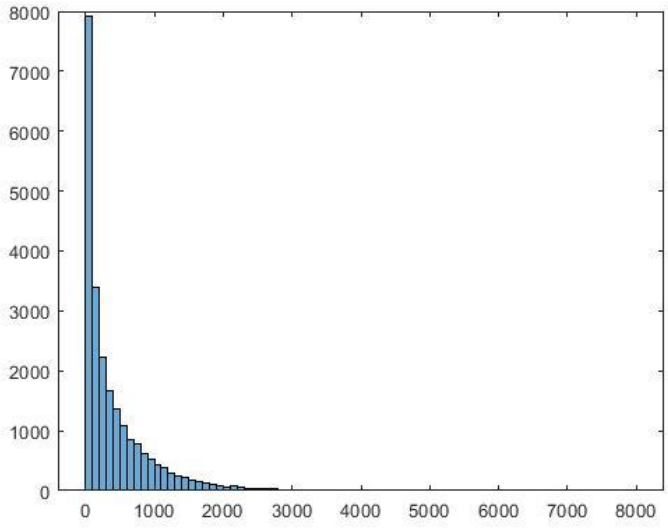

Figure 13 - Option price distribution

[155] In particular, we tested such hypothesis and proved that our features violate the normality assumption.

[156] See for instance "Improving the pricing of options: a neural network approach" by U. Anders, O. Korn, and C. Schmitt, Journal of Forecasting, Vol. 17, pp. 369-388 (1998).

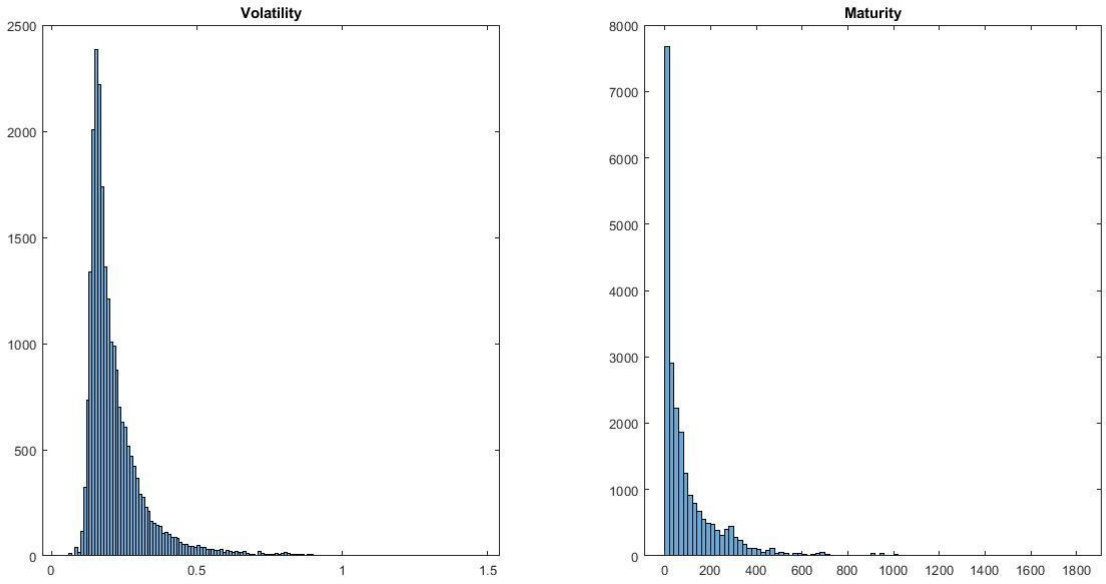

Figure 14 - Volatility and Maturity distributions

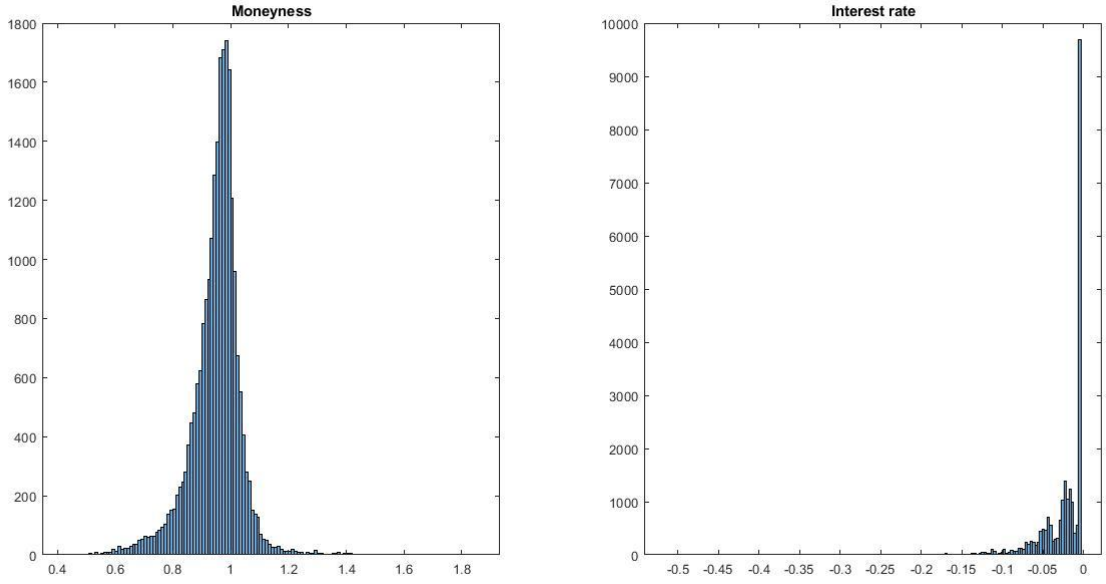

Figure 15 - Moneyness and Interest rate distributions

As we can see from Figures 13, 14, and 15, which are showing how our original data are distributed, some interesting facts can be highlighted. In particular, we notice that, except for the moneyness which, as we expected[157], follows a sort of symmetric distribution, they are all asymmetric variables. Moreover, we see that they are characterized by different scales, meaning the ranges of values of their domains are not the same. These are both problems, since they may affect the efficiency of

[157] Remember we said that one of the main reasons according to which we choose to rely on moneyness, instead of the separate stock price and strike price, it is the fact that this variable is stationary and tends to show a nice behaviour.

our neural network. Therefore, in order for us to address these drawbacks, we have to apply a normalization technique and some type of useful transformation.

Before doing that, let us start by drawing some initial observations. First of all, it should be noted that the vast majority of the data is made of at-the-money options. Indeed, Figure 15 shows that around 90% of the traded contracts have a moneyness between 0.8 and 1.2. Moreover, as we can observe, a huge part of these is extremely close to one[158]. Finally, we have already pointed out that moneyness follows a symmetric distribution; however, it is not perfectly true, as we can see that the left tail is a little bit larger, i.e., slightly out-of-the-money options tend to be a bit more traded than slightly in-the-money contracts, at least in our sample. Consequently, it should not surprise us to notice that most of the premiums are relatively small, having a monetary value lower than 1.000 euros. Indeed, since most of the traded contracts are ATM or OTM options, we can suppose they are not too expensive. This is why we observe that kind of right-skewed distribution in the cost of the options.

Then, regarding the volatility and the maturity plots, it is possible to say they both show an extremely positive asymmetry. This means we usually expect to find options characterized by a short time to maturity, in general lower than 100 days, and which embed a relatively small implied volatility. Lastly, the interest rate is computed to keep into account both the risk-free rate and the dividend yield. Since we are in a period of negative interests and the paid dividends have been relatively small, it should be clear why this feature displays a negative asymmetry.

In order to prevent overfitting and to make the learning procedure easier and consequently faster, it may be beneficial to normalize the variables between 0 and 1 so that they all have the same magnitude, i.e., their contribution is not determined by the different scale characterizing them but, instead, by the different weights the network assigns to each of them[159].

At the same time, it could be useful to transform some of these features so that they will show a more symmetric behaviour[160]. The operations performed on the explanatory and dependent variables are made available in Appendix B.

The normalization is performed by applying the following formula to each observation:

$$x_{new} = \frac{x - min_x}{max_x - min_x}$$

where x_{new} is the new normalized value, x is the original value of the analyzed variable, min_x is the minimum of x, and max_x is the maximum of x. By doing so, we make sure that each feature starts from 0 and has 1 as maximum.

The results are presented in Figure 16, Figure 17 and Figure 18:

[158] Which means perfectly at-the-money options.
[159] Note that in case of machine learning this is also called “feature scaling”.
[160] Note that these operations do not modify the structure of the data or their influence on the determination of the output, but they simply make the computation easier for the model. Moreover, once the network has been derived and the results have been estimated, they are converted back to their original form.

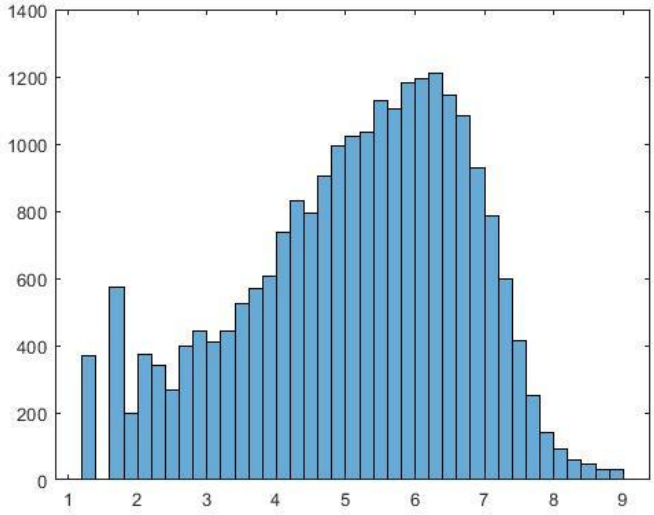

Figure 16 - New option price distribution

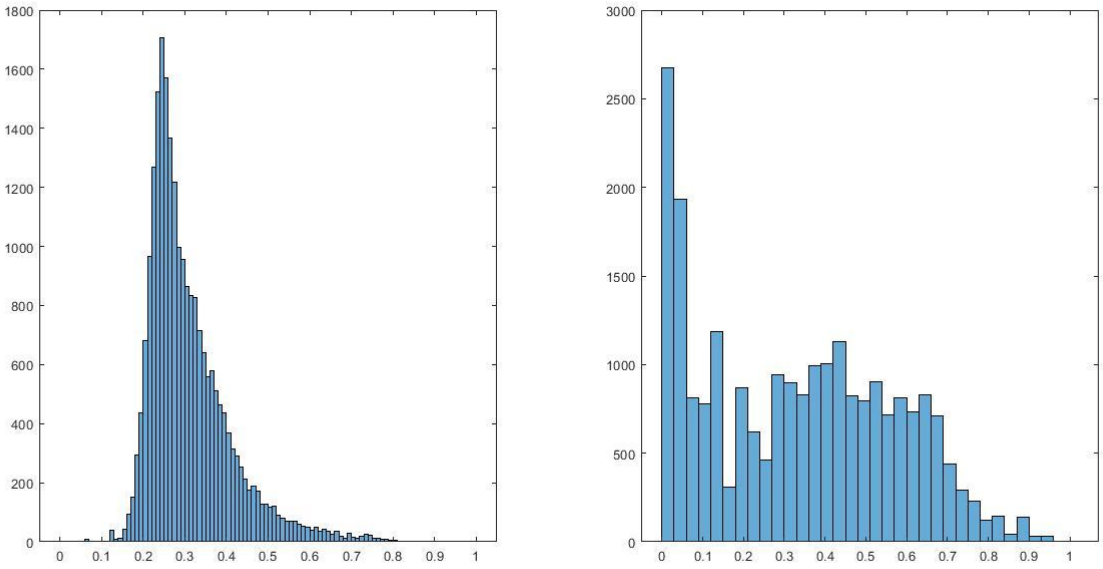

Figure 17 - New Volatility and Maturity distributions

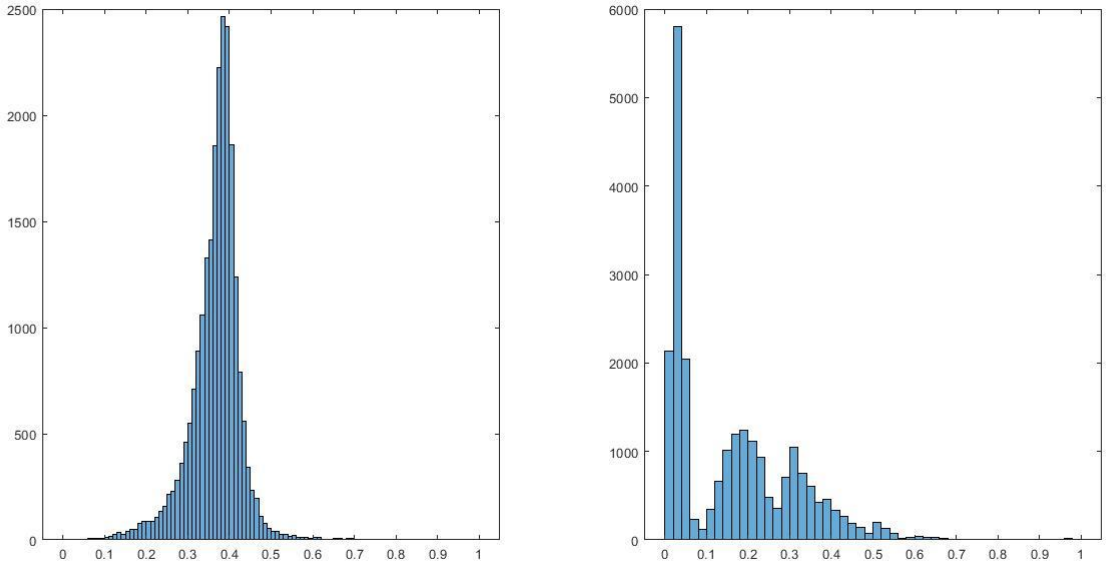

Figure 18 - New Moneyness and Interest rate distributions

As we can see, these new distributions all seem more symmetric. Of course, we have not been able to make them look perfect, but we have been still capable of improving the initial situation. Moreover, they are now all between 0 and 1, meaning they all have the same magnitude on the network and the only variable that matters in determining their impact on the output is the weight the model assigns to each of them.

This concludes the presentation of the FTSE MIB, the description of the exploited dataset and the operations performed on the explanatory variables and output. The next Chapter will present the development of a multilayer perceptron artificial neural network which aims at providing a precise and reliable estimate for the price of a European call option.

Chapter 4 An Artificial Neural Network for pricing MIBOs

In this final Chapter the performed experiment, which consists in developing a multilayer perceptron ANN for forecasting the future price of options contracts, is described in detail. In section 4.1 it is explained why we need to rely on such a kind of algorithm by showing the complexity and the high nonlinearity of our input feature space. Paragraph 4.2 is then focused on presenting the hyperparameter optimization techniques on which we chose to rely in order to tune our model. The third part of the Chapter shows instead the steps we followed to select the best architectural structure of the network, providing, at the same time, the empirical evidence that supports our choices. Finally, in section 4.4 the obtained results are analyzed. In particular, the last paragraph is organized as a comparison between the results of the developed artificial neural network and the ones of the competing Black-Scholes-Merton model.

4.1 Why an artificial neural network

At this point, we are well aware of the scope of the thesis, i.e., to develop a MLP ANN to estimate the price of European call options by exploiting the information provided by a set of input variables. In particular, in the previous Chapter we described the structure of the data and the main characteristics of our features which are implied volatility, moneyness, time to maturity and interest rate[161].

Remember we have been able to reduce the number of the inputs so much by starting from the existing literature and the previous theoretical and empirical results. Moreover, this has been done with the twofold objective of simplifying the learning procedure[162] on one hand, and improving the network performance on the other hand. However, we can ask ourselves if it is truly necessary to rely on a complex structure such as the one of a neural network. It means, can we not find a simpler way to deal with this kind of problem? For instance, why do we not exploit a multiple linear regression model? There are several reasons that support the choice of the artificial neural network, let us go into it.

First of all, since our problem has a high dimensionality, it is not possible to show the effect of all the features on the independent variable at once; in any case, we can still observe the partial effect of each explanatory variable, meaning we can plot how each individual input affects the output when the other variables are left free to change:

[161] Recall this is an artificial variable built to keep into account both the dividend yield and the risk-free rate.
[162] It should be pointed out that by reducing the computational cost the learning of the model accelerates.

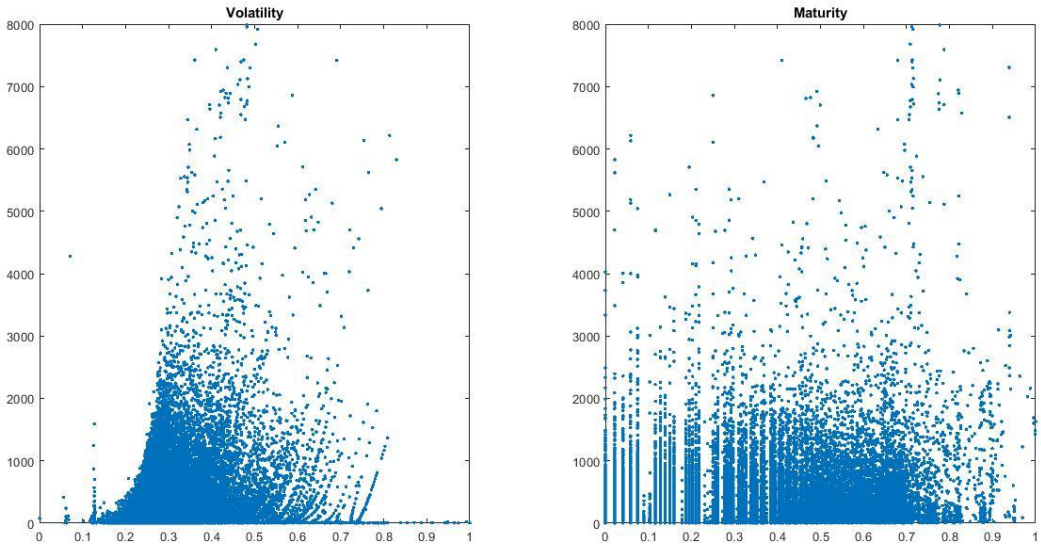

Figure 19 - Volatility and Maturity effects on call price

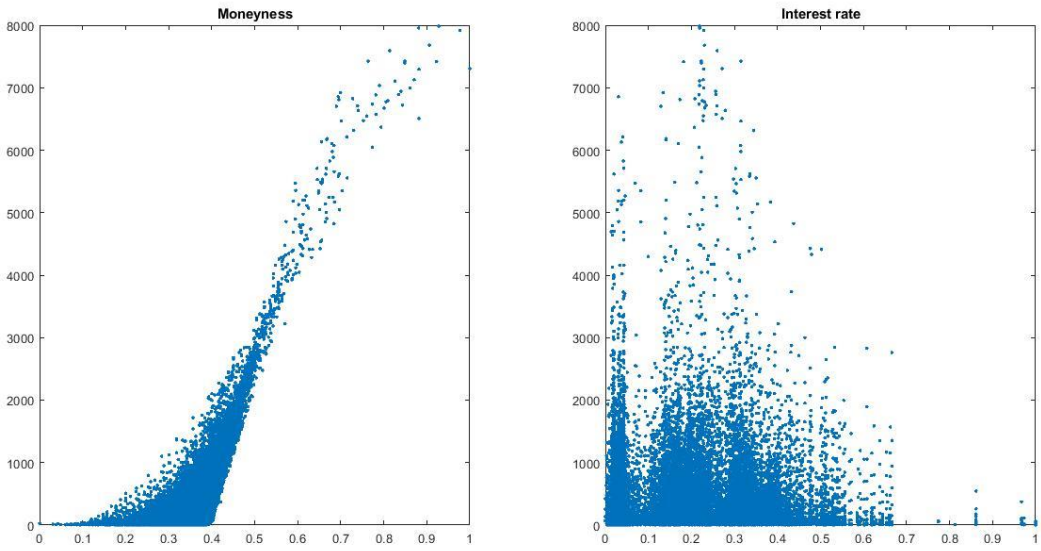

Figure 20 - Moneyness and Interest rate effects on a call price

As we can see[163], these are all pretty noisy Figures which do not show any clear, detectable relationship between the input features and the output; in particular, no linear association can be spotted. However, this is not completely true, since there seems to be one relevant exception: moneyness.

[163] Note these are the existing relationships between the transformed input features and the original output variable. We have to highlight that these results do not change significantly according to the kind of variables, original or transformed, we are considering. Therefore, we choose to plot these values to show how they impact on the original price of the option expressed in monetary terms.

Indeed, in case of moneyness, we see some kind of positive correlation between the dependent and the independent variables, which can be explained relying on the theory. In fact, due to the effect of the normalization technique we applied to the data, the value 0.5 represents now "perfectly at-the-money options", i.e., contracts with a strike price identical to the underlying price. It follows that all the points that lie on the left of such threshold are, in different ways and with different degrees, out-of-the-money options, whereas the observations on the right are in-the-money derivatives. Therefore, knowing that moneyness is determined by the intrinsic value of a contract, it is quite normal to detect an increasing trend, meaning options with larger intrinsic values are generally more expensive.

What can really surprise us is to notice that this positive behaviour is not present in other features. For instance, we already know that an increasing volatility is usually followed by an increase in the cost of the options. However, we do not spot such a clear relationship in our data. Does it mean that the theory is wrong? No, more likely it means that the effects produced by a rise in the volatility are not straightforward to explain, and they tend to affect the price of the contract in more complex ways, to the point that we can suppose there are also forms of interaction between the explanatory variables[164].

This is the second reason why we should prefer to rely on the structure of the neural network instead of closed-form formulas or other regression techniques such as multivariate linear regressions. Let us explain why neural networks tend to overperform such models with an example. In case of a multivariate linear regressions, we have to add cross products, high order terms, etc. in order to be able to account for further interactions between the input variables. The problem is that we are not sure if the new added terms are statistically significant or if they will not improve the performance of the model, therefore we have to test, in a sense manually, the goodness of this hypothesis[165].

With a neural network model, instead, the algorithm will be able to create its own features directly[166]. In other words, it is the system itself to detect the best nonlinear ways to map the inputs to the output. By recalling the universal approximator theorem of Cybenko (1988), we are also sure that, by exploiting a deep learning structure[167], a MLP ANN can represent any continuous function with a given arbitrary precision. Moreover, thanks to its particular architecture, an artificial neural network is proved to be efficient in performing this task, therefore it is a good choice to address these kinds of problems by relying on such an algorithm.

The problem with artificial neural networks is that the above-mentioned theorem says nothing about the computational cost that it is required to perform such a task. In particular, what happens to the number of observations that we need in order to estimate the parameters of the model when the complexity of the network increases? With this regard, it is well known that machine learning

[164] Meaning the effect of one explanatory variable on the output changes if one or more independent variables are modified.

[165] The drawback is that to test a null hypothesis we need to assume a proper distribution of the data, a significant threshold, and so on. Assumptions which make the model more unrealistic and far from reality.

