POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Matematica

Tesi di Laurea Magistrale

Exploiting deep learning techniques for stock price prediction

Relatore
Tania Cerquitelli

Candidato
Laura Marioni

A.A. 2020-2021

Contents

Abstract

Over the last decade, the analysis of financial time series has undergone an important development, both in terms of basic research and direct market applications. In particular, the computerization of data has made detailed information on price trends and traded volumes easily available, thus creating new fields of investigation. At the same time, the introduction of electronic trading systems has made large financial institutions interested in automating trading processes through algorithms.
In this thesis, the problem of predicting closing stock prices over multivariate time series using Deep Learning models was addressed.
Due to the volatility and mutability of the time series, it is necessary to introduce models that allow predicting future values based on past ones, with the objective that the investor faces a lower risk of loss.

The use case consists of an analysis of data coming from the site Yahoo! Finance related to the stock index ISP.MI.
In the beginning, the theory on which the neural networks are based will be deepened, the mathematics behind it will be explained, and how they can adapt to the characteristics of the problem to be faced.
We will then move on to the collection and preparation of the necessary data, and the implementation of the different Networks in Python code, exploiting an open-source library called Keras, the most widely used to use Deep Learning models.
Starting from a database of 4 years for training several models, we want to predict the closing values of the days immediately following the input days. In particular, four models have been analyzed which, although based on the same principle of pattern detection through data encoding and decoding, present different properties.
Finally, the performance of the models will be evaluated and, on the basis of these, conclusions will be drawn on the actual effectiveness of the model, its limitations and possible future developments.
The objective of this thesis is to find a model that performs well in the approximation and prediction of time series in the financial domain.

Chapter 1

Introduction

I started my internship at Accenture S.p.A on 3 May, embarking on a journey that put me in close contact with data and data control. During these months, I have been working with the team on a project in the banking sector: the aim is to build a Budget at branch level for the year 2021, the basis of which is the Forecast. The thesis is the result of my interest in the subject during my working hours, and how the studies I did during my master's degree could be applied to this branch.
The work during these months of the internship was done for one of the largest banks that dominate the Italian banking sector.
The banking system since 2007 has begun to go through a phase of downsizing and digitization to reduce costs. This has been necessary to counteract the financial crisis and the consequent increase in unemployment and decrease in profitability, leading banks to thin margins.
Technological advancement has been one of the major changes that have taken place, both within the banking system, i.e. on their operation, and in the process logics that involve digitizing an analog object, such as a paper document or image, providing an output in digital format. Since banking operations can be carried out without the need to physically go to a branch, many of these have been converted or closed to optimize in terms of cost reduction and maximize the services offered.
The positive aspects of technological development can be seen not only from the point of view of computing capacity and data storage but also from the aspect of error commissioning and prediction of results.

In the last decade, the analysis of financial time series has undergone an important development, both in terms of basic research and direct market applications. In particular, the computerization of data has made detailed information on the trend of prices and volumes traded (even at the level of a single executed transaction) easily available, thus creating new fields of investigation. At the same time, the introduction of electronic trading systems made large financial institutions interested in automating trading processes through algorithms.
The proximity to the client bank I worked for during my internship increased my interest in this field. Therefore, I decided to conduct a study on financial time series, to compare the effectiveness of modern predictive tools typically used.

1.1 Banking context

The regulatory definition of *Bank* is based on the minimum criterion of the systematic and joint exercise of the activities of collecting savings from the public and granting credit. However, the regulations applied since 1994, inspired by a European model of "universal banking", admit a very wide range of operations: from credit intermediation to securities intermediation, from proprietary trading to delegated trading, from payment services to service activities.
The distinctive features of the bank are linked to several main economic aspects:

- the performance of the monetary function, which is based on the acceptance of bank debt by the public as a means of payment. This acceptance, in turn, presupposes that the individual bank and the banking system enjoy public confidence;
- the function of transferring financial resources between surplus and deficit units. In performing this function, the bank may become part of the credit circuit, which entails taking up trading positions on its account (and hence the associated risks). Or it can implement maturity transformation, i.e. taking up asset/liability positions involving a transformation of maturities from short to medium/long term. In addition, it can also perform an ex-ante (funding) selection and ex-post control function;
- The transformation and risk management function. A bank's stability in the availability of funds is usually ensured by a diversification logic. This, therefore, makes portfolio risk acceptable.

Banks have always been the main financier of families and companies in our country. Banks are financial intermediaries, therefore one of their main focuses is to help to transfer funds, from those who intend to employ them in a remunerative form to those who need to make productive expenditures.
Banks are among the main intermediaries authorized to trade financial instruments on their account, in the execution of orders from clients.
Below we will analyze one of the available products provided by banks.
In general, financial instruments are a particular category of contracts having as their object rights and services of a financial nature, which aim at an economic return.
"The legislative decree of 24 February 1998 distinguishes financial instruments into the following types:

- stocks
- bonds
- government securities
- negotiable debt securities
- mutual funds
- futures, swaps, options, and forward contracts
- combinations of the various contracts listed above"[**1**]

Let us first introduce the acronyms S.p.A., Società per Azioni, and S.a.p.a., società in accomandita per azioni: the companies identified by these terms are corporations, i.e. they have agreed to divide their capital into shares. They are organizations of persons and means for the joint exercise of productive activity, endowed with full patrimonial autonomy.
The S.p.A. is certainly the prototype of the capital companies and constitutes the main model of a commercial company more suitable for large investments, in fact, its capital is represented by stocks.