[166] Note these are both weights and biases. However, differently from a linear regression or other kinds of models, in this case, we will not be able to observe such values. Indeed, neural networks are, in a sense, sort of "black-boxes".

[167] Meaning by relying on a neural network with at least one hidden layer.

systems are affected by the so-called "*curse of dimensionality*" (Bellman, 1957)[168]. The expression, coined in the dynamic programming context, refers to a phenomenon that naturally arises in many high dimensional problems.

In our case, this simply means that the amount of required data grows exponentially when the size of the network increases. To frame the problem in a different way, we can say that given a fixed number of training examples, the predictive power of a machine learning algorithm first tends to increase as the complexity of the model grows, i.e., the system is more capable of detecting relevant patterns in the dataset, but after a certain point, the performance starts to deteriorate; in a sense, it is as if there is not enough information to correctly train the whole model (Hughes, 1968)[169]. Figure 21 shows exactly this relationship.

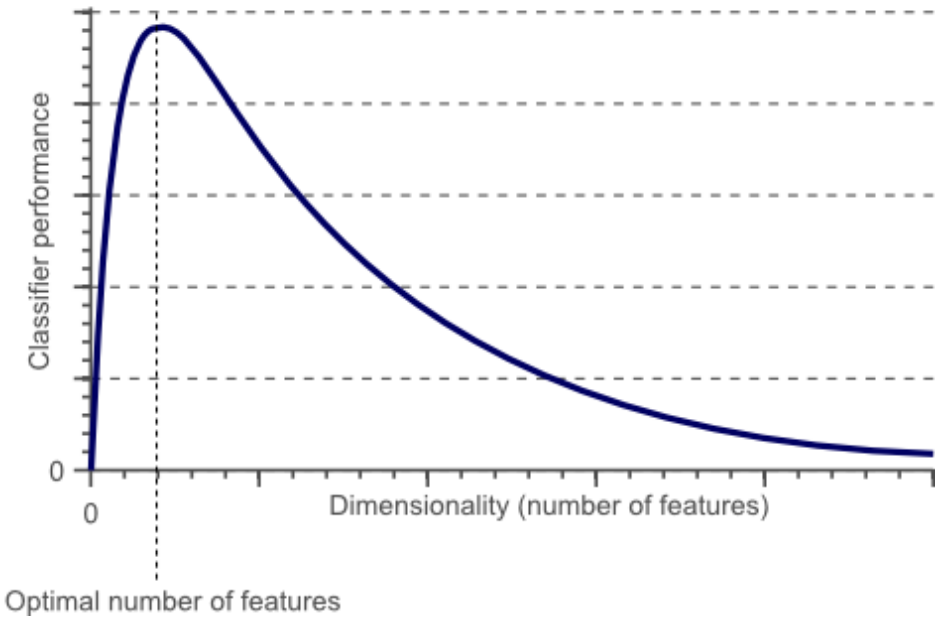

Figure 21 - Model performance versus model complexity (source: International Journal of Engineering Trends and Technology)[170]

Therefore, it should be clear that one of the most concerning aspects in building our neural network must necessarily regard the choice of a proper structure for the model, meaning the tuning of the hyperparameters, also known as hyperparameter optimization. In the next sections we will address exactly this problem, by showing the procedures followed to define the most performing architecture of our multilayer perceptron.

[168] "Dynamic programming" R. Bellman (1957).
[169] "On the mean accuracy of statistical pattern recognizers" by G. Hughes (1968).
[170] From "A comprehensive review of subspace clustering in the analysis of big data" by T. Gayathri and L. Bhaskari, International Journal of Engineering Trends and Technology, Vol. 39, pp. 135-142 (2016).

4.2 Tuning the hyperparameters

Considering what has been said so far and given the fact that the number of available data points in our application is fixed, i.e., the dataset size does not change over time, we are sure that an optimal number of parameters to use in the model exists and can be found. Therefore, we have to understand how complex the network needs to be to provide the best possible estimates. To tackle such a problem various approaches have been developed; we will see them in a moment. Before doing that, we have to recall that a neural network learns by constantly adjusting its parameters and, in particular, the operation is performed by exploiting a loss function. Indeed, this metric measures how similar the true and the predicted outputs are, and of course, the goal of the model is to minimize it.

This means that the choice of the loss function is a crucial aspect to consider since it may deeply affect the results of the model, see for instance Girosi, Jones, and Poggio (1993)[171]. With this regard, the most popular loss function, that is nowadays the default in most regression applications, is the Mean Squared Error (MSE)[172]. It is defined as follows:

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

where n is the number of observations, and $y^{(i)}$ and $\hat{y}^{(i)}$ are respectively the real value and the one predicted by the model for the i-th input-output pair.

The MSE is the preferred loss function, meaning it is the first to be used and it has to be changed if and only if we can detect strong indications that advise against its exploitation. However, we can highlight that, in general, there are plenty of reasons that support it.

From a mathematical point of view, we see that it is computed as the average of the squared differences between the actual and the predicted values, i.e., it is a quadratic function[173]. It follows that the result will always be positive, regardless of whether the predicted output is greater or lower than the target value. Moreover, by squaring the differences we are penalizing the model for making larger mistakes[174]. Finally, the MSE is also a convex function, i.e., it has a clearly defined global minimum, and it is differentiable, unlike other loss functions.

Unfortunately, it has also some disadvantages. Indeed, it tends to be quite sensible to the presence of outliers and it is scale dependent. However, in the previous Chapter we performed a deep data manipulation and, by managing the abnormal observations presented in our dataset and by normalizing the explanatory variables, we were able to reduce the magnitude of these drawbacks. Therefore, from these results, it follows that we can rely on the backpropagation algorithm and the gradient descent in the implementation of our artificial neural network[175].

[171] "Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines".
[172] Note that the MSE is defined as the L2 loss function, also known as Least Square Error (LSE), divided by the number of available observations.
[173] It ranges from 0 to plus infinity, in particular, a good model will have a low MSE.
[174] It means that the penalty is not proportional to the error but to its square instead.
[175] See for instance Janocha K, and Czarnecki, W. M. (2017) or Zahra M. M., Ali M. H. E., and Refaee, A. (2014).

We have now explained the choice regarding the loss function, but there are still other important questions that remain unsolved. In particular, we need to define the best architectural structure of the neural network and, in order to do this, we have to identify a proper value for each of the following variables:

- The number of hidden layers of the model;
- The number of hidden neurons per each hidden layer;
- The activation function we want to use;
- The partition of the data between training, validation, and test sets;
- The performance measures to exploit in the evaluation of the goodness of the network.

Let us start by addressing the "simplest" task: the number of hidden layers.

First of all, we have to highlight that, despite some decades have already passed since the first neural networks were applied to the option pricing task, there still does not exist an exact solution to the problem of what is the right number of hidden layers[176]. In particular, the existing literature has mainly exploited a trial-and-error approach. However, in more recent years, there have also been studies presenting a more formal method to deal with this matter, like for instance the paper by Stathakis (2009)[177].

In any case, we need to mention that the use of multiple hidden layers may cause different problems. Indeed, we know that very complex structures are more prone to overfitting, and the complexity of the model grows exponentially with the increase of the number of the hidden layers[178]. Moreover, deep neural networks are in general harder to train both in terms of computational resources, such as time, and dataset size[179]. Therefore, unnecessary increments of the hidden layers may have a deleterious effect on the robustness of the model, on its overall efficiency and on its capability to generalize. Last but not least, we are already aware that neural networks with one or two hidden layers are sufficiently structured to solve any nonlinear, high dimensional problem[180], meaning there is no need to introduce additional complications. These are the main reasons why almost all the analyzed papers rely on a single hidden layer structure; in this regard, consider as an example the work by Ruf and Wang (2020).

Following the previous results, we decided to implement a one hidden layer model. In particular, having four input variables and one output, the general structure of the network will be similar to the one presented in Figure 22:

[176] See for instance "The application of artificial neural networks to the analysis of remotely sensed data" by Mas, J. F., and Flores, J. J. (2008).
[177] In his work the author, Stathakis, D. (2009), proposed the application of a genetic algorithm to the specific task of optimizing the choice of the number of the hidden nodes and the hidden layers. The method was even compared with other techniques and provided encouraging results.
[178] "Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture" by Karsoliya S. (2012).
[179] They usually require millions of data to be able to perform a proper training.
[180] Cybenko's theorem of 1988.

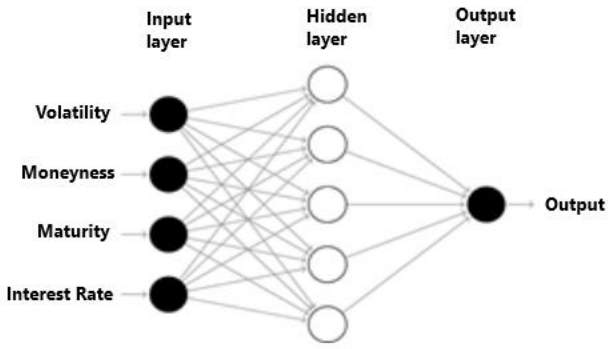

Figure 22 - Artificial Neural Network structure

Once the problem of how many hidden layers to use has been addressed, the next step is to understand the optimal number of neurons and the best split. Even in this case a general answer does not exist yet, and we have to say that it "is unlikely to be discovered any time soon" (Hutchinson et al., 1994). However, it can be pointed out that, in general, the standard approach to the problem relies on cross-validation methods and their variations (Wahba, 1990)[181]. Therefore, we are left with a question: what is cross-validation?

Cross-validation is a technique used to ensure that the learning process has been performed in a proper way; in other words, it is exploited to improve the accuracy of the model. It is possible to define various alternative implementations of such a method, but the plain vanilla version simply consists in dividing the original dataset into complementary subsets, i.e., they have zero elements in common, and then performing a series of operations. The usual setting involves the partitioning of the data into three groups:

- The training set;
- The validation set;
- The testing set[182].

The training set is used to teach the model about the structure of the data and the existing relationship between the inputs and the output. Then the validation set is exploited to provide an unbiased evaluation of the model, i.e., the information it sends us is used to validate the analyses previously made. It means this is the portion of the dataset we rely on to tune the hyperparameters. Finally, the testing set is a separate group of data which is used to assess the ability of the model to deal with new, unseen observations once the training has been completed. Therefore, in order to prevent overfitting and to be sure that the model is reliable and has a good generalization ability, we must divide the available data into two groups: one known, and the other unknown[183].

[181] "Spline models for observational data" by Wahba, G. (1990).
[182] See Chapter 2 for a more detailed explanation.
[183] Then the unknown part will be usually divided into the validation and the testing sets.

Indeed, since the optimization process aims at maximizing how the model fits the training set, namely the known part, if we do not introduce a sort of "independent control", performed through the unseen observations, the network has a strong incentive to overfitting.

Now, it should be clear why we divide the dataset into multiple groups. However, we said nothing about how this partition should be made. In particular, since the size of the three sets is a crucial factor in determining the quality of the estimates of the model, we need to be able to detect its best "composition". The problem is that there is not a thing such as an optimal split percentage, and we also have to account for the bias-variance trade-off. Indeed, on one hand, when the training set is relatively small, the model tends to show high variance. On the other hand, if the validation/test sets are not big enough, the performances of the network display a greater variability.

Over time many empirical results have been proposed and alternative rules of thumb have been developed. For instance, when there are many hyperparameters to tune, it is a good practice to rely on a larger validation set to optimize in a proper way the network. With this regard, it is interesting to notice that usually the differences between the performance estimated by the validation set and the one provided by the test sets tended to decrease "*when more samples were available for training/validation, and this is because the models were then moving towards approximations of the central limit theory for the simulated datasets used*.". Moreover, "*having too many or too few samples in the training set had a negative effect on the estimated model performance, suggesting that it is necessary to have a good balance between the sizes of training set and validation set to have a reliable estimation of model performance*." (Xu and Goodacre, 2018)[184].

Keeping in mind all what has been said, in our model, we decided to rely once again on a trial-and-error approach by testing for different split settings, each of which is characterized by an increasing portion of training data: $50/25/25, 60/20/20, 70/15/15$, and $80/10/10$[185].

At this point, we have understood how to address the data attribution between the different subsets. However, we still do not know what method we should use in tuning the hyperparameters, meaning what is the best technique to find the optimal hyperparameters? There are several possible approaches, which can be classified in the following three popular families:

- Grid search;
- Random search;
- Other more complex models.

The grid search is the most common approach since it is a theoretically simple and straightforward procedure. It can be defined as an exhaustive, brute-force estimator, meaning all the possible combinations of hyperparameters in the search space will be analyzed to determine the combination that yields to the best performance. Of course, the main advantage of this technique consists in its pervasiveness. Indeed, there is not a better way to find the optimal solution than

[184] "On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning" By Xu Y., and Goodacre R., Journal of Analysis and Testing, Vol. 2, pp. 249-262 (2018).

[185] The first number represents the percentage of the data used in the training procedure while the second value is the portion of the dataset implied in the validation step. Finally, the third number is the percentage that pertains to the test set.

trying all the possibilities. However, arriving at this perfect knowledge is quite expensive. Therefore, the biggest drawback of the grid search is the fact that it does not scale well and tends to be quite inefficient when many hyperparameters are considered. In particular, by increasing the hyperparameter search space, the computational cost, in terms of time and required resources, exponentially rises.

Random search instead does not analyze all the possible hyperparameters in the search space, but it evaluates only a specific number of them, given by the user, at random. The main advantage of this approach is that it usually performs less computations than the grid search, i.e., it is more efficient from a computational point of view. However, the risk is that, since the search is made at random, it may fail in capturing relevant information present in the data, leading to poor performances. Figure 23 shows graphically the difference between these two approaches:

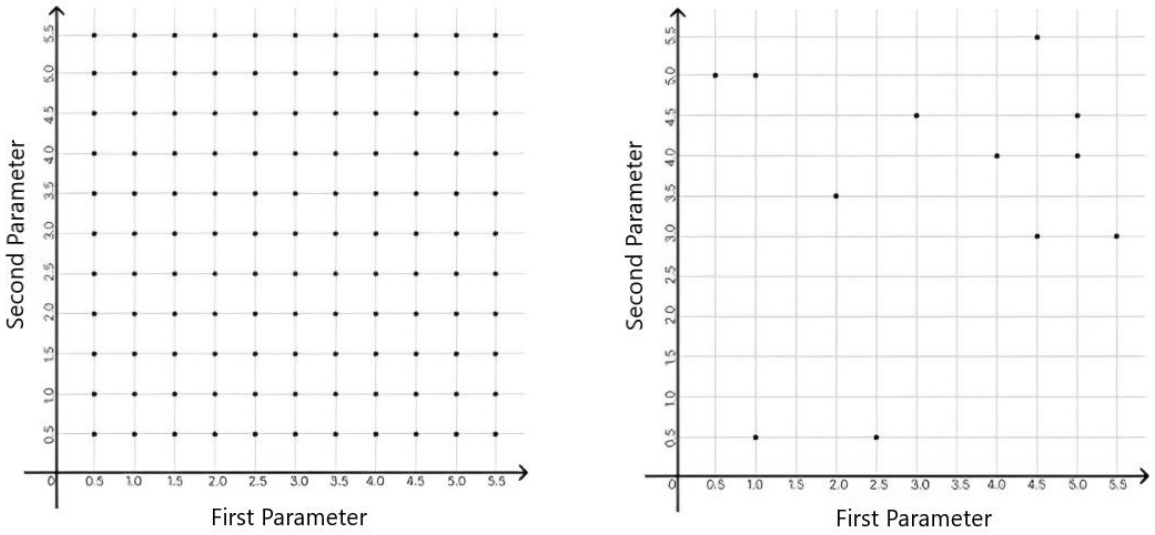

Figure 23 - Grid search vs Random search (source: github.com)

Finally, there are other complex optimization methods that can be employed in tuning the hyperparameters, like for instance Bayesian optimization or Genetic algorithm techniques. These methods are becoming more and more popular over time since they are able to provide near optimal results, extremely close to the ones obtained by a complete grid search, in a time efficient way. However, since our network considers few hyperparameters, we can rely on the grid search so that we are sure to detect the best available solution[186].

Going on with the concern of how the parameters of the model have to be estimated, we can say that a huge part of the existing literature shows that the efficiency and the accuracy of the estimation process depends also on whether the parameters are all estimated simultaneously or sequentially; in particular, we can distinguish between batch learning algorithms and online learning algorithms.

[186] See for instance "Grid search, random search, genetic algorithm: a big comparison for NAS" by Liashchynskyi P., and Liashchynskyi P. (2019).

They are different one from the other in the sense that the former takes "batches" of data to train the model all at once, while the latter instead starts from an initial guess and then by picking up the observations one by one, it recalibrates each parameter every time. Since a rigorous comparison of estimation methods is not the primary goal of this thesis, we chose to exploit the results provided by previous studies. Therefore, in our application we rely on the so-called Levenberg-Marquardt backpropagation algorithm, which has been proved to be the most efficient training function; see for instance Hutchinson (1993). It is a powerful offline batch training method which updates the value of all the weights after every simulation. On this regard, it is possible to say that it is not completely clear why, but online methods tend to overperform batch methods when they are used with neural networks to deal with large scale, highly nonlinear problems[187]. However, the Levenberg-Marquardt algorithm has been proved to provide extremely satisfactory results. In particular, it is able to yield to faster convergence when compared to other training approaches.

So far, we have made clear how to cope with the number of hidden layers and the split ratio between training, validation and test sets. Moreover, we have defined the general rules we will follow in the hyperparameters optimization. Therefore, the last elements we need to address are the number of hidden nodes, i.e., the number of neurons in the hidden layer, the activation function, the early stopping criterion, and the learning rate.

With respect to the last two elements, we can say that we chose to rely on the default values. In particular, the early stopping is a form of regularization technique used to prevent overfitting when a training iterative method, such as the gradient descent, is applied. The ratio behind is quite simple: once an arbitrarily large number of training epochs has been specified, 10.000 in our case, it is possible to suspend the training procedure when the model performance stops improving in the validation set. Moreover, since we are considering a squared loss function, this can be seen as a sort of L2 boosting technique, as explained in the work of Yao et al. (2005).

The learning rate, instead, is the hyperparameter determining how quickly the model adapts to the observed data; a better definition is provided in Chapter 2. As already said, we decided to rely on the default values for both these two parameters since, as we are going to see, they proved to be capable of providing excellent results. Therefore, the early stopping is set equal to 6, i.e., the learning procedure is terminated when the performance in the validation set does not improve for 6 consecutive epochs, and the learning rate is 0.01, meaning 1%.

Finally, also the number of hidden neurons and the choice of the activation function will be determined by exploiting a trial-and-error approach. Indeed, in the existing literature a universal rule to be followed to optimize these hyperparameters does not exist. In particular, several rules of thumb methods have been defined over time, like for instance the followings:

- Boger and Guterman (1997) suggested that the number of hidden nodes should be between 70% and 90% of the size of the input layer. Then, if this is proved to be inefficient, additional neurons can be added later on;
- Berry and Linoff (1997) developed a different approach according to which the number of hidden layer neurons should be less than twice the number of nodes in the input layer;

[187] For those who are interested, more on this topic can be found by consulting the "stochastic approximation" literature. Consider for instance Robbins and Monro (1951), Ljung and Soderstrom (1986), or Widrow and Stearns (1985).

- Blum (1992) proposed that the size of the hidden layer, in terms on neurons, should be between the input layer and the output layer sizes.

However, these and other rules *"are not considered to be always true because not only the input layer and the output layer decide the size of the hidden layer neurons"* (Karsoliya, 2012). This means that there are many other relevant factors that need to be considered, such as the activation function, the training algorithm employed, and so on. Since the same is true also for the choice of the activation function, we decided to follow the most common approach by performing a trial-and-error development.

To summarize, we will build a one hidden layer artificial neural network which takes as input four explanatory variables, namely moneyness, volatility, time to maturity and interest rate, and provides as output the monetary price of a European call option written on the FTSE MIB index. In doing so, we will try to detect the best model architecture by optimizing different hyperparameters. In particular, nodes between 2 and 30 will be tested relying on cross-validation performed with four different split settings.

Two activation functions will be alternatively employed. These are the two most performing functions already presented in Chapter 2, namely the Sigmoid and the Hyperbolic tangent[188]. Moreover, the hyperparameter optimization will be carried out by testing all the possible combinations, i.e., we will perform a complete grid search, and the parameters will be constantly updated by exploiting the Levenberg-Marquardt algorithm. This means that at the end of each iteration a new vector of weights will be generated, starting from the previous collected information.

The exploited loss function is the Mean Squared Error, and each simulation involves a huge number of epochs, namely 10.000, so that the optimal value can be detected by the model, usually by triggering the early stopping criterion. To reduce the magnitude of the random initialization of the model, three independent experiments, called "runs", will be performed. Finally, once the best model structure will be detected, a comparative analysis with the Black-Scholes-Merton formula will be made in order to show which model is able to provide the best results.

Merging all this information, we get:

$$29_{nodes} * 4_{splits} * 2_{activationFunctions} * 3_{runs} * 10.000_{epochs} = 6.960.000$$

It means that, potentially, we will analyze almost 7 million alternative scenarios in the development of our neural network.

[188] Note that Matlab calls these two functions respectively *LogSig*, Log-Sigmoid transfer function, and *TanSig*, Tan-Sigmoid transfer function. Despite the different names, these are exactly the same functions we defined in Chapter 2.

4.3 Selecting the best model

In the previous paragraph we discussed the rules to follow in the definition of the architecture of our neural network, and the hyperparameters we need to adjust. In particular, we need a performance metric that allows us to determine which is the best structure among those examined. With this regard, we already know that the learning procedure is performed by means of a MSE loss function; however, we will use a different value to assess the goodness of the analyzed model: the RMSE, namely the Root Mean Squared Error.

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

As we can see, this is simply the square root of the MSE and the reason of using it, instead of the mean squared error, is to obtain a measure which has the same measurement unit as the explanatory variables[189].

The first element we will define is the number of hidden nodes. Then, we will detect the best activation function and the most performing split to use. In this selection process, we will perform a complete analysis of the search space, i.e., we will test all the possible combinations of our parameters, and we will rely on the RMSE, but in a sort of "strange" way. Indeed, in addition to considering its pure value, we will try to deeply understand its behaviour. In order to do this, we will try to answer several questions, like for instance: does it converge from a certain point on? Is it possible to detect a robust value, meaning a point around which always acceptable results are grouped? What is the relationship between training and validation results? And so on.

Table 8, Table 9, and Table 10 show the results provided by our experiments.

[189] Therefore, we want to obtain an evaluation metric which is simpler to interpret.

		Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Activation Function	Nodes	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T
LogSig	2	208,88	212,56	1,018	241,63	234,05	0,969	330,88	336,63	1,017	238,99	246,33	1,031
TanSig		288,57	263,16	0,912	241,91	234,31	0,969	246,88	206,16	0,835	331,73	266,76	0,804
LogSig	3	251,42	229,23	0,912	301,85	343,30	1,137	368,60	351,52	0,954	258,10	274,95	1,065
TanSig		182,85	175,83	0,962	293,44	330,56	1,126	294,28	358,33	1,218	249,60	267,15	1,070
LogSig	4	169,69	193,23	1,139	126,30	127,74	1,011	246,26	174,15	0,707	249,97	191,90	0,768
TanSig		111,57	118,90	1,066	128,17	130,14	1,015	215,31	233,83	1,086	197,53	170,35	0,862
LogSig	5	198,78	192,85	0,970	122,86	150,96	1,229	271,40	258,18	0,951	100,66	77,99	0,775
TanSig		177,87	167,49	0,942	100,44	122,46	1,219	97,62	125,48	1,285	109,12	110,12	1,009
LogSig	6	86,91	91,72	1,055	113,76	108,11	0,950	86,88	92,23	1,062	78,57	76,42	0,973
TanSig		175,11	394,42	2,252	190,59	155,33	0,815	187,39	196,44	1,048	137,99	130,96	0,949
LogSig	7	95,67	71,70	0,749	157,02	149,56	0,952	76,09	73,08	0,960	94,73	100,80	1,064
TanSig		84,70	83,82	0,990	114,60	103,68	0,905	93,53	86,58	0,926	84,22	90,31	1,072
LogSig	8	74,79	79,56	1,064	72,98	75,02	1,028	82,92	77,87	0,939	111,20	113,38	1,020
TanSig		85,23	90,06	1,057	83,79	90,67	1,082	110,54	196,19	1,775	107,13	108,57	1,013
LogSig	9	74,17	84,32	1,137	87,21	92,27	1,058	63,40	67,87	1,070	83,99	96,49	1,149
TanSig		61,73	76,38	1,237	70,53	70,93	1,006	63,22	67,25	1,064	64,32	66,40	1,032
LogSig	10	72,56	76,25	1,051	70,09	72,78	1,038	93,19	75,11	0,806	63,93	57,44	0,898
TanSig		161,98	205,75	1,270	76,73	79,57	1,037	66,10	73,02	1,105	74,94	75,51	1,008
LogSig	11	56,83	56,84	1,000	69,40	70,91	1,022	57,60	55,13	0,957	75,82	73,07	0,964
TanSig		75,28	88,48	1,175	86,57	84,41	0,975	91,62	90,90	0,992	81,52	85,95	1,054
LogSig	12	57,93	61,46	1,061	63,55	68,70	1,081	72,39	91,40	1,263	75,88	76,25	1,005
TanSig		70,50	91,08	1,292	54,37	59,93	1,102	60,72	61,57	1,014	59,42	66,84	1,125
LogSig	13	68,57	76,99	1,123	68,37	75,48	1,104	52,77	52,98	1,004	62,49	49,73	0,796
TanSig		63,05	66,07	1,048	52,76	54,08	1,025	60,23	63,18	1,049	70,04	68,33	0,976
LogSig	14	58,31	71,28	1,223	67,65	117,48	1,737	63,95	73,16	1,144	61,58	96,68	1,570
TanSig		89,53	104,92	1,172	71,19	79,23	1,113	74,10	70,36	0,950	70,75	101,02	1,428
LogSig	15	139,93	216,51	1,547	57,18	71,38	1,248	64,25	63,46	0,988	65,41	66,49	1,016
TanSig		64,00	70,67	1,104	55,95	60,13	1,075	59,41	60,61	1,020	60,18	62,25	1,035
LogSig	16	52,15	86,89	1,666	71,73	90,53	1,262	59,25	71,22	1,202	47,65	48,75	1,023
TanSig		47,14	61,08	1,296	63,67	76,84	1,207	79,37	81,83	1,031	53,64	57,50	1,072
LogSig	17	46,91	53,63	1,143	49,60	56,37	1,137	52,49	55,94	1,066	111,18	113,98	1,025
TanSig		65,72	78,86	1,200	58,45	59,21	1,013	55,73	56,94	1,022	58,80	59,86	1,018
LogSig	18	68,04	83,62	1,229	84,26	90,90	1,079	55,06	65,00	1,180	47,87	48,58	1,015
TanSig		43,74	44,69	1,022	66,57	78,26	1,176	52,68	54,36	1,032	48,27	51,43	1,065
LogSig	19	82,71	101,68	1,229	45,59	51,81	1,136	56,03	97,21	1,735	66,35	71,52	1,078
TanSig		60,55	73,71	1,217	53,95	54,74	1,015	48,68	49,47	1,016	70,54	73,67	1,044
LogSig	20	63,90	92,86	1,453	57,78	58,23	1,008	56,69	57,32	1,011	50,40	52,88	1,049
TanSig		43,75	70,74	1,617	48,40	50,97	1,053	50,80	52,61	1,036	63,83	67,39	1,056
LogSig	21	51,81	59,64	1,151	51,11	56,76	1,110	50,91	49,85	0,979	55,94	60,50	1,082
TanSig		48,44	50,21	1,036	45,02	48,87	1,086	48,87	61,70	1,263	61,08	65,08	1,065
LogSig	22	65,67	85,13	1,296	66,15	135,15	2,043	43,44	45,29	1,042	81,15	94,04	1,159
TanSig		45,13	52,84	1,171	55,24	62,90	1,139	73,16	140,68	1,923	57,98	66,86	1,153
LogSig	23	40,76	37,26	0,914	58,50	78,89	1,349	48,71	45,81	0,940	56,59	60,31	1,066
TanSig		39,16	47,40	1,210	54,26	112,44	2,072	44,07	51,01	1,158	45,81	84,95	1,854
LogSig	24	54,71	80,85	1,478	51,51	75,42	1,464	44,79	56,45	1,260	52,82	75,14	1,423
TanSig		42,76	43,72	1,022	45,61	61,67	1,352	49,42	85,60	1,732	65,73	82,32	1,252
LogSig	25	53,84	61,60	1,144	45,01	52,30	1,162	49,07	68,14	1,389	50,19	65,89	1,313
TanSig		42,08	49,31	1,172	51,56	82,41	1,598	47,21	55,42	1,174	68,15	78,24	1,148
LogSig	26	38,89	58,77	1,511	60,75	66,59	1,096	56,21	79,84	1,420	53,49	58,44	1,093
TanSig		62,10	65,49	1,055	64,29	61,75	0,961	42,43	66,07	1,557	44,28	61,14	1,381
LogSig	27	48,98	63,93	1,305	72,03	71,21	0,989	55,39	82,18	1,484	70,49	85,00	1,206
TanSig		53,76	54,52	1,014	66,63	98,98	1,486	74,90	90,04	1,202	58,03	57,36	0,988
LogSig	28	54,81	69,89	1,275	54,78	98,03	1,790	47,95	63,47	1,324	62,08	68,99	1,111
TanSig		48,71	54,75	1,124	49,40	65,53	1,326	55,50	53,14	0,957	59,06	72,13	1,221
LogSig	29	44,47	77,55	1,744	50,84	57,90	1,139	55,36	65,76	1,188	43,78	52,06	1,189
TanSig		47,98	62,96	1,312	51,17	63,52	1,241	48,72	60,65	1,245	45,44	49,27	1,084
LogSig	30	42,86	54,46	1,271	41,03	41,13	1,002	52,76	75,58	1,432	45,24	82,11	1,815
TanSig		40,47	49,74	1,229	52,75	54,20	1,028	48,86	65,01	1,331	80,18	123,21	1,537

Table 8 - Results of the first experiment

		Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Activation Function	Nodes	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T
LogSig	2	283,42	270,68	0,955	328,98	324,13	0,985	335,27	395,51	1,180	323,72	346,80	1,071
TanSig		203,21	229,99	1,132	235,50	220,30	0,935	329,01	332,35	1,010	343,99	365,55	1,063
LogSig	3	250,36	276,65	1,105	270,71	287,57	1,062	530,23	577,43	1,089	291,36	281,70	0,967
TanSig		167,48	149,34	0,892	309,65	380,71	1,229	308,41	267,03	0,866	301,38	290,86	0,965
LogSig	4	199,73	185,00	0,926	186,10	231,16	1,242	193,55	205,38	1,061	222,13	202,05	0,910
TanSig		98,71	107,52	1,089	215,34	263,57	1,224	218,23	240,01	1,100	301,69	358,08	1,187
LogSig	5	162,30	180,78	1,114	119,57	124,94	1,045	92,99	92,25	0,992	201,25	218,04	1,083
TanSig		153,31	153,98	1,004	156,58	160,98	1,028	399,75	572,17	1,431	480,88	272,24	0,566
LogSig	6	301,93	523,53	1,734	108,23	122,88	1,135	124,36	316,45	2,545	143,14	154,64	1,080
TanSig		120,35	115,49	0,960	84,36	85,68	1,016	78,24	78,39	1,002	116,13	121,99	1,050
LogSig	7	83,82	85,00	1,014	97,37	91,14	0,936	102,64	104,22	1,015	82,29	101,31	1,231
TanSig		76,23	83,59	1,097	81,88	75,46	0,922	72,90	69,59	0,955	149,48	177,49	1,187
LogSig	8	95,63	103,89	1,086	85,49	87,65	1,025	96,04	106,98	1,114	93,71	117,52	1,254
TanSig		85,58	97,87	1,144	97,04	81,38	0,839	77,52	77,95	1,006	79,18	87,39	1,104
LogSig	9	69,69	64,33	0,923	86,90	85,77	0,987	61,85	57,35	0,927	80,27	84,19	1,049
TanSig		280,16	502,43	1,793	71,63	66,04	0,922	65,91	66,45	1,008	103,75	98,47	0,949
LogSig	10	56,96	53,84	0,945	106,98	125,14	1,170	72,88	71,98	0,988	64,35	94,12	1,463
TanSig		58,12	71,67	1,233	78,62	81,19	1,033	113,17	119,91	1,060	65,17	66,62	1,022
LogSig	11	67,57	72,46	1,072	59,33	70,13	1,182	124,53	128,69	1,033	81,50	90,03	1,105
TanSig		75,93	63,76	0,840	88,29	173,60	1,966	89,35	82,75	0,926	59,79	64,71	1,082
LogSig	12	81,92	207,84	2,537	55,61	68,82	1,238	75,93	95,24	1,254	66,52	68,62	1,032
TanSig		61,27	75,24	1,228	64,36	74,85	1,163	56,88	77,33	1,359	69,74	63,84	0,915
LogSig	13	57,56	57,99	1,007	53,89	78,15	1,450	99,53	117,33	1,179	60,25	61,68	1,024
TanSig		69,16	71,89	1,039	78,00	112,92	1,448	67,06	98,29	1,466	62,37	53,99	0,866
LogSig	14	60,99	62,67	1,027	58,50	61,29	1,048	66,36	67,50	1,017	86,81	127,69	1,471
TanSig		63,80	84,17	1,319	72,05	97,82	1,358	56,10	62,22	1,109	80,36	86,04	1,071
LogSig	15	47,14	52,39	1,111	73,72	106,16	1,440	71,76	73,38	1,023	131,01	79,01	0,603
TanSig		67,83	107,31	1,582	71,21	91,20	1,281	57,18	56,60	0,990	85,36	94,68	1,109
LogSig	16	60,30	63,96	1,061	82,09	104,30	1,271	71,06	85,64	1,205	54,57	59,31	1,087
TanSig		57,10	69,53	1,218	54,76	92,98	1,698	57,39	73,70	1,284	62,08	63,87	1,029
LogSig	17	52,93	56,93	1,076	56,53	81,21	1,437	53,53	58,80	1,098	50,82	54,84	1,079
TanSig		50,78	51,92	1,023	49,94	67,90	1,360	86,86	108,80	1,253	56,99	69,46	1,219
LogSig	18	51,35	68,05	1,325	53,65	69,65	1,298	53,73	62,33	1,160	52,57	54,14	1,030
TanSig		41,94	46,68	1,113	65,68	79,00	1,203	66,20	78,37	1,184	69,94	82,60	1,181
LogSig	19	48,11	49,19	1,023	60,82	61,36	1,009	57,32	122,33	2,134	49,71	55,70	1,120
TanSig		56,36	62,38	1,107	50,02	61,51	1,230	52,82	57,69	1,092	56,64	64,30	1,135
LogSig	20	46,24	54,40	1,176	51,32	53,52	1,043	44,50	51,78	1,164	77,16	74,13	0,961
TanSig		55,51	57,15	1,030	48,10	49,69	1,033	49,36	50,59	1,025	79,59	83,44	1,048
LogSig	21	57,26	83,58	1,460	51,38	95,75	1,864	50,38	56,85	1,129	49,64	58,63	1,181
TanSig		68,14	97,85	1,436	45,24	66,68	1,474	56,28	54,25	0,964	54,60	109,88	2,013
LogSig	22	59,33	82,79	1,395	70,97	72,57	1,023	50,79	60,06	1,182	48,50	49,18	1,014
TanSig		43,16	45,19	1,047	69,77	67,54	0,968	60,15	66,24	1,101	48,69	46,78	0,961
LogSig	23	154,16	327,86	2,127	62,87	87,59	1,393	53,58	106,18	1,982	44,05	62,65	1,422
TanSig		43,59	49,16	1,128	46,93	70,05	1,493	45,01	58,88	1,308	45,76	58,82	1,285
LogSig	24	43,48	52,59	1,210	50,05	54,13	1,081	50,73	58,16	1,146	49,91	65,44	1,311
TanSig		45,91	48,23	1,050	48,18	54,22	1,126	43,34	57,80	1,334	58,92	80,21	1,361
LogSig	25	44,55	61,44	1,379	61,82	71,93	1,163	50,32	69,28	1,377	47,27	71,34	1,509
TanSig		61,12	71,53	1,170	51,72	90,52	1,750	54,03	71,61	1,325	51,74	70,63	1,365
LogSig	26	38,20	48,31	1,265	51,54	55,28	1,073	43,98	47,97	1,091	44,07	55,89	1,268
TanSig		44,49	52,01	1,169	59,76	155,32	2,599	59,68	81,72	1,369	59,37	70,61	1,189
LogSig	27	60,82	76,38	1,256	56,34	66,82	1,186	45,93	55,77	1,214	46,13	74,21	1,609
TanSig		35,48	41,76	1,177	58,68	71,90	1,225	55,67	84,51	1,518	60,63	112,78	1,860
LogSig	28	47,21	65,58	1,389	47,19	67,20	1,424	41,75	49,94	1,196	41,16	39,29	0,955
TanSig		46,15	47,12	1,021	49,23	136,95	2,782	63,03	136,33	2,163	46,44	58,79	1,266
LogSig	29	41,89	48,37	1,155	41,29	43,52	1,054	45,42	48,19	1,061	43,79	54,97	1,255
TanSig		36,80	45,42	1,235	41,82	50,92	1,218	56,48	73,36	1,299	62,88	130,62	2,077
LogSig	30	51,55	52,52	1,019	74,66	102,51	1,373	41,20	46,88	1,138	42,21	40,99	0,971
TanSig		40,07	45,36	1,132	50,65	68,41	1,351	53,85	51,54	0,957	50,40	51,41	1,020

Table 9 - Results of the second experiment

		Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Activation Function	Nodes	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T
LogSig	2	303,53	271,67	0,895	322,77	348,64	1,080	328,10	337,78	1,030	330,91	483,17	1,460
TanSig		350,18	362,69	1,036	322,78	348,66	1,080	330,60	333,15	1,008	245,16	299,71	1,223
LogSig	3	241,98	254,17	1,050	201,62	229,64	1,139	222,30	253,77	1,142	258,01	222,83	0,864
TanSig		252,56	227,54	0,901	229,14	260,77	1,138	272,84	275,17	1,009	257,96	224,30	0,870
LogSig	4	189,16	251,68	1,331	315,53	344,81	1,093	217,01	177,46	0,818	190,33	194,12	1,020
TanSig		193,56	182,17	0,941	139,02	160,10	1,152	140,40	170,95	1,218	216,85	219,80	1,014
LogSig	5	92,28	109,75	1,189	88,66	100,25	1,131	113,42	115,15	1,015	108,54	92,21	0,850
TanSig		136,91	403,21	2,945	151,64	125,38	0,827	98,08	118,05	1,204	103,69	89,00	0,858
LogSig	6	113,85	120,99	1,063	177,92	179,19	1,007	114,52	111,26	0,972	85,58	89,06	1,041
TanSig		78,21	90,02	1,151	85,02	88,15	1,037	85,36	82,03	0,961	81,07	85,35	1,053
LogSig	7	193,25	130,23	0,674	94,05	84,88	0,903	92,23	98,30	1,066	75,15	77,06	1,025
TanSig		68,53	73,41	1,071	110,87	151,35	1,365	120,06	178,72	1,489	106,82	123,35	1,155
LogSig	8	84,97	95,76	1,127	84,48	88,30	1,045	85,76	92,95	1,084	80,02	62,63	0,783
TanSig		112,93	105,63	0,935	116,99	118,59	1,014	114,81	121,20	1,056	71,84	61,21	0,852
LogSig	9	73,83	81,78	1,108	70,88	69,40	0,979	81,25	75,77	0,933	231,83	294,61	1,271
TanSig		68,01	70,47	1,036	96,41	87,56	0,908	73,88	92,08	1,246	72,22	82,47	1,142
LogSig	10	55,13	57,33	1,040	93,99	94,45	1,005	64,69	67,36	1,041	63,91	64,26	1,005
TanSig		130,83	95,09	0,727	68,35	70,20	1,027	67,22	57,77	0,859	76,67	73,22	0,955
LogSig	11	71,96	47,82	0,664	75,62	68,28	0,903	62,18	71,24	1,146	68,95	91,57	1,328
TanSig		58,10	62,91	1,083	62,81	68,09	1,084	58,95	50,98	0,865	72,48	63,47	0,876
LogSig	12	60,18	68,22	1,134	68,28	88,65	1,298	72,71	274,59	3,776	78,74	78,85	1,001
TanSig		68,06	115,84	1,702	82,86	115,94	1,399	66,90	82,01	1,226	64,09	59,86	0,934
LogSig	13	54,32	52,11	0,959	51,83	53,97	1,041	61,50	63,64	1,035	69,94	136,03	1,945
TanSig		73,10	92,11	1,260	67,29	63,55	0,945	81,07	68,17	0,841	67,85	72,31	1,066
LogSig	14	48,37	57,21	1,183	61,21	120,65	1,971	50,36	66,44	1,319	52,80	49,81	0,943
TanSig		57,72	58,56	1,015	71,63	128,17	1,789	54,42	66,38	1,220	74,42	77,59	1,043
LogSig	15	67,01	87,31	1,303	78,67	95,96	1,220	68,83	77,91	1,132	51,90	57,15	1,101
TanSig		48,70	48,38	0,993	61,70	65,89	1,068	50,87	52,93	1,040	62,99	71,57	1,136
LogSig	16	59,40	60,29	1,015	87,11	114,28	1,312	52,07	82,94	1,593	54,48	65,12	1,195
TanSig		59,81	62,14	1,039	67,99	84,79	1,247	60,58	70,38	1,162	73,01	77,56	1,062
LogSig	17	45,02	47,31	1,051	55,58	60,67	1,092	61,73	71,58	1,159	53,33	54,78	1,027
TanSig		54,54	69,58	1,276	62,62	70,74	1,130	74,63	122,86	1,646	51,66	52,81	1,022
LogSig	18	47,06	50,78	1,079	52,37	53,28	1,017	64,55	68,14	1,056	56,10	57,42	1,024
TanSig		46,77	55,99	1,197	60,45	63,19	1,045	48,74	50,07	1,027	51,18	53,84	1,052
LogSig	19	42,52	46,50	1,094	51,79	54,32	1,049	60,42	98,79	1,635	48,71	51,62	1,060
TanSig		55,87	55,89	1,000	44,13	46,13	1,045	53,13	62,67	1,179	51,35	57,86	1,127
LogSig	20	51,02	57,69	1,131	47,06	56,12	1,193	70,11	98,86	1,410	66,98	68,11	1,017
TanSig		47,93	59,18	1,235	53,89	68,41	1,269	63,24	69,89	1,105	59,98	76,67	1,278
LogSig	21	44,05	66,25	1,504	43,53	52,03	1,195	42,00	47,26	1,125	47,82	56,21	1,175
TanSig		53,53	57,86	1,081	52,75	63,95	1,212	75,55	82,30	1,089	52,93	63,46	1,199
LogSig	22	44,26	56,54	1,277	54,23	100,16	1,847	50,47	59,63	1,182	48,34	46,11	0,954
TanSig		51,39	63,97	1,245	60,39	63,00	1,043	63,88	65,13	1,020	49,20	57,11	1,161
LogSig	23	43,74	53,17	1,216	52,65	57,71	1,096	53,04	143,31	2,702	45,33	51,51	1,136
TanSig		41,93	39,19	0,935	57,71	61,94	1,073	51,35	59,08	1,151	49,08	72,73	1,482
LogSig	24	56,65	84,33	1,489	59,19	88,86	1,501	52,72	109,79	2,082	46,53	64,15	1,379
TanSig		56,06	79,30	1,415	74,84	106,33	1,421	45,79	75,33	1,645	46,35	40,65	0,877
LogSig	25	40,00	47,43	1,186	43,38	68,34	1,575	47,40	68,12	1,437	48,25	51,50	1,067
TanSig		43,35	49,74	1,147	54,57	71,99	1,319	55,03	63,44	1,153	45,51	73,88	1,623
LogSig	26	41,69	60,47	1,451	53,60	78,14	1,458	58,56	58,84	1,005	44,48	60,42	1,358
TanSig		53,62	67,40	1,257	63,86	85,14	1,333	62,51	78,99	1,264	47,16	47,18	1,001
LogSig	27	35,13	37,50	1,068	48,22	67,72	1,404	44,09	64,46	1,462	53,49	84,52	1,580
TanSig		38,40	45,78	1,192	58,41	116,25	1,990	60,06	65,62	1,093	53,55	72,54	1,355
LogSig	28	38,78	42,32	1,091	55,76	66,87	1,199	47,85	48,21	1,008	57,57	54,96	0,955
TanSig		59,33	64,47	1,087	43,00	57,74	1,343	52,06	49,55	0,952	52,82	87,30	1,653
LogSig	29	65,65	83,62	1,274	51,60	66,87	1,296	44,07	40,75	0,925	42,72	43,44	1,017
TanSig		37,46	83,71	2,235	46,47	64,20	1,381	48,59	63,99	1,317	42,32	63,92	1,510
LogSig	30	52,80	71,83	1,360	45,11	53,14	1,178	57,87	64,69	1,118	42,15	82,66	1,961
TanSig		40,54	43,02	1,061	39,76	45,62	1,148	49,02	64,71	1,320	42,47	58,47	1,377

Table 10 - Results of the third experiment

Table 11 instead provides the average of the results of the three experiments.

		Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Activation Function	**Nodes**	**RMSE Training**	**RMSE Validation**	**Ratio V/T**	**RMSE Training**	**RMSE Validation**	**Ratio V/T**	**RMSE Training**	**RMSE Validation**	**Ratio V/T**	**RMSE Training**	**RMSE Validation**	**Ratio V/T**
LogSig	2	265,28	251,64	0,96	297,79	302,27	1,01	331,42	356,64	1,08	297,87	358,76	1,19
TanSig		280,65	285,28	1,03	266,73	267,76	0,99	302,16	290,55	0,95	306,96	310,67	1,03
LogSig	3	247,92	253,35	1,02	258,06	286,84	1,11	373,71	394,24	1,06	269,16	259,83	0,97
TanSig		200,97	184,24	0,92	277,41	324,01	1,16	291,84	300,18	1,03	269,65	260,77	0,97
LogSig	4	186,19	209,97	1,13	209,31	234,57	1,12	218,94	185,66	0,86	220,81	196,02	0,90
TanSig		134,61	136,20	1,03	160,85	184,60	1,13	191,31	214,93	1,13	238,69	249,41	1,02
LogSig	5	151,12	161,13	1,09	110,36	125,38	1,13	159,27	155,20	0,99	136,82	129,41	0,90
TanSig		156,03	241,56	1,63	136,22	136,27	1,02	198,48	271,90	1,31	231,23	157,12	0,81
LogSig	6	167,56	245,41	1,28	133,30	136,73	1,03	108,59	173,31	1,53	102,43	106,70	1,03
TanSig		124,56	199,98	1,45	119,99	109,72	0,96	117,00	118,95	1,00	111,73	112,77	1,02
LogSig	7	124,25	95,64	0,81	116,15	108,52	0,93	90,32	91,87	1,01	84,05	93,06	1,11
TanSig		76,49	80,27	1,05	102,45	110,16	1,06	95,50	111,63	1,12	113,51	130,38	1,14
LogSig	8	85,13	93,07	1,09	80,98	83,66	1,03	88,24	92,60	1,05	94,98	97,84	1,02
TanSig		94,58	97,85	1,05	99,27	96,88	0,98	100,96	131,78	1,28	86,05	85,72	0,99
LogSig	9	72,56	76,81	1,06	81,67	82,48	1,01	68,84	67,00	0,98	132,03	158,43	1,16
TanSig		136,64	216,43	1,36	79,52	74,84	0,95	67,67	75,26	1,11	80,10	82,45	1,04
LogSig	10	61,55	62,47	1,01	90,35	97,45	1,07	76,92	71,48	0,94	64,06	71,94	1,12
TanSig		116,98	124,17	1,08	74,57	76,99	1,03	82,16	83,57	1,01	72,26	71,79	1,00
LogSig	11	65,45	59,04	0,91	68,12	69,78	1,04	81,44	85,02	1,05	75,43	84,89	1,13
TanSig		69,77	71,72	1,03	79,22	108,70	1,34	79,97	74,88	0,93	71,26	71,38	1,00
LogSig	12	66,68	112,50	1,58	62,48	75,39	1,21	73,68	153,74	2,10	73,72	74,58	1,01
TanSig		66,61	94,05	1,41	67,20	83,58	1,22	61,50	73,64	1,20	64,42	63,51	0,99
LogSig	13	60,15	62,36	1,03	58,03	69,20	1,20	71,27	77,98	1,07	64,23	82,48	1,25
TanSig		68,44	76,69	1,12	66,01	76,85	1,14	69,45	76,55	1,12	66,75	64,88	0,97
LogSig	14	55,89	63,72	1,14	62,45	99,81	1,59	60,22	69,03	1,16	67,06	91,39	1,33
TanSig		70,35	82,55	1,17	71,62	101,74	1,42	61,54	66,32	1,09	75,18	88,22	1,18
LogSig	15	84,70	118,74	1,32	69,85	91,16	1,30	68,28	71,58	1,05	82,77	67,55	0,91
TanSig		60,17	75,45	1,23	62,95	72,41	1,14	55,82	56,72	1,02	69,51	76,17	1,09
LogSig	16	57,28	70,38	1,25	80,31	103,04	1,28	60,79	79,93	1,33	52,23	57,73	1,10
TanSig		54,68	64,25	1,18	62,14	84,87	1,38	65,78	75,30	1,16	62,91	66,31	1,05
LogSig	17	48,29	52,62	1,09	53,90	66,09	1,22	55,92	62,11	1,11	71,78	74,53	1,04
TanSig		57,01	66,79	1,17	57,01	65,95	1,17	72,41	96,20	1,31	55,82	60,71	1,09
LogSig	18	55,48	67,48	1,21	63,43	71,28	1,13	57,78	65,16	1,13	52,18	53,38	1,02
TanSig		44,15	49,12	1,11	64,23	73,49	1,14	55,87	60,93	1,08	56,46	62,63	1,10
LogSig	19	57,78	65,79	1,12	52,73	55,83	1,06	57,92	106,11	1,83	54,92	59,61	1,09
TanSig		57,59	63,99	1,11	49,37	54,13	1,10	51,54	56,61	1,10	59,51	65,28	1,10
LogSig	20	53,72	68,32	1,25	52,05	55,96	1,08	57,10	69,32	1,19	64,85	65,04	1,01
TanSig		49,06	62,36	1,29	50,13	56,36	1,12	54,47	57,70	1,06	67,80	75,83	1,13
LogSig	21	51,04	69,82	1,37	48,67	68,18	1,39	47,76	51,32	1,08	51,13	58,45	1,15
TanSig		56,71	68,64	1,18	47,67	59,83	1,26	60,23	66,08	1,11	56,20	79,47	1,43
LogSig	22	56,42	74,82	1,32	63,78	102,63	1,64	48,23	54,99	1,14	59,33	63,11	1,04
TanSig		46,56	54,00	1,15	61,80	64,48	1,05	65,73	90,68	1,35	51,96	56,92	1,09
LogSig	23	79,55	139,43	1,42	58,00	74,73	1,28	51,78	98,43	1,87	48,66	58,16	1,21
TanSig		41,56	45,25	1,09	52,97	81,48	1,55	46,81	56,32	1,21	46,88	72,17	1,54
LogSig	24	51,61	72,59	1,39	53,59	72,80	1,35	49,41	74,80	1,50	49,75	68,24	1,37
TanSig		48,24	57,08	1,16	56,21	74,07	1,30	46,18	72,91	1,57	57,00	67,73	1,16
LogSig	25	46,13	56,82	1,24	50,07	64,19	1,30	48,93	68,52	1,40	48,57	62,91	1,30
TanSig		48,85	56,86	1,16	52,61	81,64	1,56	52,09	63,49	1,22	55,13	74,25	1,38
LogSig	26	39,59	55,85	1,41	55,29	66,67	1,21	52,92	62,22	1,17	47,35	58,25	1,24
TanSig		53,41	61,63	1,16	62,64	100,74	1,63	54,87	75,59	1,40	50,27	59,64	1,19
LogSig	27	48,31	59,27	1,21	58,86	68,58	1,19	48,47	67,47	1,39	56,70	81,24	1,46
TanSig		42,55	47,35	1,13	61,24	95,71	1,57	63,54	80,06	1,27	57,40	80,89	1,40
LogSig	28	46,93	59,26	1,25	52,58	77,37	1,47	45,85	53,87	1,18	53,60	54,41	1,01
TanSig		51,40	55,44	1,08	47,21	86,74	1,82	56,86	79,67	1,36	52,78	72,74	1,38
LogSig	29	50,67	69,85	1,39	47,91	56,10	1,16	48,28	51,56	1,06	43,43	50,16	1,15
TanSig		40,75	64,03	1,59	46,49	59,54	1,28	51,26	66,00	1,29	50,21	81,27	1,56
LogSig	30	49,07	59,60	1,22	53,60	65,59	1,18	50,61	62,38	1,23	43,20	68,59	1,58
TanSig		40,36	46,04	1,14	47,72	56,08	1,18	50,58	60,42	1,20	57,68	77,70	1,31

Table 11 – Average results

By analyzing these results, it is possible to draw some preliminary observations. First of all, we can say that the learning process seems to work in the proper, expected way. Indeed, we can see a relatively high RMSE in both training and validation sets, for all the tested splits, when the model complexity is low. This is a typical behaviour for machine learning algorithms, which can be explained by the concepts of underfitting and overfitting.

In particular, when the network has only few nodes it is too simple to be able to correctly understand the existing relationship that maps the inputs to the output; in other words, it will be characterized by a high bias. This means that there is room for improvement. Therefore, by increasing the complexity of the structure of the model, we observe that the difference between the predicted values and the true results, expressed in terms of the RMSE, tends to decrease. However, from a certain moment on, this relationship is not true anymore. On the contrary, the error seems to stabilize, and, in a sense, it even starts to increase a bit. This means we are introducing too many parameters and the network has begun to mistake the noise present in the data for actual information. Note that these overcomplex models will be prone to overfitting and they will tend to be characterized by a high variance.

Before going further on, it may be useful to provide a graphical representation of such phenomenon. In particular, since we are now interested in evaluating the overall performance of the model to select the right number of neurons for the hidden layer, instead of plotting a curve for each possible sample[190], it may be more efficient to analyze the mean values directly. It means that Figure 24, Figure 25 and Figure 26 represent, for each of the two activation functions, the average behaviour of the trained neural networks. In particular, the curves that we can observe are obtained without making distinction between the different split settings, but, instead, considering all the data of each experiment as generated by the same "general" split. This has been done to analyze in a synthetic way if the learning procedure has been performed properly. Then, Figure 27 presents the average behaviour of the three simulations. Indeed, it was obtained by merging the previous results and computing their mean values, and it aims at providing a further generalization.

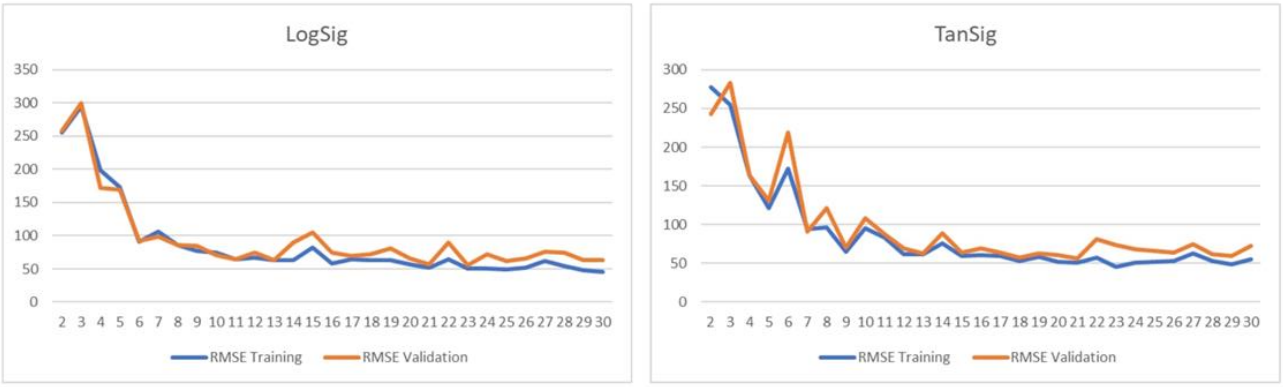

Figure 24 - First experiment – RMSE for different activation functions, computed by averaging the provided split settings

[190] Each sample consisting in the training and validation sets, and the specific split and activation function. This means that if we want to plot everything, we need 24 different figures: $4_{splits} * 2_{activationFunctions} * 3_{runs}$.

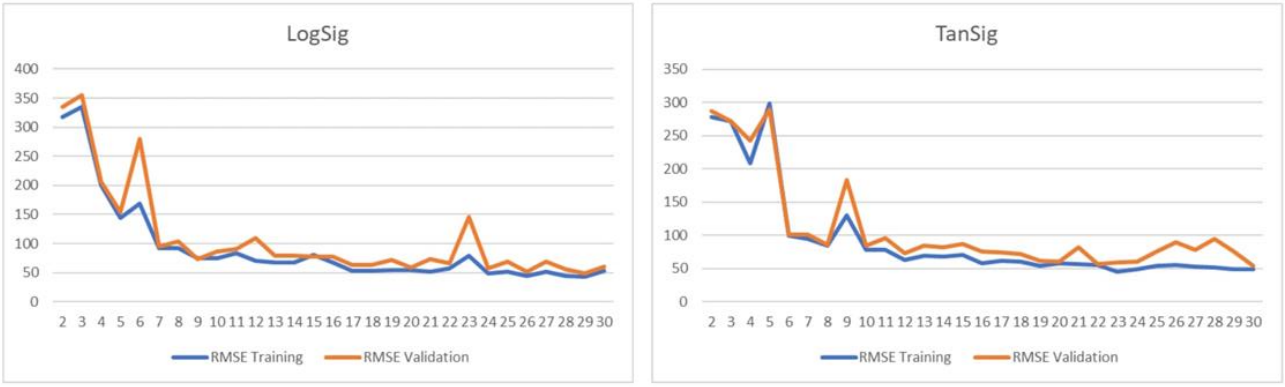

Figure 25 - Second experiment – RMSE for different activation functions, computed by averaging the provided split settings

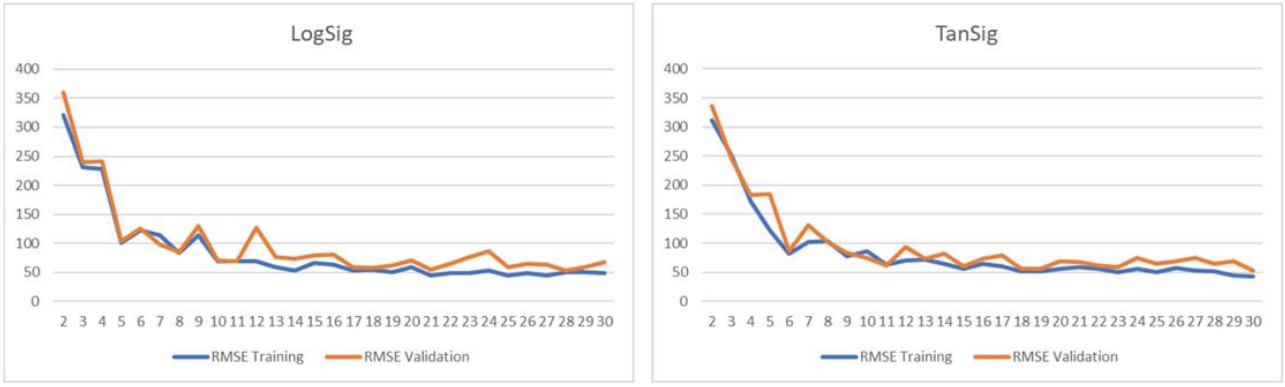

Figure 26 - Third experiment – RMSE for different activation functions, computed by averaging the provided split settings

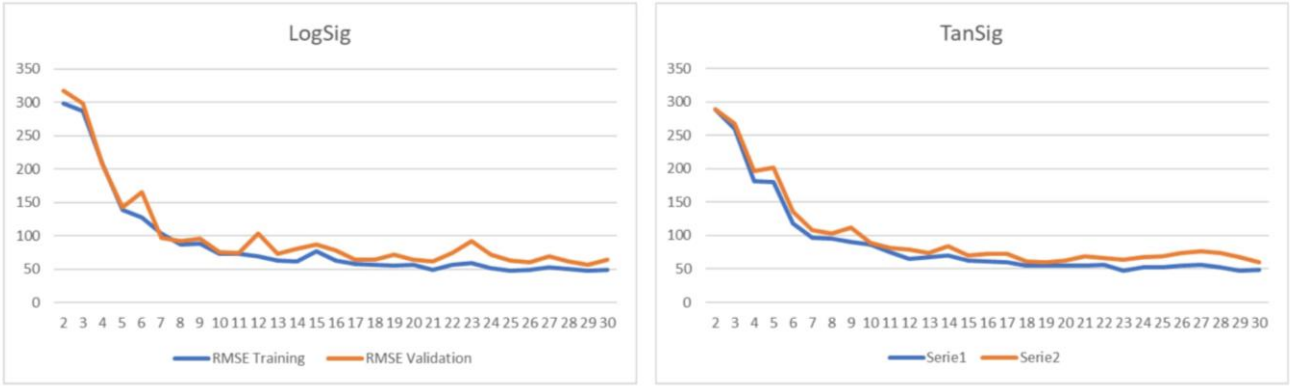

Figure 27 - Average computed with respect to the three experiments

Therefore, now, it should be clear what these Figures show, i.e., the average RMSE, computed as the mean of every split setting for each experiment, changes by increasing the number of hidden nodes.

Fortunately, all these plots show a decreasing behaviour, meaning that, by adding some hidden neurons, the model becomes able to generate new weights and biases which make it more capable to explain the existing relationship between the features and the output. Moreover, by doing so, the network is also able to account for interactions between the explanatory variables. However, while the RMSE in the training set ended up by reaching a sort of plateau, i.e., excluding minimal variations it becomes constant, the RMSE of the validation set, after having minimized its distance with the training results, starts to spread again this distance, settling on slightly higher values. This means that, by continuing to add neurons, the system becomes less capable of generalizing when it has to address unseen data. That is the reason why it is necessary to conduct such analyses: to avoid both underfitting and overfitting and to solve in the best possible way the bias-variance trade-off.

We have said that the previous figures are averages, while the tables show all the available results. Why is that? The reason is simple: since we are interested in finding the best possible solution, we have to evaluate both the specific and the general performances of the network. Therefore, while an individual analysis can be performed relying on the tables, an overall evaluation can be conducted exploiting the information provided by the figures. Indeed, the averages can be considered a good approximation of the general observable behaviour[191]. Therefore, what can be said from these results? The first thing we notice is that the best model is probably around the middle point, i.e., we have to privilege a structure neither too simple nor too complex. Indeed, on one hand, we observe an almost constantly decreasing behaviour of both the training and validation RMSE when increasing the number of neurons from 2 up to 13/14.

The problem is that, even considering such a complex structure, it is still possible to detect some strange peaks in, more or less, all the available cases. We definitely do not like this circumstance, which can be interpreted as a signal: probably it is possible to get better results. On the other side, we see that when the number of neurons is very high, namely above 21/22, the validation performance begins to deteriorate, meaning that it does not remain constant but starts to show higher values.

From these results we can say that the best structure should be between 15 and 20 neurons. In support of this hypothesis, we need to highlight that in general we do not want to pick the single fittest choice, but instead the one that appears to be the most robust. It means that a good practice consists in taking an intermediate value between a range of optimal solutions, and not the best value in absolute terms. This is because a single extremely good value may be due to chance, whereas by considering the optimal interval we ensure a sort of consistency that supports our choice. Therefore, the selected model will involve a number of neurons in the hidden layer equal to 18. Indeed, this has proved to be one of the most resistant among all the possible values.

In particular, if we consider all the three experiments, the mean of the RMSE for the training set is equal to 56.19, and the average of the RMSE for the validation set is 62.93. This represents an

[191] Note that once the number of nodes will be selected, in order to detect the best activation function and the most performing split, we will analyze each case one by one.

extremely good achievement, both in terms of low variability and predictive power[192]. Moreover, since also the "neighborhood" models tend to provide similar outcomes, this has proved to be a very robust result.

Note that we do not consider lower values because neural networks with 18 hidden nodes overperformed them in almost all the considered cases, whereas we do not take higher values into consideration because, even if they were able to obtain similar results, since the difference was not substantial, it is possible to rely on the so-called Occam's razor principle. According to this idea, if two or more alternative models provide the same results, the simplest one must always be preferred.

Now, we have decided the number of hidden neurons that our model will have. However, before analyzing the activation function and the split setting, we need to make some further observations. Indeed, in our data it is possible to detect some unusual observations. In particular, we can spot that in some training-validation pairs the error computed by the training set is greater than the one of the corresponding validation set. How is it possible? This is probably due to the sampling criteria we adopted.

Since we are interested in testing if a neural network is able to provide accurate estimates to forecast the value of a call option, we need to partition the data in a chronological way. Therefore, the division between training- validation and testing observations is based on time. However, to make sure that the model is able to understand most of the existing patterns and the generating process, the partition between training and validation data is randomly made. With this regard, useful references can be found in the works of Yao et al. (2000) and Ruf and Wang (2020). In particular, since there exists some sort of "time-inhomogeneity" in financial data, i.e., volatility changes over time, price series are not stationary, etc., this approach may improve the general performance. Indeed, we have to remember that the distinction between training and validation is made only to perform the hyperparameter optimization, but the performance of the model will be evaluated only in the test set, which is composed only by unseen observations[193]. This explains why sometimes we find the RMSE of the validation set to be lower than the corresponding RMSE of the training set[194].

Once the number of neurons has been detected, we are left with two relevant questions: which are the best activation function and the best split ratio to use? To provide a correct answer we need to perform a deeper analysis focusing on the selected model.

[192] It should be noted that with a similar value for the validation set the overall performance of such networks lies in the best 10^{th} percentile.

[193] With this regard, a common practice consists in re-training the whole model once its hyperparameters have been optimized. Note that in this final training procedure, no validation set is considered.

[194] That is because the validation set has encountered easier to predict values.

		Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Activation Function	Experiment	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validation	Ratio V/T
LogSig	1	68,04	83,62	1,229	84,26	90,90	1,079	55,06	65,00	1,180	47,87	48,58	1,015
TanSig		43,74	44,69	1,022	66,57	78,26	1,176	52,68	54,36	1,032	48,27	51,43	1,065
LogSig	2	51,35	68,05	1,325	53,65	69,65	1,298	53,73	62,33	1,160	52,57	54,14	1,030
TanSig		41,94	46,68	1,113	65,68	79,00	1,203	66,20	78,37	1,184	69,94	82,60	1,181
LogSig	3	47,06	50,78	1,079	52,37	53,28	1,017	64,55	68,14	1,056	56,10	57,42	1,024
TanSig		46,77	55,99	1,197	60,45	63,19	1,045	48,74	50,07	1,027	51,18	53,84	1,052

Table 12 - Results of the 18 nodes Neural Networks

Table 12 shows the results of the three runs for the 18 neurons networks. However, also in this case, it may be more useful to provide the averages. In Table 13 we can observe exactly this, i.e., the averages computed by activation function first and by split setting then.

	Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Activation Function	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T
LogSig	55,48	67,48	1,21	63,43	71,28	1,13	57,78	65,16	1,13	52,18	53,38	1,02
TanSig	44,15	49,12	1,11	64,23	73,49	1,14	55,87	60,93	1,08	56,46	62,63	1,10

	Split 50-25-25			Split 60-20-20			Split 70-15-15			Split 80-10-10		
Experiment	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Mean Ratio V/T
1	55,89	64,16	1,13	75,42	84,58	1,13	53,87	59,68	1,11	48,07	50,00	1,04
2	47,55	56,37	1,17	60,11	73,96	1,24	53,20	58,35	1,10	50,42	52,78	1,05
3	46,64	57,36	1,22	59,67	74,32	1,25	59,96	70,35	1,17	61,25	68,37	1,11

Table 13 - Averages of the 18 nodes Neural Networks

As we can see, the first observation that we can make is that the 60-20-20 split is the least efficient solution. This is in a sense a counterintuitive result since we would have expected the 50-25-25 setting to provide the poorest performance. Indeed, empirical evidence has clearly shown that having a larger training set size is beneficial for the model. However, this has been proved to not be completely true in our sample. Nonetheless, even in our case, we observe that assigning more data to the training process is generally helpful in terms of performances. In particular, we notice that the 70-15-15 and the 80-10-10 cases tend to outperform the others, since they have relatively small values and more consistency. With this regard, we can have a look at their validation/training ratios and observe that they are the lowest.