1.1.1 Stocks

Stocks are the object of the empirical analysis in this work: it is, therefore, appropriate to fully understand how they work and how they are used by financial actors. In this section, we therefore briefly describe what they are.

Stock is "a financial product characterized by high risk, high return and flexible trading". The objective of every investor is to obtain a return on the shares held. This is not trivial as one has to estimate the share price trend; this is influenced by many factors such as the economic situation of the country, the occurrence of an unforeseen event, such as Covid, or the managerial decisions of the company.
Stocks are securities whose function is to represent the ownership of a portion of a company's capital. Its investors, therefore, own part of the company itself, even if it is only a very small fraction. Units of stock are called shares.
By being part of the company's share capital, one acquires administrative rights, such as the right to exercise a vote at shareholders' meetings, and financial-economic rights, i.e. the possibility of receiving dividends.
Not all stocks are the same and therefore do not have the same administrative and economic rights. It is the company itself that decides on the characteristics of the shares, both when it sets up and when it raises new share capital.
Stocks defined as *ordinary* are the most classic. They do not have any privileges in terms of dividend distribution, or administrative powers, or even, in the worst case, when the company is liquidated.
Another type is that of *savings stocks*, issued exclusively by companies listed on a regulated market. Holders obtain economic privileges, such as a privilege in terms of dividend distribution, which is manifested by a guaranteed minimum or a minimum spread compared to ordinary shares. However, they lose the right to vote at ordinary or extraordinary meetings.
The last two categories of shares are *preference stocks*, which are characterized by a right of priority over distributed profits and/or repayment of capital in the event of bankruptcy, and *restricted voting stocks*, which have an explicit limitation on voting rights at general meetings without necessarily having equity privileges in return.
Owning each of these different types of shares allows the investor, in some way, to take a position on the future of the company: from the future development of interest rate sensitive financial asset prices or stock market quotations, or even exchange rates, the buyer will be entitled to vote in proportion to his possession.

1.1.2 Risk management

The financial system also performs a risk management function in a more direct form than is derived from factors as information, liquidity, or transformation. There are two essential components of this activity:

- Foreign contracts encompass a wide range of applications, from commodities to financial instruments. Futures trading in commodities is a way of encouraging the development of trade and a trader can manage the risk he faces about his position for that commodity. The problem can be significant, particularly concerning the future price development of the commodity itself.

- Insurance business is a special area of activity in the financial system that has as its object the trading of so-called pure risks, i.e. those that occur in the form of future losses or damages that cannot be defined in frequency or severity. The management of pure risks through an insurance policy involves transferring the risk to a specialized intermediary (insurance company): the insured transforms a damaging future event that is uncertain in its severity and frequency (and therefore in its cost) into a certain cost. The insurance company can meet its compensation obligations through a pooling process, i.e. by taking on a sufficiently large and diversified number of risks, for the whole of which the overall cost can be predicted with good approximation. In strictly financial terms, the transfer of pure risk has as its counterpart the payment of premiums that are invested in reserves (usually financial investments), from which the funds needed to compensate policyholders will be taken.

One of the main objectives of a bank is risk management, i.e. to counteract losses in an investment. It is necessary to find methods to identify and predict risks to mitigate their possible effects.
Through market forecasting, it is possible to contingency risk of losses incurred "when the market moves in a direction opposite to our expectations".[**2**]
Thus, risk management is used to reduce losses and maximize revenues, even when facing possible market fluctuations and problems. These can be caused by economic, social, or political events, which affect market performance suddenly and often unexpectedly.
For these reasons it is important to know "in advance the risks of an investment to ensure that we can withstand them if things go wrong."[**2**]

Risk is a crucial issue for a multitude of parties, and its correct identification and measurement can be decisive in achieving the desired objective. As can be trivially assumed, the banking sector is the area where most research work related to this issue is concentrated. This interest stems from the fact that the risk of default of debtors has a direct bearing on the core business of banks.

At this point, we understand how these problems can be addressed and solved. The evolution of computers has increasingly enabled the development of numerous algorithms that can be exploited in the financial field. Moreover, in recent decades, a large number of works based on Deep Learning models have been published.

Building a trading algorithm means, in simpler words, deciding to buy or sell a certain asset to profit from the increase or decrease in price. A trading algorithm can be based on simple rules, such as observation of past trends or specific indicators, or on highly complex and non-linear mathematical models, which Deep Learning models attempt to approximate in some way. These algorithms can be based on prediction models: one starts from models whose aim is to predict the future price or at least its fluctuation, and from there one draws the decision on the purchase or sale of the security subject to the prediction.