For this reason, we shrink down our alternatives considering only the splits that assign the largest part to the training set. Among these two possibilities, we have then to choose on which activation function we will rely. In particular, since for the considered samples it provides on average better results, we will exploit the Sigmoid function.

From the previous results, it follows that we will implement a MLP artificial neural network with one hidden layer and 18 hidden nodes. Moreover, the transfer function will be the Sigmoid. With regard to the choice of the split ratio, we can choose between two alternatives: 70-15-15 or 80-10-10. The

existing literature suggests using the first setting, since it allows for a bigger testing set and helps in preventing overfitting; however, even in this case, it is better to test if it is true also empirically.

	Split 70-15-15			Split 80-10-10		
Run	RMSE Training	RMSE Validation	Ratio V/T	RMSE Training	RMSE Validatio	Ratio V/T
1	74,60	104,22	1,397	46,40	54,46	1,174
2	60,15	69,96	1,163	73,14	111,65	1,527
3	58,19	67,87	1,166	48,89	63,91	1,307
4	56,90	59,47	1,045	78,74	103,11	1,310
5	52,92	57,96	1,095	52,60	55,27	1,051
6	53,66	55,96	1,043	58,53	58,77	1,004
7	53,38	54,53	1,022	48,51	68,69	1,416
8	60,39	74,21	1,229	57,30	59,26	1,034
9	53,35	70,28	1,317	59,69	56,34	0,944
10	56,37	62,73	1,113	62,60	63,93	1,021
11	49,86	48,42	0,971	67,97	68,91	1,014
12	59,36	62,26	1,049	57,90	59,76	1,032
13	53,23	57,25	1,075	64,66	68,87	1,065
14	50,81	57,15	1,125	62,30	87,51	1,405
15	61,65	72,54	1,177	50,59	57,25	1,132
16	51,37	54,66	1,064	50,41	60,51	1,200
17	47,18	48,15	1,020	53,65	54,38	1,014
18	49,08	51,85	1,056	65,19	88,88	1,363
19	49,32	52,98	1,074	55,15	57,27	1,038
20	90,51	93,48	1,033	52,71	49,70	0,943
21	48,51	53,46	1,102	51,77	65,07	1,257
22	57,41	59,38	1,034	62,90	76,24	1,212
23	50,95	51,09	1,003	116,37	126,88	1,090
24	66,61	77,01	1,156	56,15	57,46	1,023
25	51,98	57,52	1,107	75,10	90,86	1,210
26	52,99	55,12	1,040	68,70	86,02	1,252
27	53,74	57,78	1,075	66,24	109,38	1,651
28	58,54	66,72	1,140	51,53	59,76	1,160
29	58,34	67,60	1,159	57,91	59,78	1,032
30	68,32	69,43	1,016	56,41	85,99	1,524
31	60,69	90,60	1,493	53,73	54,89	1,022
32	54,69	74,23	1,357	50,68	51,79	1,022
33	49,24	54,63	1,109	49,15	49,90	1,015
34	52,91	58,62	1,108	65,80	80,07	1,217
35	53,45	60,81	1,138	53,70	58,80	1,095
36	80,11	90,92	1,135	72,70	71,58	0,985
37	55,92	59,77	1,069	48,61	54,72	1,126
38	45,56	47,52	1,043	49,72	59,13	1,189
39	55,13	63,20	1,147	52,24	52,16	0,998
40	48,51	49,15	1,013	60,92	65,06	1,068
41	52,73	69,18	1,312	45,46	51,56	1,134
42	49,79	51,18	1,028	51,31	53,14	1,036
43	58,84	64,64	1,098	50,23	64,38	1,282
44	46,75	47,49	1,016	48,58	49,88	1,027
45	66,25	76,65	1,157	72,82	58,28	0,800
46	54,96	63,51	1,156	52,15	57,64	1,105
47	84,78	98,67	1,164	49,49	58,82	1,188
48	66,19	64,79	0,979	49,70	52,30	1,052
49	68,81	73,71	1,071	50,51	52,13	1,032
50	53,61	56,86	1,060	60,92	65,32	1,072

Table 14 - Testing alternative split compositions

Table 14 shows the results of 50 simulations of our neural network in which we test the two alternative split settings. First of all, we need to clarify why we chose to perform so many simulations. The main reason that prompted us to do it is that in this way we are able to rely on the Central Limit Theorem. This means that, by considering a large size sample of independent variables such as the one we have defined, we are sure[195] that the distribution of the sample mean is normal. Therefore, we know for sure that the mean and the median coincide, we can compute the sample standard deviation, which are equal to 13.35 and 17.62 respectively[196], and so on.

Starting from the obtained results, we can highlight once again the robustness of the model. Indeed, this is able to provide a satisfactory performance in all the provided cases. In particular, we have to mention that there are some relatively high values, especially in the 80-10-10 scenario. However, the averages are extremely good, and they support what has been said so far, as we can see in Table 15.

Split 70-15-15			Split 80-10-10		
Mean RMSE Training	Mean RMSE Validation	Ratio V/T	Mean RMSE Training	Mean RMSE Validation	Ratio V/T
57,37	64,14	1,11	58,37	66,55	1,14

Table 15 – Alternative split composition averages

Therefore, by exploiting the collected information, it is possible to say that the architecture that has proved to be the most performing is the one that relies on the 70-15-15 split.

4.4 Final results and comparison of competing models

Once the model has been properly trained, it is possible to evaluate its performance and to make a comparison with the results provided by the Black-Scholes-Merton model. However, in order to do this, we need to define some additional performance measures through which to test the goodness of the competing systems. In particular, we will still rely on the Root Mean Squared Error since it is the most popular metrics to perform such kinds of evaluations.

We already mentioned the advantages of using this type of function, i.e., it is convex, differentiable, the errors have the same unit of the explanatory variables, etc. However, it should be clear that the RMSE alone is not sufficient to evaluate a model; therefore, we need to identify other useful measures and, in order to do it, we can rely on previous studies and exploit some useful empirical results.

[195] This is true independently of how the population from where the sample is extracted is distributed.
[196] These are the values of the validation set and they are useful because we can exploit them to test different hypotheses or to construct confidence interval for the analyzed parameters, e.g. the sample mean.

Indeed, by analyzing the paper of Ruf and Wang (2020) it is possible to understand which are the most common performance metrics applied to evaluate the accuracy of a neural network. From those results it is possible to draw some interesting conclusions. In particular, we can notice that MAE, MAPE, and R^2 are, by far, the most popular exploited functions other than MSE and RMSE. Let us see them in detail, starting with the MAE:

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

where y_i is the actual value and $\hat{y}_i$ is the predicted value.

The Mean Absolute Error, also known as the L1 loss function, is one of the simplest to understand, yet most used evaluation metric. As we can see, it is the arithmetic average of the absolute errors. In this sense, it is a function that quantifies the magnitude of the errors, ignoring at the same time their direction. Of course, the lower the value, the better the estimate.

This performance measure has some important advantages; in particular it is very easy to compute, and the computed errors all have the same weight, meaning that MAE does not penalize larger errors. However, this may also be seen as a drawback of the model, depending on how we interpret it. Indeed, MAE treats big and small errors in the same way. The problem is that we usually do not want to have such a feature, because we prefer more robust and more consistent models, i.e., models that are penalized for committing larger mistakes. Moreover, this function is not differentiable and therefore it cannot be used in backpropagation or similar algorithms unless complex adjustments are implemented.

With this regard, it is possible to define a sort of refined version of MAE: the Mean Absolute Percentage Error, MAPE.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} * 100\%$$

From its formulation, we immediately notice that this loss function is extremely similar to MAE. Indeed, these two measures provide the same kind of information but in a slightly different way. In particular, MAPE is a weighted average of the absolute errors and therefore, it introduces a weighting parameter which is not present in MAE. Thanks to this adjustment it is possible to express MAPE in percentage terms, condition that makes it easier to interpret.

The main advantage of such a metric is the fact that, since all the errors are normalized, it is completely independent of the scale of the variables and consequently very easy to understand. The drawbacks are similar to the ones of the mean absolute error. In particular, involving an absolute term it does not distinguish between positive and negative errors. Moreover, being a ratio, we may face the “division by zero” problem. However, despite these issues, the MAPE is the second

most used performance metric, following RMSE, in the papers analyzed by Ruf and Wang. This is the reason why we also chose to rely on this metric[197].

A third extremely common metric is the R^2, also known as coefficient of determination. This is a very useful performance measure which can be used to evaluate the precision of a regression-type machine learning model. Specifically, it tries to determine the variability of the predictions, which is explained by the network. How does it work? First of all, let us see the formula and then we will try to understand what it means:

$$R^2 = 1 - \frac{SSE}{TSS}$$

As we can see, the coefficient of determination is essentially a ratio of two terms: SSE and TSS. These are respectively the Residual Sum of Squares and the Total Sum of Squares. They are computed as follows:

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

$$TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where $\bar{y}$ is the sample mean of the data.

Therefore, the R^2 measures how much the model is capable of explaining the variance present in the observations. In general, this provides a useful information which can be seen as complementary to those given by the other above-defined metrics. Indeed, also this function, as the MAPE, is expressed in percentage terms and has a quite straightforward interpretation: the higher the value, the more able to correctly fit the data the model is. In general, with financial data, a kind of rule of thumb can be applied, according to which a coefficient of determination of 0.7 or greater is considered an extremely good result, since it means that the analyzed variables show a high level of correlation.

We have to say that over time many other alternative measures have been suggested. For instance, in recent years, Buehler et al. (2019) proposed to use the Conditional Value at Risk (CVaR) in evaluating the performance of a neural network which tries to price option contracts. In any case, the functions previously described are by far the most exploited ones in real world applications. Therefore, in our experiment we will rely on: RMSE, MAPE and R^2.

Of course, the performance of our artificial neural network alone will not be much informative, i.e., we have to compare it with a given benchmark. As already stated, since we are considering European call options, the Black-Scholes-Merton formula can be applied. In particular, since it is such a relevant reference in the derivative pricing field, this will be our benchmark. In reality, the choice is a pretty popular one; indeed, we should not be surprised in discovering that it is the most widely used. However, it is worth mentioning that other new approaches are nowadays becoming

[197] It should be noted that since MAE and MAPE provide the same kind of information, in the same way of MSE and RMSE, it is possible to rely only on one of the two. Therefore, we choose to exploit MAPE since it has nicer properties.

more and more spread, like for instance the use of volatility pricing models or Greeks as references[198].

At the same time, we have to mention that they have not become the state-of-the-art yet. With this regard, Ruf and Wang (2020) observed that when the benchmark is defined so that it includes both the delta and the vega[199], then the presented ANNs were never able to overperform other networks.

Now that all the necessary preliminary steps have been taken, it is possible to compare the results of our neural network and the ones provided by the competing model.

Let us start by having a look at how the network is able to predict the true output in the training, validation and test sets:

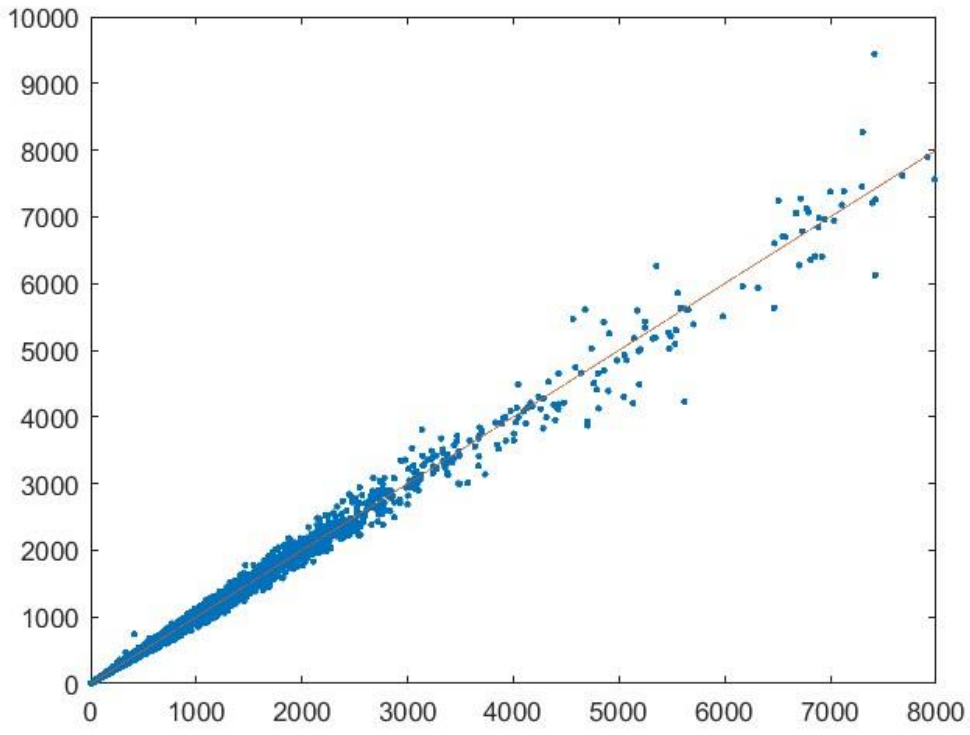

Figure 28 - Performance of the network in the training set

[198] See for instance Gencay and Gibson (2007), Jang and Lee (2019) or Liu et al. (2019)

[199] It means when the benchmark is based on volatility and another Greek.

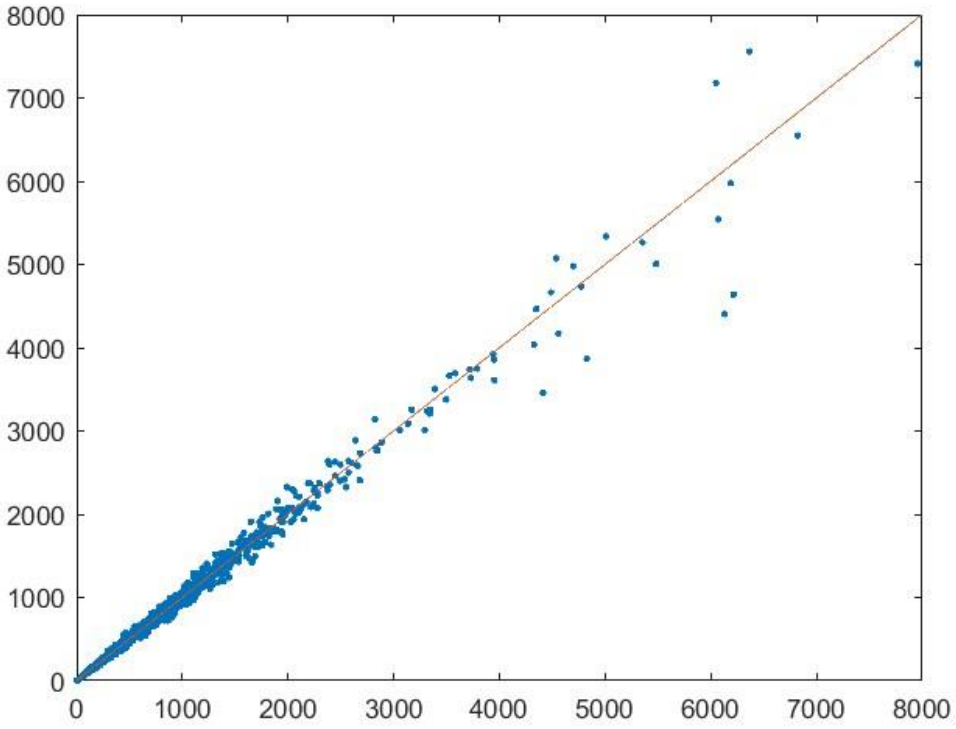

Figure 29 - Performance of the network in the validation set

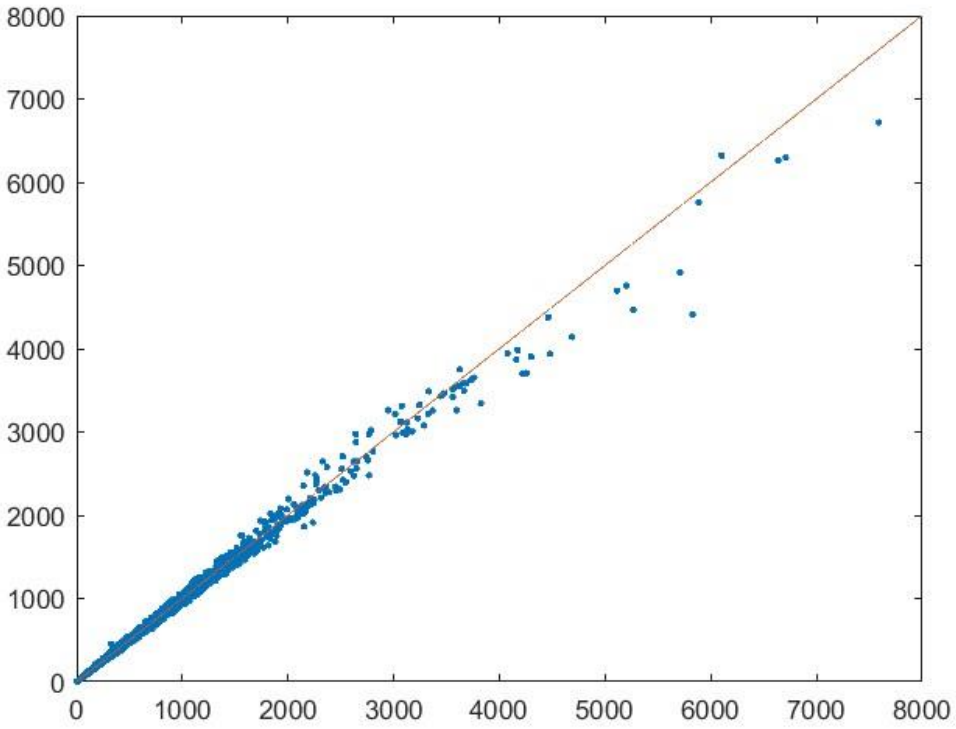

Figure 30 - Performance of the network in the test set

Figure 28, Figure 29 and Figure 30 show on the x-axis the true price of the option and on the y-axis the price predicted by the network. Therefore, the more the points are aligned, the more precise the estimates are. As we can see, the model has been able to provide extremely good results in every analyzed scenario. Nonetheless, it is strange to notice that the training performance seems to be poorer than the one of the test set. However, this is not true. We will see the numerical values in a moment but, in the meantime, we need to keep into account the different numerosity of the three sets.

Indeed, by relying on the 70-15-15 split ratio, we have about 20.000 data in the training set and less than 4.000 in the validation and test groups. Therefore, it is quite normal that we can spot "stranger" observations in the training set simply because it contains much more observations. Another useful note regards the fact that these are all meaningful estimates. Indeed, as we can observe, the model has never predicted negative values.

RMSE Training	RMSE Validation	RMSE Test
53,35	70,28	57,02

Table 16 - RMSE of the Network

The first thing we can notice is that, despite how the graphs look like, the training set is still characterized by the lowest RMSE. Moreover, we are pleased in observing that the model has provided a similar outcome to the ones generated by the other artificial neural networks we tested before. Indeed, this is another confirmation of the robustness of the system.

As we can see, these are also extremely good results. In fact, we have to understand that the average price of a call in the available dataset is greater than 1.000 Euros. It follows that a RMSE of 57.02 means an error lower than 0.057, i.e., 5.7%, in first approximation[200].

An unusual circumstance we have to highlight is the fact that the RMSE of the validation set is larger than the RMSE of the test set. In any case, this is not a serious problem since they are both unseen partitions of the data for the network and since the training error outperforms both the validation and the test errors. Therefore, we can conclude that we are sufficiently sure that the model is not overfitting. Now, let us see how the BSM model has performed on the same data.

[200] In the computation we assume that the average price of a call is exactly equal to 1.000 euros, however, since it is greater, this is a downward estimate.

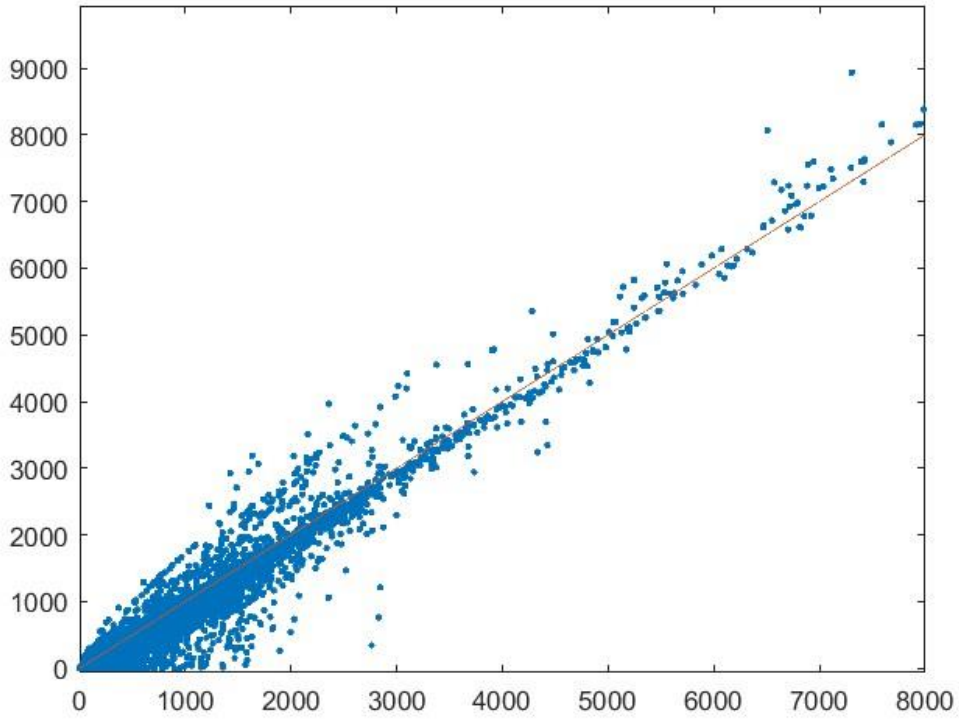

Figure 31 – Performance of the Black-Scholes-Merton model

In Figure 31, we can observe the results provided by the Black-Scholes-Merton model. Even at first glance, the plot seems more confused when compared with the ones provided by the network. However, we must take into account the RMSE for a more precise comparison. In particular, we obtain a RMSE equal to 142.29 in this case.

The first remark we can make is about the precision of the pricing equation. Even if it provides worse results than the network, they are quite good in spite of everything. Indeed, even if we exploited the original version of the model with Merton's adjustment for dividends, and we did not take into account further modifications which have been proved to be able to improve the accuracy of the formula, the predictions look pretty accurate.

A second note regards the noise present in the Figure. We see that it is prevalently concentrated in the first part of the plot and in the last observations. These are options characterized by extremely low and extremely high prices; therefore, we can suppose they are probably out-of-the-money and in-the-money contracts. With this regard, we already know that the BSM model tends to provide good forecasts for ATM options, but biased estimates in the other cases. Keeping that into account, we can state that, overall, the model is quite capable of predicting the future, and this explains why this formula is still so popular among option traders.

In any case, considering all that has been said, we can conclude that our network has been able to provide an astonishing result. Indeed, it has been capable of reducing the regression error of the Black and Scholes model by over 60%.

The RMSE is a good measure, however it is not sufficient to correctly evaluate the performance of the model, meaning that if we want to better understand the general accuracy of our network, we need to rely also on the other functions we have presented.

MAPE Training	MAPE Validation	MAPE Test	MAPE BSM
4,33%	4,46%	4,58%	26,49%

Table 17 - MAPE comparison

Table 17 shows how MAPE behaves in the neural network splits and in the BSM model. In particular, we notice that it supports the previous results. As we can see, the network estimates tend to be much more precise than the ones of the closed-form formula. Indeed, on one hand, we expect that the forecast of the network will differ from the real price of about $\pm 4.6\%$, on average. On the other hand, the difference between the predictions of the BSM model and the true output is expected to be around $\pm 26\%$.

This is a huge achievement in terms of improved performance for our model, but how is it possible that the Black and Scholes equation is so wrong? The problem is that we are considering all types of options at the same time. Therefore, it may be useful to analyze if this behaviour changes when the data are divided in: OTM, ATM, and ITM contracts.

Before doing that, it is necessary to provide a brief comment on the R^2. In particular, the coefficient of determination is equal to 0.9429 for the test set and to 0.8538 for the BSM model[201]. It means that both systems are able to explain most of the variability present in our data, but also that in this case the network outperforms the former pricing equation.

Now, we can divide the test set, i.e., the last 3488 observations, according to moneyness, so that it is possible to analyze the specific performances of the models. In particular, by applying the simple following rule:

$$\begin{cases} ITM\ option & Moneyness \geq 1.05 \\ ATM\ option & 0.95 < Moneyness < 1.05 \\ OTM\ option & Moneyness \leq 0.95 \end{cases}$$

We obtain a sample of 1789 ATM options, 1492 OTM contracts, and 207 ITM derivatives. As already observed, ATM and OTM options are the most traded, while ITM contracts are usually characterized by smaller volumes. Now we can have a look at how the results change.

[201] Note that training and validation are characterized by comparable values.

Model	RMSE OTM	RMSE ATM	RMSE ITM
Network	48,42	52,89	77,96
BSM	63,86	87,01	196,78

Table 18 - RMSE per moneyness

Model	MAPE OTM	MAPE ATM	MAPE ITM
Network	3,87%	4,73%	6,29%
BSM	7,18%	16,93%	29,84%

Table 19 - MAPE per moneyness

As we can notice, the results are not exactly what we expected, but they are still quite informative. In particular, we are surprised that both the neural network and the BSM model have been able to estimate more correctly out-of-the-money options than at-the-money options. However, to try to explain this phenomenon we can recall that, in our dataset, there seem to be more OTM options, see for instance Figure 20 of Chapter 4. Therefore, in a sense, our original dataset appears to be slightly biased. In any case, we see that the differences between the committed errors are quite low both for OTM and ATM contracts.

Another element which may have contributed to determine this result is the fact that we are considering only call options. Indeed, since these are characterized by a smaller implied volatility than other kinds of options, they have probably been less affected by the changing structure of the available data. This means that, since implied volatility has changed over time, the price of the options that embed the smallest amount of it can be determined in a simpler and more precise way because these are in a sense “more deterministic”, or better, less sensible to this random variation.

To conclude, we see that ITM options are the most problematic ones. This is probably due to the small data availability for such kind of contracts. Therefore, a possible way out consists in providing the model with more of these observations. With this regard, it should also be noted that our results partially confirm the “smile bias”. Indeed, ITM options appear to be the most difficult to price and we know that this is mainly due to the large amount of implied volatility they involve. However, this is not completely true since both the presented models have been able to deal with OTM options, whereas according to the volatility smile also these contracts should be difficult to manage due to their low implied volatility. In any case, we are happy to note that the developed artificial neural network has been capable of solving a large part of this problem, being able to correctly price also this kind of derivatives.

Conclusion

The results provided in the previous Chapters allow us to draw some interesting conclusions. In particular, in presenting our final observations, it is better to start by considering the two initial hypotheses.

Hypothesis 1. *It is possible to develop a multilayer perceptron artificial neural network able to correctly price European call options written on the FTSE MIB index.*

According to the results presented in Chapter 4, we can conclude that there is empirical evidence supporting this first hypothesis. Indeed, we have clearly shown that when forecasting the price of a European call option on the FTSE MIB index, it may be useful to rely on a MLP artificial neural network. However, from the previous results it emerges that, depending on the type of contract we are considering, the quality of our estimates varies. In particular, the developed model proved to be much more precise in dealing with out-of-the-money and at-the-money options, with respect to dealing with in-the-money contracts. With this regard, if we normalize the results so that the minimum error is equal to 1, i.e., we divide all the computed RMSE for 48.42, which is the RMSE obtained for OTM options, we notice that the error committed for ATM options is 1.092, meaning it is 9% larger than the one of OTM options, on average, whereas the ITM error is 1.61. This means that, when dealing with ITM contracts, our model provides poorer estimates. As we already mentioned, this is probably due to the small availability of such kind of data points. Indeed, in the testing set, only 207 observations out of 3488 were ITM options, i.e., only 5.9% of the available data.

Hypothesis 2. *Option prices generated by such a model overperform those provided by a traditional pricing formula, i.e., the Black-Scholes-Merton model.*

Even in this second case our outcome strongly supports the hypothesis. Indeed, the neural network model has been able to provide extremely good performances when compared to the ones of the Black-Scholes-Merton model. In particular, the machine learning system has been capable of always committing a smaller error. Moreover, this is true in all the analyzed scenarios. Specifically, the RMSE computed by the network for OTM options has decreased of almost 25% with respect to the one of the BSM model; instead, the one of ATM contracts has shrunk down of about 40%, and finally the error of ITM options has more than halved, diminishing over 60%.

Therefore, the presented results confirm that using an artificial neural network is beneficial for the option pricing task, at least in the Italian equity market case. Moreover, we notice that the system is able to outperform the traditional competing model, and in particular it proved to be capable of partially correcting the volatility smile bias that affects the BSM equation. With this regard, it is interesting to notice that the size of the improvement is not constant, but instead it depends on the type of contract considered. Indeed, the largest contribution of the model regards the correction of the pricing of in-the-money options, but, at the same time, we cannot ignore that the error committed in this case is, in absolute terms, the most serious one. This means that probably we can improve the performance of the network by feeding more data to the model.

Of course, in order to not distort the results, we cannot provide arbitrarily more observations by, for instance, including contracts with a zero-trading volume from the original dataset, otherwise the algorithm will receive biased and distorted data. Therefore, the best solution in this case is to

enlarge the dataset size by increasing the observation period, so that there are more data available and, at the same time, the true existing proportions of the different kind of options are not altered.

We also have to highlight that the fact the network has been able to overperform the BSM model is an important finding, since other papers were not able to achieve such a result. It means that our thesis supports the validity of relying on artificial intelligence, at least in addressing financial regression-type problems. With this regard, we need to mention that the obtained outcomes and the process we followed to derive them raise a number of interesting observations. Therefore, by relying on them, it is possible to suggest some future steps which may be useful to take.

In particular, we already underlined the importance of having a larger training set to correct the presented biases and to make the model more aware of the process generating the observations. With respect to this, it may be worth trying to develop a more complex network, exploiting a deep learning structure based on more than one hidden layer. At the same time, it can be useful to test different combinations of hyperparameters, like for instance trying to exploit alternative training and activation functions. Moreover, the lack of a general acceptable performance measure which can be used to evaluate the accuracy of different machine learning algorithms, emerges from the thesis. It follows that the formulation of alternative metrics capable of measuring the accuracy of such systems is another element which requires additional studies and further research.

Related to the data accessibility, there is the problem regarding the input features availability. We briefly mentioned that other than the presented explanatory variables, it is possible to provide the model with additional informative features, such as the volume or the open interest. To study the effect that introducing such variables has on the overall forecasting ability of the model, is definitely something interesting and that should be done in the future. With this regard, we have to mention that some papers have already started to test the potential improve of the predicting power of a network that includes this information, see for instance Montesdeoca and Niranjan (2016) or Cao et al. (2019), which proposed the use of the underlying return as an additional explanatory variable.

With respect to the input parameters, there are also the problems related to the volatility measure we want to exploit in our model. In particular, we previously presented the different solutions that can be adopted according to which kind of “variability formulation” we choose to provide the model with. The problem is that so far a general approach which has been proved to be always better than the others does not exist, and the variability computation remains a hard task to tackle. Considering how crucial this component is in the determination of the cost of an option, developing models able to provide more precise volatility estimates may be a fundamental step that sooner or later will be necessarily done.

Finally, it may be helpful to understand which is the best machine learning algorithm that can be exploited to solve this kind of problems. In doing so, it may be wise to compare the performance of different models, like for instance Support Vector Machines, when performing this specific task. With this regard, it may be extremely interesting also to develop systems based on other learning paradigms; in particular, testing reinforcement learning algorithms may be a very promising research field.

To summarize, we can say that we have succeeded in demonstrating that neural networks are a powerful tool to address complex problems characterized by a high dimensionality and nonlinear

relationship such as option pricing, therefore additional studies in this sense should be conducted to improve the general performance of this kind of models, since they will probably represent a more and more helpful tool for those interested in trading or evaluating option contacts.

Appendix A – MATLAB Code: Put – Call parity

The first part of the code is about cleaning the data and ordering them, so that we can compare call and put options characterized by the same features, i.e., same maturity, strike, expiration and underlying price.

```
clear
close all
clear all

format long

%Read table
put = readtable('DATI_LORIS_put_puliti');
call = readtable('DATI_LORIS_call_puliti');

%Convert date inputs into numbers
put.Date = datenum(put.Date);
put.Expiration = datenum(put.Expiration);
call.Date = datenum(call.Date);
call.Expiration = datenum(call.Expiration);

%Convert table into matrix
put = table2array(put);
call = table2array(call);

dimPut = size(put, 1);
dimCall = size(call,1);

for i = 1:dimPut
    for k = 1:dimCall
        if put(i,1)== call(k,1) && put(i,6) == call(k,6) && put(i,7) == call(k,7)
            newCall(k,:) = call(k,:);
            newPut(i,:) = put(i,:);

        end
    end
end
newCall(newCall(:,1) == 0,:) = [];
newPut(newPut(:,1) == 0,:) = [];

newCallOrdered = sortrows(newCall, [1 7 6]);
newPutOrdered = sortrows(newPut, [1 7 6]);

dateCall = datetime(newCallOrdered(:,1),'ConvertFrom','datenum');
expirationCall = datetime(newCallOrdered(:,7),'ConvertFrom','datenum');
datePut = datetime(newPutOrdered(:,1),'ConvertFrom','datenum');
expirationPut = datetime(newPutOrdered(:,7),'ConvertFrom','datenum');

callData = array2table(newCallOrdered);
putData = array2table(newPutOrdered);

callData.newCallOrdered1 = dateCall;
callData.newCallOrdered7 = expirationCall;
putData.newPutOrdered1 = datePut;
putData.newPutOrdered7 = expirationPut;


writetable(callData, 'DataCall.xlsx');
writetable(putData, 'DataPut.xlsx');
```

In the second part of the code, instead, we compute the left-hand side and the right-hand side of the put-call parity. Then, we prove that the two sides are comparable in terms of values. This means that the relationship holds.

```
clear all
close all
clear

format long
call_table = readtable('DataCall.xlsx');
put_table = readtable('DataPut.xlsx');

call_table.Date = datenum(call_table.Date);
call_table.Expiration = datenum(call_table.Expiration);
put_table.Date = datenum(put_table.Date);
put_table.Expiration = datenum(put_table.Expiration);

%Convert table into matrix
call_matrix = table2array(call_table);
put_matrix = table2array(put_table);

call_matrix(call_matrix(:,7) == 0, :) = [];
put_matrix(put_matrix(:,7) == 0, :) = [];

dim = size(call_matrix, 1);

%call = call_matrix(:,3) + call_matrix(:,4).*exp(-call_matrix(:,7).*call_matrix(:,5));
%put = put_matrix(:,3) + (put_matrix(:,8)./1000);


left = call_matrix(:,3) - put_matrix(:,3);
right = (put_matrix(:,8)./1000) - call_matrix(:,4)./((1+call_matrix(:,7)).^(call_matrix(:,5)./365));
total = right+left;
relationship = mean(total);

% xlswrite('left.xlsx', left);
% xlswrite('right.xlsx', right);
```

Appendix B – MATLAB Code: Neural Network and BSM model

First of all, we have to prepare the data by removing the observations with a time to maturity lower than a week.

```
clear all
close all
clc

%Remove observations with a maturity lower than a week

T = readtable('DATI_LORIS_call_puliti.xlsx');

data = table2array(T(:,1));
expiration = table2array(T(:,7));

dim = size(T, 1);
CallNoLastWeek = T;
indice=zeros(dim,1);

for i=1:dim
    if data(i) == expiration(i) || data(i) == expiration(i) - 1 ...
            || data(i) == expiration(i) - 2 || data(i) == expiration(i) - 3 ...
            || data(i) == expiration(i) - 4 || data(i) == expiration(i) - 5 ...
            || data(i) == expiration(i)- 6
        indice(i)=1;
    end
end

CallNoLastWeek.indice = indice;

toDeleteCall = CallNoLastWeek.indice == 1;
CallNoLastWeek(toDeleteCall, :) = [];

CallNoLastWeek(:,9)= [];

CallNoLastWeek = sortrows(CallNoLastWeek, [1 7 6]);

writetable(CallNoLastWeek, 'DataCall.xlsx');
```

Then we need to perform the risk-free interpolation. Note that we have to keep into account all the different possible maturities.

```
format long
call_table = readtable('DataCall.xlsx');
risk_free_table = readtable('Curva dei tassi Loris.xlsx');


%Convert date inputs into numbers
call_table.Date = datenum(call_table.Date);
call_table.Expiration = datenum(call_table.Expiration);
risk_free_table.Date = datenum(risk_free_table.Date);

%Convert table into matrix
call_matrix = table2array(call_table);
risk_free_matrix = table2array(risk_free_table);

dim = size(call_matrix, 1);
maturity = [];

for i = 1:dim
    maturity(i, 1) = call_matrix(i, 7) - call_matrix(i, 1);
end

risk_free = zeros(dim, 1);
```

```
%Risk free interpolation

for k = 1:dim
    if maturity(k) == 7 || maturity(k) == 28 || maturity(k) == 29 || maturity(k) == 30 ...
            || maturity(k) == 31 || maturity(k) == 59 || maturity(k) == 60 || maturity(k) == 61 ...
            || maturity(k) == 62 || maturity(k) == 89 || maturity(k) == 90 || maturity(k) == 91 ...
            || maturity(k) == 92 || maturity(k) == 181 || maturity(k) == 182 || maturity(k) == 183 ...
            || maturity(k) == 184 || maturity(k)== 365 || maturity(k) == 366
        current_date = call_matrix(k, 1);
        position = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == maturity(k));
        if position ~= 0
            risk_free(k,1) = risk_free_matrix(position, 3);
        end

    elseif maturity(k)>7 && maturity(k)<28
        current_date = call_matrix(k, 1);
        position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 7);
        R1 = risk_free_matrix(position_R1, 3);
        position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 28);
        if position_R2 ~= 0
            R2 = risk_free_matrix(position_R2, 3);
            risk_free(k,1) = R1 + ((R2 - R1)/(28 - 7))*((maturity(k) - 7));
        else
            position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 29);
            if position_R2 ~= 0
                R2 = risk_free_matrix(position_R2, 3);
                risk_free(k,1) = R1 + ((R2 - R1)/(29 - 7))*((maturity(k) - 7));
            else
                position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 30);
                if position_R2 ~= 0
                    R2 = risk_free_matrix(position_R2, 3);
                    risk_free(k,1) = R1 + ((R2 - R1)/(30 - 7))*((maturity(k) - 7));
                else

                    position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 31);
                    if position_R2 ~= 0
                        R2 = risk_free_matrix(position_R2, 3);
                        risk_free(k,1) = R1 + ((R2 - R1)/(31 - 7))*((maturity(k) - 7));
                    end
                end
            end
        end

    elseif maturity(k)>28 && maturity(k)<59
        current_date = call_matrix(k, 1);
        position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 28);
        if position_R1 ~= 0
            R1 = risk_free_matrix(position_R1, 3);
            t1 = 28;
        else
            position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 29);
            if position_R1 ~= 0
                R1 = risk_free_matrix(position_R1, 3);
                t1 = 29;
            else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 30);
                if position_R1 ~= 0
                    R1 = risk_free_matrix(position_R1, 3);
                    t1 = 30;
                else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 31);
                    if position_R1 ~= 0
                        R1 = risk_free_matrix(position_R1, 3);
                        t1 = 31;
                    end
                end
            end
        end
```

```
    position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 59);
    if position_R2 ~= 0
        R2 = risk_free_matrix(position_R2, 3);
        risk_free(k,1) = R1 + ((R2 - R1)/(59 - t1))*((maturity(k) - t1));
    else
        position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 60);
        if position_R2 ~= 0
            R2 = risk_free_matrix(position_R2, 3);
            risk_free(k,1) = R1 + ((R2 - R1)/(60 - t1))*((maturity(k) - t1));
        else
            position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 61);
            if position_R2 ~= 0
                R2 = risk_free_matrix(position_R2, 3);
                risk_free(k,1) = R1 + ((R2 - R1)/(61 - t1))*((maturity(k) - t1));
            else
                position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 62);
                if position_R2 ~= 0
                    R2 = risk_free_matrix(position_R2, 3);
                    risk_free(k,1) = R1 + ((R2 - R1)/(62 - t1))*((maturity(k) - t1));
                end
            end
        end
    end

elseif maturity(k)>59 && maturity(k)<92
    current_date = call_matrix(k, 1);
    position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 59);
    if position_R1 ~= 0
        R1 = risk_free_matrix(position_R1, 3);
        t1 = 59;

    else
        position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 60);
        if position_R1 ~= 0
            R1 = risk_free_matrix(position_R1, 3);
            t1 = 60;
        else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 61);
            if position_R1 ~= 0
                R1 = risk_free_matrix(position_R1, 3);
                t1 = 61;
            else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 62);
                if position_R1 ~= 0
                    R1 = risk_free_matrix(position_R1, 3);
                    t1 = 62;
                end
            end
        end
    end
    position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 89);
    if position_R2 ~= 0
        R2 = risk_free_matrix(position_R2, 3);
        risk_free(k,1) = R1 + ((R2 - R1)/(89 - t1))*((maturity(k) - t1));
    else
        position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 90);
        if position_R2 ~= 0
            R2 = risk_free_matrix(position_R2, 3);
            risk_free(k,1) = R1 + ((R2 - R1)/(90 - t1))*((maturity(k) - t1));
        else
            position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 91);
            if position_R2 ~= 0
                R2 = risk_free_matrix(position_R2, 3);
                risk_free(k,1) = R1 + ((R2 - R1)/(91 - t1))*((maturity(k) - t1));
            else
                position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 92);
                if position_R2 ~= 0
                    R2 = risk_free_matrix(position_R2, 3);
                    risk_free(k,1) = R1 + ((R2 - R1)/(92 - t1))*((maturity(k) - t1));
                end
            end
        end
    end
```

```
elseif maturity(k)>92 && maturity(k)<184
    current_date = call_matrix(k, 1);
    position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 89);
    if position_R1 ~= 0
        R1 = risk_free_matrix(position_R1, 3);
        t1 = 89;
    else
        position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 90);
        if position_R1 ~= 0
            R1 = risk_free_matrix(position_R1, 3);
            t1 = 90;
        else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 91);
            if position_R1 ~= 0
                R1 = risk_free_matrix(position_R1, 3);
                t1 = 91;
            else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 92);
                if position_R1 ~= 0
                    R1 = risk_free_matrix(position_R1, 3);
                    t1 = 92;
                end
            end


        end
    end
    position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 181);
    if position_R2 ~= 0
        R2 = risk_free_matrix(position_R2, 3);
        risk_free(k,1) = R1 + ((R2 - R1)/(181 - t1))*((maturity(k) - t1));
    else
        position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 182);
        if position_R2 ~= 0
            R2 = risk_free_matrix(position_R2, 3);
            risk_free(k,1) = R1 + ((R2 - R1)/(182 - t1))*((maturity(k) - t1));
        else
            position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 183);
            if position_R2 ~= 0
                R2 = risk_free_matrix(position_R2, 3);
                risk_free(k,1) = R1 + ((R2 - R1)/(183 - t1))*((maturity(k) - t1));
            else
                position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 184);
                if position_R2 ~= 0
                    R2 = risk_free_matrix(position_R2, 3);
                    risk_free(k,1) = R1 + ((R2 - R1)/(184 - t1))*((maturity(k) - t1));
                end
            end
        end
    end


elseif maturity(k)>184 && maturity(k)<366
    current_date = call_matrix(k, 1);
    position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 181);
    if position_R1 ~= 0
        R1 = risk_free_matrix(position_R1, 3);
        t1 = 181;
    else
        position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 182);
        if position_R1 ~= 0
            R1 = risk_free_matrix(position_R1, 3);
            t1 = 182;
        else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 183);
            if position_R1 ~= 0
                R1 = risk_free_matrix(position_R1, 3);
                t1 = 183;
            else position_R1 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 184);
                if position_R1 ~= 0
                    R1 = risk_free_matrix(position_R1, 3);
                    t1 = 184;
                end
            end
        end
    end
```

```
        position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 365);
        if position_R2 ~= 0
            R2 = risk_free_matrix(position_R2, 3);
            risk_free(k,1) = R1 + ((R2 - R1)/(365 - t1))*((maturity(k) - t1));
        else
            position_R2 = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 366);
            if position_R2 ~= 0
                R2 = risk_free_matrix(position_R2, 3);
                risk_free(k,1) = R1 + ((R2 - R1)/(366 - t1))*((maturity(k) - t1));
            end
        end
    elseif maturity(k)>366
        current_date = call_matrix(k, 1);

          position = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 365);
          if position ~= 0
              risk_free(k, 1) = risk_free_matrix(position, 3);
          else
              position = find(risk_free_matrix(:,1) == current_date & risk_free_matrix(:,2) == 366);
              risk_free(k, 1) = risk_free_matrix(position, 3);
          end
      end
end

xlswrite('risk_free.xlsx', risk_free);
xlswrite('maturity.xlsx', maturity);
```

After having performed all these operations, it is possible to retrieve the data for the Black-Scholes-Merton model and for the Artificial Neural Network.

BSM model:

```
%Data for Black-Scholes-Merton model

format long
call_table = readtable('DataCall.xlsx');
risk_free = readtable('risk_free.xlsx');
maturity = readtable('maturity.xlsx');

%Rename columns
risk_free.Properties.VariableNames = ["Risk_free"];
maturity.Properties.VariableNames = ["Maturity"];

%Merge the tables
newTable = [call_table risk_free maturity];

%Convert date inputs into numbers
newTable.Date = datenum(newTable.Date);
newTable.Expiration = datenum(newTable.Expiration);

%Convert tables into matricies
newTable_matrix = table2array(newTable);
newTable_matrix(newTable_matrix(:, 9) == 0, :) = [];

dim = size(newTable_matrix, 1);

dateCall = datetime(newTable_matrix(:, 1),'ConvertFrom','datenum');
expirationCall = datetime(newTable_matrix(:,7),'ConvertFrom','datenum');

callData = array2table(newTable_matrix);

callData.newTable_matrix1 = dateCall;
callData.newTable_matrix7 = expirationCall;

callData.Properties.VariableNames = ["Date", "Last", "Underlying", "Volatility", "Volume", "Strike", "Expiration", "Dividend", "Risk_Free", "Maturity"];

writetable(callData, 'CallForBSM.xlsx');
```

ANN model:

```
%Data for Artificial Neural Network

format long
call_table = readtable('DataCall.xlsx');
risk_free = readtable('risk_free.xlsx');
maturity = readtable('maturity.xlsx');

%Rename columns
risk_free.Properties.VariableNames = ["Risk_free"];
maturity.Properties.VariableNames = ["Maturity"];

%Merge the tables
newTable = [call_table risk_free maturity];

%Convert date inputs into numbers
newTable.Date = datenum(newTable.Date);
newTable.Expiration = datenum(newTable.Expiration);

%Convert tables into matricies
newTable_matrix = table2array(newTable);
newTable_matrix(newTable_matrix(:, 9) == 0, :) = [];

dim = size(newTable_matrix, 1);

moneyness = newTable_matrix(:, 3)./(newTable_matrix(:, 6)./1000);
rate = newTable_matrix(:, 9) - newTable_matrix(:, 8);

newTable_matrix(:, [3 5 6 8 9]) = [];

newTable_matrix = [newTable_matrix moneyness rate];

dateCall = datetime(newTable_matrix(:, 1),'ConvertFrom','datenum');
expirationCall = datetime(newTable_matrix(:,4),'ConvertFrom','datenum');

callData = array2table(newTable_matrix);

callData.newTable_matrix1 = dateCall;
callData.newTable_matrix4 = expirationCall;

callData.Properties.VariableNames = ["Date", "Last", "Volatility", "Expiration", "Maturity", "Moneyness", "Rate"];

writetable(callData, 'Call.xlsx');
```

Then we can implement the two competing systems.

Black-Scholes-Merton model:

```
call = readtable('callForBSM.xlsx');
call(call.Volatility <= 0, :) = [];


S = call.Underlying;        % current stock price (spot price)
K = call.Strike./1000;         % exercise price (strike price)
sigma = call.Volatility;   % volatility of stock
T = call.Maturity/365;       % expiry time in days
rf = (call.Risk_Free./call.Maturity)*365;       % daily risk-free interest rate
d = (call.Dividend./call.Maturity)*365;
r = rf - d;

d1 = (log(S./K) + (r + sigma.^2/2) .* T)./(sigma .* sqrt(T));
d1 = d1(:,1);
d2 = d1 - sigma .* sqrt(T);

%Compute N(d1) and N(d2)

N1 = 0.5 * (1 + erf(d1./sqrt(2)));
N2 = 0.5 * (1 + erf(d2./sqrt(2)));

BSM_price_call = S.*N1 - K.*exp(-r.*T).*N2;
true = call.Last;

RMSE_BSM = sqrt(mean((BSM_price_call - true).^2));
MAPE_BSM = mean((abs(BSM_price_call - true))./true);
MAE_BSM = (sum(abs(BSM_price_call - true)))/size(call,1);

SSE = sum((true - BSM_price_call).^2);
med = mean(true);
TSS = sum((true - med).^2);

R2 = 1- (SSE/TSS);
```

Artificial Neural Network:

```
% MLP

clc
close all
clear all

format long



runs = 3; % number of run
performance = [];
% testpf = [];
```

input-output

```
n_input = 4; % number of inputs
call = readtable('Call.xlsx');

%Summary(call)

%Remove negative volatility (missing values -99.99 by default)

call(call.Volatility <= 0, :) = [];


x = [call.Volatility call.Maturity call.Moneyness call.Rate];
y = [call.Last];
dim = size(call, 1);

%Visualize the data and transform it so that the distribution is more symmetric

histogram(y)
y2 = log(1+y);
histogram(y2)

histogram(x(:,1))
% x1 = 1 + x(:,1);
% x1 = log(x1);
% histogram(x1)
%or cube root
x(:,1) = x(:,1).^(1/3);
histogram(x(:,1))

histogram(x(:,2))
x(:,2) = log(1+x(:,2));
histogram(x(:,2))

histogram(x(:,3))

histogram(x(:,4))
x4 = x(:,4).*(-1);
histogram(x4) %tried log, sqrt and cube root which is the best
x(:,4) = x4.^(1/3);
histogram(x(:,4))
```

```
%Plot relationship between input and output

% plot(x(:,1), y, '.')
% plot(x(:,2), y, '.')
% plot(x(:,3), y, '.')
% plot(x(:,4), y, '.')


%Normalize the features and transform the output

for i = 1:n_input
    x2(:, i) = (x(:,i)-min(x(:,i)))/(max(x(:,i))-min(x(:,i)));
end
```

```
%Check min,max, distribution etc.

% histogram(x2(:,1))
% histogram(x2(:,2))
% histogram(x2(:,3))
% histogram(x2(:,4))
% summary(x2(:,1))
% summary(x2(:,2))
% summary(x2(:,3))
% summary(x2(:,4))

%They are all between 0-1, so it is right. Plot relationship

% plot(x2(:,1), y, '.')
% plot(x2(:,2), y, '.')
% plot(x2(:,3), y, '.')
% plot(x2(:,4), y, '.')
```

Split training and testing sets

```
splitting_ratio = 0.8;
split = round(dim*splitting_ratio);
```

```
x_training = x2(1:split, :);
x_testing = x2(split+1:end, :);

y_training = y2(1:split, :);
y_testing = y2(split+1:end, :);
```

Train the Artificial Neural Network

```
x_trainingT = x_training';
y_trainingT = y_training';
x_testingT = x_testing';
y_testingT = y_testing';

tic

for j = 1:runs
    for k = 2:30

        hiddenLayerSize = k;
        net = fitnet(hiddenLayerSize);
        net.divideParam.trainRatio = 85/100;
        net.divideParam.valRatio = 15/100;
        net.divideParam.testRatio = 0/100;

        % training
        net.trainParam.showWindow =     true;
        net.trainFcn = 'trainlm'; % 'trainlm' 'traingd' 'traingdm' 'traingda' 'traingdx'
        net.performFcn = 'mse';
        net.trainParam.lr = 0.01; % learning rate (default = 0.01)
        %net.trainParam.goal = 1e-15;
        net.trainParam.max_fail = 6; % default = 6
        net.trainParam.epochs = 10000;

        [net, tr] = train(net, x_trainingT, y_trainingT);
        %view(net)


        %Convert back true and predicted outputs

        yTrain = exp(net(x_trainingT(:,tr.trainInd)))-1;
        yVal = exp(net(x_trainingT(:,tr.valInd)))-1;
        yTest = exp(net(x_testingT))-1;

        yTrainTrue = exp(y_trainingT(tr.trainInd))-1;
        yValTrue = exp(y_trainingT(tr.valInd))-1;
        yTestTrue = exp(y_testingT)-1;
```

```
        RMSE_train(k) = sqrt(mean((yTrain - yTrainTrue).^2));
        RMSE_val(k) = sqrt(mean((yVal - yValTrue).^2));
%         RMSE_test(k) = sqrt(mean((yTest - yTestTrue).^2));
        MSE_train(k) = mean((yTrain - yTrainTrue).^2);
        MSE_val(k) = mean((yVal - yValTrue).^2)
        MAPE_train(k) = mean((abs(yTrain - yTrainTrue))./yTrainTrue);
        MAPE_val(k) = mean((abs(yVal - yValTrue))./yValTrue);
%         MAPE_test(k) = mean((abs(yTest - yTestTrue))./yTestTrue);
        MAE_train(k) = (sum(abs(yTrain - yTrainTrue)))./size(yTrain,2);%controlla
        MAE_val(k) = (sum(abs(yVal - yValTrue)))./size(yVal,2);%controlla
%         MAE_test(k) = (sum(abs(yTest - yTestTrue)))./size(yTest,2);%controlla
        performance = [performance; j k RMSE_train(k) RMSE_val(k) MSE_train(k) MSE_val(k) MAPE_train(k) MAPE_val(k) ...
            MAE_train(k) MAE_val(k)];
%         testpf = [testpf; j k RMSE_test(k) MAPE_test(k)];
    end
end
```

Select the optimal

plot(1:k, RMSE_train); hold on; plot(1:k, RMSE_val); %plot(1:k, RMSE_test); hold off;

plot(yTrainTrue, yTrain, 'x'); hold on; plot(1:8000,1:8000); Values above 5000 are the most problematic ones

```
% Do the same for validation and test

% Compute R^2 for training set

%SSE = sum((yTrainTrue - yTrain).^2);
%med = mean(yTrainTrue);
%TSS = sum((yTrainTrue - med).^2);
%R2 = 1- (SSE/TSS);

% Do the same for validation and test
```

References

Aboukarima, A., Elsoury, H., and Menyawi, M., 2015. *Artificial neural network model for the prediction of the cotton crop leaf area*. International Journal of Plant and Soil Science, Vol. 8, pp. 1-13.

Ait-Sahalia, Y., and Lo, A. W., 1998. *Nonparametric estimation of state-price densities implicit in financial asset prices*. The Journal of Finance, Vol. 53, pp. 499-547.

Ait-Sahalia, Y., and Lo, A. W., 2000. *Nonparametric risk management and implied risk aversion. Journal of Econometrics*, Vol. 94, pp. 9-51.

Alpaydin, E., 2020. *Introduction to machine learning*, fourth edition. The MIT Press.

Ambrož, L., 2002. *Oceˇnovanie opcií*. C.H. Beck. ISBN: 80-7179-531-3.

Amilon, H., 2003. *A neural network versus Black-Scholes: a comparison of pricing and hedging performances*. Journal of Forecasting, Vol. 22, pp. 317-335.

Anders, U., Korn, O., and Schmitt, C., 1998. *Improving the pricing of options: a neural network approach*. Journal of Forecasting, Vol. 17, pp. 369-388.

Andersen, L., and Brotherton-Ratcliffe, R., 1997. *The equity option volatility smile: an implicit finite-difference approach*. Journal of Computational Finance, Vol. 1, pp. 5-38.

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

Andreou, P. C., Charalambous, C., and Martzoukos, S. H., 2010. *Generalized parameter functions for option pricing*. Journal of Banking and Finance, Vol. 34, pp. 633–646.

Anwar, M. N., and Andallah, L. S., 2018. *A study on numerical solution of Black-Scholes model*. Journal of Mathematical Finance, Vol. 8, 372-381.

Arif, J., Chaudhuri, N. R., ray, S., and Chaudhuri, B., 2009. *Online Levenberg-Marquardt algorithm for neural network based estimation and control of power systems*. International Joint Conference on Neural Networks, pp. 199-206.

Ascioglu, A., Holowczak, R., Louton, D., and Saraoglu, H., 2017. *The evolution of market share among the U.S. options market platforms*. The Quarterly Review of Economics and Finance, Vol. 64, pp. 196-214.

Backus, D., Foresi, S., Li, K., and Wu, L., 1997. *Accounting for biases in Black-Scholes*. Fordham University.

Bahra, B., 1997. *Implied risk-neutral probability density functions from option prices: theory and application*. Bank of England working paper.

Bakshi, G., Cao, C., and Chen, Z., 1997. *Empirical performance of alternative option pricing models*. The Journal of Finance, Vol. 52, pp. 2003-2049.

Banko, M., and Brill, E., 2001. *Scaling to very very large corpora for natural language disambiguation.* Proceedings of the 39th annual meeting of the Association for Computational Linguistic, pp. 26-33.

Barr, J. K., 2009. *The implied volatility bias and option smile: is there a simple explanation?.* Iowa State University.

Bates, D. S., 1995. *Testing option pricing models*. In chapter 20 of G. S. Maddala and C. R. Rao, editors, Statistical Methods in Finance, Handbook of Statistics, Vol. 14, pp. 567-611.

Bates, D. S., 1996. *Jumps and stochastic volatility: exchange rate process implicit in deutsche mark options*. The Review of Financial Studies, Vol. 9, pp. 69-107.

Bates, D. S., 2008. *The market for crash risk.* Journal of Economic Dynamics and Control, Vol. 32, pp. 2291-2321.

Bellman, R., 1957. *Dynamic programming*. Dover Publications Inc.

Bennell, J. A., and Sutcliffe, C., 2004. *Black-Scholes versus neural networks in pricing FTSE 100 options*. intelligent Systems in Accounting, Finance and Management, Vol. 12, pp. 00-156.

Bergstra, J., and Bengio, Y., 2012. *Random search for hyperparameter optimization*. Journal of Machine Learning Research, Vol. 13, pp. 281-305.

Berry, M. J. A., and Linoff, G. S., 1997. *Data mining techniques*. Wiley and Sons.

Billio, M., Corazza, M., and Gobbo, M., 2002. *Option pricing via regime switching models and multilayer perceptrons: a comparative approach*. Rendiconti per gli Studi Economici Quantitativi, Vol. 2002, pp. 39–59.

Bishop, C. M., 1995. *Neural networks for pattern recognition*. Oxford University Press.

Black, F., and Scholes, M., 1973. *The pricing of options and corporate liabilities*. The Journal of Political Economy, Vol. 81, pp. 637-654.

Bloch, D. A., 2019. *Option pricing with machine learning*. Universitè Paris VI et Marie Curie.

Block, H. D., 1970. *A review of "Perceptrons: an introduction to computational geometry".* Information and Control, Vol. 17, pp. 501-522.

Blum, A. L., and Rivest, R., 1992. *Training a 3-node neural network is NP-Complete*. Neural networks, Vol. 5, pp. 117-127.

Blynski, L., and Faseruk, A., 2006. *Comparison of the effectiveness of option price forecasting: Black–Scholes vs simple and hybrid neural networks*. Journal of Financial Management and Analysis, Vol. 19, pp. 46–58.

Bodurtha, J. N., and Courtadon, G. R., 1987. *The pricing of foreign currency options*. New York: Salomon Brothers Center for the Study of Financial Inst.

Bodurtha, J. N., and Jermakyan, M., 1996. *Nonparametric estimation of an implied volatility surface.* Georgetown University.

Boger, Z., and Guterman, H., 1997. *Knowledge extraction from artificial neural network models*. IEEE International Conference on Systems, Man and Cybernetics, Vol. 4, pp. 3030-3035.

Bossu, S., and Henrotte, P., 2012. *The Black-Scholes model*. Wiley and Sons.

Bouchaud, J. P., and Potters, M., 2000. *Theory of financial risk and derivative pricing. From statistical physics to risk management.* Cambridge University.

Brenner, M., and Eom, Y. H., 1997. *No-arbitrage option pricing: new evidence on the validity of the martingale property*. New York University.

Breunig, M. M., Kriegel, H. P., Ng, R. T., and Sander, J., 1999. *OPTICS-OF: identifying local outliers*. Proceedings of the third European Conference on Principles and Practice of Knowledge Discovery in Databases, Prague.

Broadie, M., and Detemple, J. B., 2004. *Option pricing: valuation models and applications*. Management Science, Vol. 50, pp. 1145-1177.

Brown, S. J., and Dybvig, P. H., 1986. *The empirical implications of the Cox, Ingersoll, Ross theory of the term structure of interest rates.* The Journal of Finance, Vol. 41, pp. 617-630.

Brownlee, J., 2018. *Use early stopping to halt the training of neural networks at the right time*. Published in Deep Learning Performance.

Buehler, H., Gonon, L., Teichmann, J., and Wood, B. 2019. *Deep hedging*. Quantitative Finance, Vol. 19, pp. 1271–1291.

Campa, J. M., and Chang, P. H. K., 1995. *Testing the expectations hypothesis on the term structure of volatilities in foreign exchange options*. Journal of Finance, Vol. 50, pp. 529-547.

Campa, J. M., and Chang, P. H. K., 1996. *Arbitrage-based tests of target-zone credibility: evidence from ERM cross-rate options*. The American Economic Review, Vol. 86, pp. 726-740.

Campa, J. M., Chang, P. H. K., and Reider, R. L., 1998. *Implied exchange rate distributions: evidence from the OTC option markets*. Journal of International Money and Finance, Vol. 17, pp. 117-160.

Cao, J., Chen, J., and Hull, J. C., 2019. *A neural network approach to understanding implied volatility movements*. SSRN 3288067, 2019

Carbonell, J. G., Michalski, R. S., and Mitchell, T. M., 1983. *Machine learning: an artificial intelligence approach*. Morgan Kaufmann.

Carr, P. P., and Madan, D., 1998. *Determining volatility surfaces and option values from an implied volatility smile*. University of Maryland.

Carsten, J. J., 1999. *Option implied risk-neutral distributions and implied binomial trees: a literature review*. Journal of Derivatives, Vol. 7, pp. 66-82.

Carverhill, A. P., and Cheuk, T. H., 2003. *Alternative neural network approach for option pricing and hedging*. SSRN 480562, 2003.

Cawley, G. C., and Talbot, N. L. C., 2010. *On overfitting in model selection and subsequent selection bias in performance evaluation.* Journal of Machine Learning Research, Vol. 11, pp. 2079-2107.

Černy, M., 2008. *On estimation of volatility of financial time series for pricing derivatives*. Prague University of Economics and Business.

Chappell, D, 1992. *On the derivation and solution of the Black-Scholes option pricing model: a step-by-step guide*. Sheffield University Management School.

Chen, F., and Sutcliffe, C., 2011. *Pricing and hedging short sterling options using neural networks*. Henley University of Reading, ICMA Centre.

Cho, H., Kim, Y., Lee, E., Choi, D., Lee, Y., and Rhee, W., 2020. *Basic enhancement strategies when using Bayesian optimization for hyperparameter tuning of deep neural networks*. IEE Access, Vol. 8, pp. 52588-52608.

Chollet, F., 2017. *Deep learning with Python*. Manning Pubns.

Christie, A. A., 1982. *The stochastic behaviour of common stock variances: value, leverage and interest rate effects.* Journal of Financial Economics, Vol. 10, pp. 407-432.

Clark, P. B., 1973. *Uncertainty, exchange risk, and the level of international trade*. Economic Inquiry, Vol. 11, pp. 302-313.

Clark, P. K., 1973. *A subordinated stochastic process model with finite variance for speculative process*. Econometrica, Vol. 41, pp. 135-155.

Colusso, P., 2019. *A machine learning approach to parametric option pricing*. M. Sc. Thesis.

Commarata, M., 2016. *Corporate diversity in corporate governance and its effect on financial performance: a study on FTSE MIB listed companies*. Universidade NOVA, M. Sc. Thesis.

Constantinides, G. M., Jackwerth, J. C., and Perrakis, S., 2009. *Mispricing of S&P500 index options. the Review of Financial Studies*, Vol. 22, pp. 1247-1277.

Contreras, M., Llanquihuen, A., and Villena, M., 2016. On the solution of the multi asset Black-Scholes model: correlations, eigenvalues and geometry. Journal of Mathematical Finance, Vol. 6, Art. No. 4.

Corrado, C. J., and Su, T., 1996. *Skewness and kurtosis in S&P500 index returns implied by option prices*. The Journal of Financial Research, Vol. 19, pp. 175-192.

Cox, J. C., and Ross, S. A., 1976. *The valuation of options for alternative stochastic processes.* Journal of Financial Economics, Vol. 3, pp. 145-166.

Cox, J. C., Ross, S. A., and Rubinstein, M., 1979. *Option pricing: a simplified approach*. Journal of Financial Econometrics, Vol. 7, pp. 229-263.

Culkin, R., and Das, S. R., 2017. *Machine learning in finance: the case of deep learning for option pricing*. Santa Clara University.

Cybenko, G., 1989. *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals and Systems, Vol. 2, pp. 303-314.

Daburra, I., 2018. *Coding neural network – feedforward propagation and backpropagation*. Published by Towards Data Science.

Dar, A. A., and Anuradha, N., 2018. *Comparison: binomial model and Black-Scholes model*. Quantitative Finance and Economics, Vol. 2, pp. 230-245.

Das, S. P., and Padhy, S., 2017. *A new hybrid parametric and machine learning model with homogeneity hint for European-style index option pricing*. Neural Computing and Applications, Vol. 28, pp. 4061-4077.

Das, S. R., and Sundaram, R. K., 1999. *Of smiles and smirks: a term structure perspective*. The Journal of Financial and Quantitative Analysis, Vol. 34, pp. 211-239.

Dayhoff, J. E., 1990. *Neural network architectures: an introduction*. Published by Van Nostrand Reinhold Company.

Dey, A., 2016. *Machine learning algorithms: a review*. International Journal of Computer Science and Information Technologies, Vol. 7, pp. 1174-1179.

Dongare, A. D., Kharde, R. R., and Kachare, A. D., 2012. *Introduction to artificial neural network*. International Journal of Engineering and Innovative Technology, Vol. 2.

Dreyfus, S. E., 1957. *Computational aspects of Dynamic Programming*. Operations Research, Vol. 5, pp. 409-415.

Dreyfus, S. E., 1973. *The computational solution of optimal control problems with time lag*. Transactions on Automatic Control, Vol. 18, pp. 383-385.

Dreyfus, S. E., 1990. *Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure*. University of California, Berkeley.

Duarte, J. C., and Echeverria, F. R., 2002. *Time series forecasting using ARIMA, neural networks and neo fuzzy neurons*. Universidad de Los Andes.

Dufour, J. M., and Neves, J., 2019. *Conceptual econometrics using R. Grid search*. Handbook of statistics, Vol. 41, pp. 2-314.

Dugas, C., Bengio, Y., B́elisle, F., Nadeau, C., and Garcia, R. 2009. *Incorporating functional knowledge in neural networks*. Journal of Machine Learning Research, Vol. 10, pp.1239–1262.

Fradkov, A. L., 2020. *Early history of machine learning*. IFAC – Papers Online, Vol. 53, pp. 1385-1390.

Garcia, R., and Gencay, R., 1998. *Option pricing with neural networks and a homogeneity hint*. Decision Technologies for Computational Finance, pp. 195–205.

Gayathri, T., and Bhaskari, L., 2016. *A comprehensive review of subspace clustering in the analysis of big data.* International Journal of Engineering Trends and Technology, Vol. 39, pp. 135-142.

Gencay, R., and Salih, A., 2003. *Degree of mispricing with the Black-Scholes model and nonparametric cures*. Economics and Finance, Vol. 4, pp. 73-101.

Géron, A., 2019. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

Ghaziri, H., Elfakhani, S., and Assi, J., 2000. *Neural networks approach to pricing options*. Neural Network World, Vol. 10, pp. 271–277.

Ghysels, E., Patilea, V., Renault, E., and Torres, O., 1997. *Nonparametric methods and option pricing*. CIRANO Working Paper.

Gibbons, M. R., and Ramaswamy, K., 1993. *A test of the Cox, Ingersoll, and Ross model of the term structure*. The Review of Financial Studies, Vol. 6, pp. 619-658.

Gikhman, I. I., 2006. *Drawbacks in options definition*. SSRN.

Girosi, F., Jones, M. J., and Poggio, T. A., 1993. *Priors, stabilizers and basis functions: from regularization to radial, tensor and additive splines*. MIT Press.

Gradojevic, N., Gencay, R., and Kukolj, D., 2009. *Option pricing with modular neural networks. Transactions on Neural Networks*, Vol. 20, pp. 626-637.

Grossman, S. J., and Zhou, Z., 1996. *Equilibrium analysis of portfolio insurance*. The Journal of Finance, Vol. 51, pp. 1379-1403.

Gu, S., Kelly, B., and Xiu, D., 2020. *Empirical asset pricing via Machine Learning*. The Review of Financial Studies, Vol. 33, pp. 2223-2273.

Gupta, M., 2017. *Cross-validation in machine learning*. Published by Towards Data Science.

Gupta, M., 2022. *A comparative study on supervised machine learning algorithm*. Published by Towards Data Science.

Hahn, T., 2014. *Option pricing using artificial neural networks: an Australian perspective*. PhD thesis. Bond University.

Hakan, O. Y., 2005. *Criticism of the Black-Scholes model: but why is it still used? (The answer is simpler than the formula).* MPRA, paper No. 63208, New York University.

Hamid, S. A., and Habib, A., 2005. *Can neural networks learn the Black-Scholes model? A simplified approach.* Southern New Hampshire University.

Hammoudeh, S., and McAleer, M., 2013. *Risk management and financial derivatives. An overview.* The North American Journal of Economics and Finance, Vol. 25, pp. 109-115.

Hanczar, B., Zehraoui, F., Issa, T., and Arles, M., 2020. *Biological interpretation of deep neural network for phenotype prediction based on gene expression*. BMC Informatis, Vol. 21, Art. No. 501.

Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The elements of statistical learning. Data mining, inference, and prediction*. Springer Series in Statistics. Springer, Second Edition.

Haug, E. G., 2007. *Derivatives: models on models*. Wiley and Sons.

Haug, E. G., 2007. *The complete guide to option pricing formulas*. McGraw Hill, second edition.

Haug, E. G., and Taleb, N. N., 2009. *Option traders use (very) sophisticated heuristics, never the Black-Scholes-Merton formula*. Journal of Economic Behaviour and Organization, pp. 97-106.

Hawkins, D. M., 1980. *Identification of outliers*. Part of the series Monographs on Statistics and Applied Probability.

Healy, J., Dixon, M., Read, B., and Cai, F., 2002. *A data-centric approach to understanding the pricing of financial options.* The European Physical Journal, Vol. 27, pp. 219–227.

Hebb, D. O., 1949. *The organization of behaviour; a neuropsychological theory*. Wiley and Sons.

Helm, J. M., Swiergosz, A. M., Haeberle, H. S., Karnuta, J. M., Schaffer, J. L., Krebs, V. E., Spitzer, A. I., and Ramkumar, P. N., 2020. *Machine learning and artificial intelligence: definitions, applications, and future directions*. Current Reviews in Musculoskeletal Medicine, Vol. 13, pp. 69-76.

Hentschel, L., 2009. *Errors in implied volatility estimation*. Journal of Financial and Quantitative Analysis, Vol. 38, pp. 779-810.

Heskes, T., and Wiegerinck, W., 1996*. A theoretical comparison of batch-mode, on-line, cyclic, and almost-cyclic learning.* IEEE Transactions on Neural Networks, Vol. 7, pp. 919-925.

Hoffman, K., 2021. *Machine learning: how to prevent overfitting*. Published in The Startup.

Hornik, K., 1991. *Approximation capabilities of multilayer feedforward networks*. Neural Networks, Vol. 4, pp. 251-257.

Hughes, G., 1968. *On the mean accuracy of statistical pattern recognizers*. Transactions on Information Theory, Vol. 14, pp. 55-63.

Hull, J. C., 2008. *Options, Futures, and other Derivatives*. 7th ed. Upper Saddle River, New Jersey: Pearson Education.

Hull, J. C., 2021. *Machine learning in business: an introduction to the world of data science*. Amazon Fulfillment Poland Sp. Zoo.

Hutchinson, J. M., Lo, A. W., and Poggio, T., 1994. *A nonparametric approach to pricing and hedging derivatives securities via learning networks*. The Journal of Finance, Vol. 49, pp. 851-889.

Ingersoll, J. E., Cox, J. C., and Ross, S. A. J., 1985. *A theory of the term structure of interest rates*. Econometrica, Vol. 53, pp. 385-407.

Ismiguzel, I., 2021. *Hyperparameter tuning with grid search and random search and a deep dive into how to combine them*. Published by Towards Data Science.

Ivascu, C. F., 2021. *Option pricing using machine learning*. Bucharest University of Economic Studies.

Jackwerth, J. C., and Rubinstein, M., 1996. *Recovering probability distributions from option prices.* The Journal of Finance, Vol. 51, pp. 1611-1631.

Jankova, Z., 2018. *Black-Scholes model differential equation and its modifications for valuation of financial derivatives*. Brno University of Technology.

Jankova, Z., 2018. *Drawbacks and limitations of Black-Scholes model for option pricing*. Journal of Financial Studies and Research, Vol. 2018, pp. 1-7.

Janocha, K., and Czarnecki, W. M., 2017. *On loss functions for deep neural networks in classification*. Cornell University.

Jarrow, R., 2013. *Option pricing and market efficiency*. The Journal of Portfolio Management, Vol. 40, pp. 88-94.

Karlik, B., and Olgac, A. V., 2011. *Performance analysis of various activation functions in generalized MLP architectures of neural networks*. International Journal of Artificial Intelligence and Expert Systems, Vol. 1, pp. 111-122.

Karsoliya, S., 2012. *Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture*. International Journal of Engineering Trends and Technology, Vol. 3, pp. 714-717.

Kelley, H. J., 1960. *Gradient theory of optimal flight paths*. ARS Journal, Vol. 30, pp. 947-954.

Kleene, S. C., 1956. *Representation of events in nerve nets and finite automa*. The Annals of Mathematics Studies, Vol. 34, pp. 3-41.

Kotsiantis, S. B., Zaharakis, I. D., and Pintelas, P. E., 2007. *Supervised machine learning: a review of classification techniques*. Published in the Emerging Artificial Intelligence Applications in Computer Engineering conference.

Lajbcygier, P. R., and Connor, J. T., 1997. *Improved option pricing using artificial neural networks and bootstrap methods*. International Journal of Neural Systems, Vol. 8, pp. 457-471.

Larsson, J., 2020. *Optimization of option pricing. Variance reduction and low-discrepancy techniques*. Umea School of Business, Economics and Statistics, Bachelor Thesis.

Lauterbach, B., and Schultz, P., 1990. *Pricing warrants: an empirical study of the Black-Scholes model and its alternatives*. Journal of Finance, Vol. 45, pp. 1181-1209.

Lawrence, S., Giles, C. L., and Tsoi, A. C., 1998. *What size neural network gives optimal generalization? Convergence properties of backpropagation*. University of Maryland.

LeCun, Y., Bottou, L., Orr, G. B., and Muller, K. R., 1998. *Efficient backprop*. New York University.

Liashchynskyi, P., and Liashchynskyi, P., 2019. *Grid search, random search, genetic algorithm: a big comparison for NAS*. Cornell University.

Liu, S., Oosterlee, C. W., and Bohte, S. M., 2019. *Pricing options and computing implied volatilities using neural networks*. Risks, Vol. 7, pp. 1–22.

Longstaff, F. A., 1990. *Pricing options with extendible maturities: analysis and applications*. The Journal of Finance, Vol. 45, pp. 935-957.

Longstaff, F. A., 1995. *Option pricing and the martingale restriction*. Review of Financial Studies, Vol. 8, pp. 1091-1124.

Lurent, J. P., and Leisen, D. P. J., 1998. *Building a consistent pricing model from observed option prices.* Stanford University.

Mahesh, B., 2019. *Machine learning algorithms – a review*. International Journal of Science and Research, Vol. 9, pp. 381-386.

Malliaris, M., and Salchenberger, L., 1993. *A neural network model for estimating option prices*. Applied Intelligence, Vol. 3, pp. 193-206.

Mas, J. F., and Flores, J. J., 2008. *The application of artificial neural networks to the analysis of remotely sensed data.* International Hournal of Remote Sensing, Vol. 29, pp. 617-663.

McCulloch, W. S., and Pitts, W. H., 1943. *A logical calculus of ideas immanent in nervous activity*. Bulletin of Mathematical Biology, Vol. 5, pp. 115-133.

McKean, H. P. J., 1969. *Stochastic integrals*. Academic Press, New York.

Merton, R. C., 1971. *Optimum consumption and portfolio rules in a continuous-time model*. Journal of Economic Theory, Vol. 3, pp. 373-413.

Merton, R. C., 1973. *Theory of rational option pricing*. The Bell Journal of Economics and Management Science, Vol. 4, pp. 141-183.

Merton, R. C., 1974. *On the pricing of corporate debt: the risk structure of interest rates*. The American Finance Association Meeting 1974.

Merton, R. C., 1976. *Option pricing when underlying stock returns are discontinuous*. Journal of Financial Economics, Vol. 3, pp. 125-144.

Merton, R. C., 1976. *The impact on option pricing of specification error in the underlying stock price returns*. The Journal of Finance, Vol. 31, pp. 333-350.

Merton, R. C., 1991. *Continuous-time finance*. John Wiley and Sons.

Minsky, M. and Papert, S., 1969. *An introduction to computational geometry*. Cambridge University.

Minsky, M. and Papert, S., 1972. *Research at the Laboratory in Vision, Language, and Other Problems of Intelligence: Progress Report*. Massachusetts Institute of Technology.

Minsky, M., 1960. *Steps toward artificial intelligence*. Cambridge University.

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

Mostafa, F., and Dillon, T., 2008. *A neural network approach to option pricing*. WIT Transactions on Information and Communication Technologies, Vol. 41, pp. 71-85.

Nair, A., 2022. *Grid search vs random search vs Bayesian optimization*. Published by Towards data Science.

Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., Band, S. S., Reuter, U., Gama, J., and Gandomi, A. H., 2020. *Data science in economics: comprehensive review of advanced machine learning and deep learning methods*. Journal of Mathematics, Vol. 8, issue 10.

Osborne, M. F. M., 1959. *Brownian motion in the stock market*. Operations Research, Vol. 7, pp. 145-173.

Phani, B. V., Chandra, B., and Raghav, V., 2011. *Quest for efficient option pricing prediction model using machine learning techniques*. International Joint Conference of Neural Networks, 2011, pp. 654-657.

Pires, M. M., and Marwala, T., 2005. *American option pricing using Bayesian multilayer perceptrons and Bayesian support vector machines*. International Conference on Computational Cybernetics, 2005, pp. 219-224.

Pogorui, A., and Rodriguez-Dagnino, R. M., 2009. *Evolution process as an alternative to diffusion process and Black-Scholes formula*. Random Operators and Stochastic Equations, Vol. 17, pp. 61-68.

Pohlodek, J., Morabito, B., Zometa, P., and Findeisen, R., 2022. *Flexible development and evaluation of machine learning supported optimal control and estimation methods via HILO-MPC*. Published by DeepAI.

Rachev, S. T., Menn, D. C., and Fabozzi, F. J., 2008. *Black-Scholes option pricing model*. Wiley and Sons.

Rao, S. V. A., Kondaiah, K., Chandra, G. R., and Kumar, K. K., 2017. *A survey on machine learning: concept, algorithms and applications*. Published in International Conference on Innovative Research in Computer and Communication Engineering.

Reed, R., 1999. *Neural smithing: supervised learning feedforward artificial neural networks*. MIT Press.

Reshma, P., Prasanth, P., and Udayanandan, K. M., 2021. *The curse of dimensionality in physics*. Progress in Physics, Vol. 17.

Riachy, W., 2019. *Comparing Black-Scholes and binomial models for pricing European options*. Usek School of Business.

Robbins, H., and Monro, S., 1951. *A stochastic approximation method*. The Annals of Mathematical Statistics, Vol. 22, pp. 400-407.

Rosenberg, J. V., 1998. *Pricing multivariate contingent claims using estimated risk-neutral density functions*. Journal of International Money and Finance, Vol. 17, pp. 229-247.

Rosenberg, J. V., and Engle, R. F., 1997. *Option hedging using empirical pricing kernels*. New York University.

Rosenblatt, F., 1958. *The perceptron: a probabilistic model for information storage and organization in the brain*. Psychological Revie, Vol. 65, pp. 386-408.

Ross, S. A., 1976. *Options and efficiency*. The Quarterly Journal of Economics, Vol. 90, pp. 75-89.

Rubinstein, M., 1985. *Nonparametric tests of alternative option pricing models using all reported trades and quotes on the 30 most active CBOE option classes from August 23, 1976 through August 31, 1978*. The Journal of Finance, Vol. 40, pp. 455-480.

Rubinstein, M., 1994. *Implied binomial trees*. The Journal of Finance, Vol. 49, pp. 771-818.

Ruf, J, and Wang, W., 2020. *Neural networks for option pricing and hedging: a literature review*. Journal of Computational Finance, Vol. 24, pp. 1-46.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986. *Learning representations by backpropagating errors*. Nature, Vol. 323, pp. 533-536.

Samuel, A. L., 1959. *Some studies in machine learning using the game of checkers*. IBM Journal of Research and Development, Vol. 3, pp. 210-229

Samur, Z. I., and Temur, G. T., 2009. *The use of artificial neural network in option pricing: the case of S&P 100 index options*. International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering, Vol. 3, pp. 644–649.

Sasakawa, T., Hu, J., and Hirasawa, H., 2007. *A brain like learning system with supervised, unsupervised, and reinforcement learning*. Wiley and Sons.

Scholes, M., 1976. *Taxes and the pricing of options*. The Journal of Finance, Vol. 31, pp. 319-332.

Shah, T., 2017. *About train, validation and test sets in machine learning*. Published in Towards Data Science.

Sharma, S., and Sharma, S., 2020. *Activation functions in neural networks*. International Journal of Engineering Applied Sciences and Technology, Vol. 4, pp. 2455-2143.

Shinde, A. S., and Takale, K. C., 2012. *Study of Black-Scholes model and its applications*. Procedia Engineering, Vol. 38, pp. 270-279.

Singh, A., Thakur, N., and Sharma, A., 2016. *A review of supervised machine learning algorithms*. Published in the International Conference on Computing for Sustainable Global Development.

Smith, C. W. Jr, 1976. *Option pricing: a review*. Journal of Financial Economics, Vol. 3, pp. 3-51.

Smiti, A., 2020. *A critical overview of outlier detection methods*. Computer Science Review, Vol. 38.

Spiegeleer, J. D., Madan, D. B., Reyners, S., and Schoutens, W., 2018. *Machine learning for quantitative finance: fast derivative pricing, hedging and fitting*. Quantitative Finance, Vol. 18, pp. 1635-1643.

Srivastava, T., 2015. *Introduction to online machine learning: simplified*. Published by Analytics Vidhya.

Stathakis, D., 2009. *How many hidden layers and nodes?.* International Journal of Remote Sensing, Vol. 30, pp. 2133-2147.

Stewart, M., 2019. *Simple guide to hyperparameter tuning in Neural Networks*. Published by Towards Data Science (2019).

Stutzer, M., 1996. *A simple nonparametric approach to derivative security valuation.* Journal of Finance, Vol. 51, pp. 1633-1652.

Su, X., and Tsai, C. L., 2011. *Outlier detection*. Wires Data Mining and Knowledge Discovery, Vol. 1, pp. 261-268.

Sutton, F. E., Beyl, R., Early, K. S., Cefalu, W. T., Ravussin, E., and Peterson, C. M., 2018. *Early time restricted feeding improves insulin sensitivity, blood pressure, and oxidative stress even without weight loss in men with prediabetes*. Cell Metab, Vol. 27, pp. 1212-1221.

Telgarsky, M., 2016. *Benefits of depth in neural networks*. JMLR: workshop and Conference Proceedings, Vol. 49, pp. 1-23.

Teneng, D., 2011. *Limitations of the Black-Scholes model*. International Research Journal of Finance and Economics, pp. 99-102.

Twomey, J. M., and Smith, A. E., 1995. *Performance measures, consistency, and power for artificial neural network models.* Mathematical and Computer Modeling, Vol. 21, pp. 243-258.

Vanstone, B. J., Finnie, G., and Hahn, T., 2010. *Stock market trading using fundamental variables and neural networks*. Bond Business School.

Von Neumann, J., 1956. *Probabilistic logics and synthesis of reliable organisms from unreliable components*. The Annals of Mathematics Studies, Vol. 34, pp. 43.98.

Wahba, G., 1990. *Spline models for observational data*. Society for Industrial and Applied Mathematics.

Wang, M. X., and Qu, Y., 2021. *Approximation capabilities of neural networks on unbounded domains*. Harvard University.

Wang, S. C., 2003. *Artificial neural network*. Interdisciplinary Computing in Java Programming, pp. 81-100.

Weigand, A., 2019. *Machine learning in empirical asset pricing*. Financial Markets and Portfolio Management, Vol. 33, pp. 93-104.

Werbos, P., 1975. *Backpropagation through time: what it does and how to do it*. Proceeding of the IEEE, Vol. 78, pp. 1550-1560.

Widrow, B., and Stearns, S. D., 1985. *Adaptive signal processing*. Alan V. Oppenheim Editor.

Wilamowski, B. M., 2009. *Neural network architectures and learning algorithms*. Industrial Electronics Magazine, Vol. 3, pp. 56-63.

Wistuba, M., Rawat, A., and Pedapati, T., 2019. *A survey on neural architecture search*. Cornell University.

Xu, S., and Cheng, L., 2008. *A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining*. International Conference on Information Technology and Applications, 2008.

Xu, Y., and Goodacre, R., 2018. *On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning.* Journal of Analysis and Testing, Vol. 2, pp. 249-262.

Yalincak, O. H., 2019. *Criticism of the Black-Scholes model: but why is it still used? The answer is simpler than the formula*. New York University.

Yang, Y., Zheng, Y., and Hospedales, T., 2017. *Gated neural networks for option pricing: rationality by design*. Conference of Artificial Intelligence.

Yao, J., Li, Y., and Tan, C. L., 2000. *Option price forecasting using neural networks*. Omega, Vol. 28, pp. 455-466.

Yao, Y., Rosasco, L., and Caponetto, A., 2007. *On early stopping in gradient descent learning*. Constructive Approximation, Vol. 26, pp. 289-315.

Yathish, V., 2022. *Loss functions and their use in neural networks*. Published by Towards Data Science (2022).

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2016. *Understanding deep learning (still) requires rethinking generalization.* Communications of the ACM. Vol. 64, pp. 107-115.

Zhang, D., Lyle, M., and Nault, B. R., 2020. *Trading volume and open interest from options markets as measures of the effect of IT announcements*. Haskayne School of Business.

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

Zhang, G. P., and Qi, M., 2005. *Neural network forecasting for seasonal and trend time series*. European Journal of Operational research, Vol. 160, pp. 501-514.

Zhang, T, and Yu, B., 2005. *Boosting with early stopping: convergence and consistency*. The Annals of Statistics, Vol. 33, pp. 1538-1579.

Zheng, M., Tang, W., and Zhao, X., 2018. *Hyperparameter optimization of neural network driven spatial models accelerated using cyber-enabled high-performance computing*. International Journal of Geographical Information Science, Vol. 33, pp. 1-32.

Zhou, Y., 2022. *Option trading volume by moneyness, firm fundamentals, and expected stock returns.* Journal of Financial Markets, Vol. 58, 100648.

Sitography

Analytics Vidhya. Available at: https://www.analyticsvidhya.com/

Banca d'Italia. Available at: https://www.bancaditalia.it/

Banco BPM: IDEM. Available at: https://www.bancobpm.it/magazine/glossario/idem/

Bloomberg. Available at: https://www.bloomberg.com/

Board of Governors of the Federal reserve System. Available at: https://www.federalreserve.gov/

Borsa Italiana. Available at: https://www.borsaitaliana.it/homepage/homepage.htm

Cambridge digital library. Available at: https://cudl.lib.cam.ac.uk/

CBOE Global Markets. Available at: https://www.cboe.com/

Commodity Futures Trading Commission. Available at: https://www.cftc.gov/

Constellate. Available at: https://constellate.org/

Corporate Finance Institute: Put-Call parity. Available at: https://corporatefinanceinstitute.com/resources/knowledge/finance/put-call-parity/

DeepAI. Available at: https://deepai.org/

Fortmann-Roe: Understanding the Bias-Variance tradeoff. Available at: https://scott.fortmann-roe.com/docs/BiasVariance.html

FTSE Russell. Available at: https://www.ftserussell.com/

GitHub. Available at: https://github.com/

IBM Cloud Education: Gradient Descent. Available at: https://www.ibm.com/cloud/learn/gradient-descent

InteractiveBrokers. Available at: https://www.interactivebrokers.ie/en/

Investing. Available at: https://it.investing.com/

Investopedia. Available at: https://www.investopedia.com/

Matlab. Available at: https://it.mathworks.com/

National institute of Standards and Technology. Available at: https://www.nist.gov/

Option Metrics: https://optionmetrics.com/

SeekingAlpha. Available at: https://seekingalpha.com/

Sole24Ore. Available at: https://www.ilsole24ore.com/

StackExchange. Available at: https://stackexchange.com/

Statista. Available at: https://www.statista.com/

Supervised Machine Learning: regression and classification. By Ng, A., Coursera. Available at: https://www.coursera.org/learn/machine-learning

The OCC. Available at: https://www.theocc.com/Market-Data/Market-Data-Reports/Series-and-Trading-Data/Directory-of-Listed-Products

The StartUp. Available at: https://medium.com/swlh

Towards Data Science. Available at: https://towardsdatascience.com/

U.S. Security and Exchange Commission. Available at: https://www.sec.gov/

Wiley online library. Available at: https://onlinelibrary.wiley.com/