1.2 The structure

Throughout this thesis, I want to highlight the most innovative aspects of forecasting the values of the stock price, based on deep learning methods.
In the second chapter, a brief introduction to the state of the art on forecast formation and composition will be made in detail. The most frequently used methods for time series prediction in trading are then shown. It starts with the most basic method which is the decomposition method, used to identify the various components of a time series to make a forecast. Then the moving average model and the autoregressive model are studied, two possible types of exponential smoothing are analyzed, and finally we will see the linear regressive model.
The next chapter, the third, focuses on Machine Learning and Deep Learning, two branches of Artificial Intelligence, the theory behind them, and their main differences. The five macro-categories of learning models are described: supervised learning, unsupervised learning, reinforcement learning, ensemble learning, and deep learning. Then it's about Deep Learning, applied to time series data. Specifically, there will be an explanation of the prediction models used.
The beginning of the fourth chapter discusses the data analysis and prediction of stock prices values collected day by day for one year. The objective is to forecast its values to be made over the next 5 working days. Specifically, we will try to establish whether the models used are actually able to provide precise and accurate predictions, and therefore applicable to a real context.
To conclude, the last part of the thesis is dedicated to the conclusions that can be drawn from the analyzes carried out on this data. Moreover, the bases are laid for further investigation and future research.

A final aspect to consider is that of software, i.e. the programming language I use to build the models and their algorithms. First of all, let us distinguish between *front-end* and *back-end* programming languages. A programming language is more high-level the more it uses pre-built built-in functions based on other functions that are already implemented and tested in files called libraries. They can be incorporated into the main code and then called up and used. A back-end language, on the other hand, constitutes the functions in the libraries on which high-level languages rely.
During data analysis, prediction and testing, Python was used exclusively as a front-end language. This choice was made because it exploits the functions of the most popular and useful libraries for the implementation and management of Neural Networks, such as the Keras or Tensorflow libraries. These are back-end languages.

1.3 Thesis objectives

The development of Artificial Intelligence in the form of Deep Neural Networks has allowed the evolution of attempts to solve a problem that unites various scientific fields: the study of historical series, and in particular, the prediction of future values of the series.
The central object of this thesis is the forecasting of temporal data. At the base of the study, first of all, it is necessary to understand the benefits that bring applications of forecasting at the strategic, tactical, and operational levels.
In general, forecasting data is useful for planning investments, optimizing revenue, and avoiding loss of money. Observing the trend in the future period is important to understand the current positive influencing factors and to set goals[**8**].
Time series analysis is a relevant and well-established subject of applied research, occupying areas as diverse as statistics, econometrics, dynamic systems theory, and so on. In the context of data forecasting, the study of time series makes it possible to:[**18**]

- Show the new possible models that allow predicting data in a time series
- Verify the advantages of using deep learning methods, through indicators such as the mean squared error
- Highlight the qualitative and quantitative indicators that influence the demand
- Briefly describe the trend of stock prices over time and assess the presence of any regularities or outliers
- Explain the phenomenon, identifying its generating mechanism and possible relationships with other phenomena

1.4 About Accenture

Accenture was founded in 1953 as the consulting division of Arthur Andersen; in 1989, this division permanently separated from the main company. Today, Accenture is the largest consulting firm in the world. It employs 513,000 people in some 51 countries.
Combining unique experience and expertise across all business functions, supported by the world's largest network of delivery centers, Accenture works at the intersection of business and technology to help clients improve their performance and create sustainable value for their stakeholders.
Accenture's core businesses are strategy, consulting, digital, technology and operations.

Chapter 2

The state of art

Here there will be presented the main phenomenological features of time series, and the standard linear models used to describe them and make predictions, in this case relating to loan values.
This chapter reviews the state of the art of methods and models dedicated to the analysis and modeling of time series, with the ultimate goal of forecasting. Initially, there is a brief introduction to the definitions of time series, what are stochastic processes, and in particular stationary processes.
Then the decomposition method is introduced, which constituted the first forecasting method based on separating detectable systematic patterns from accidental oscillations. The method dates back to the 1920s and still forms the basis for the most frequently used methods. Next was the ARMA model, the combination of the AR autoregressive model and the MA moving average model, which is adapted to time series data to analyze data or predict future data points on a time scale.
Then it's briefly reviewed the smoothing method, where the main idea behind this method is to transform the moving average in an average whose weights follow an exponential trend. The smoothing parameters have the effect to eliminate random disturbances so that, once a pattern has been identified, it can be projected into the future to produce the prediction. Finally, the linear regression applied to time series is presented.
At the end of the chapter, some simple techniques are shown to understand which of the methods analyzed is the most suitable for the data taken into analysis.

2.1 Introduction to time series analysis in forecasting

Forecasting techniques are therefore based on the use of historical data called time series, which are a sequence of observations ordered concerning time.
A **time series** is therefore a finite sequence of numbers $\{y_t : t = 1, ..., n\}$, containing the observed values of a variable Y from time 1 to time n. The purpose of time series analysis is to study the past evolution of the phenomenon to time; the prediction is obtained by assuming that these regularities of behavior are repeated in the future.

In Figure 2.1 it's possible to see an example of representation of time series.
to give a mathematically rigorous definition, it is necessary to introduce the

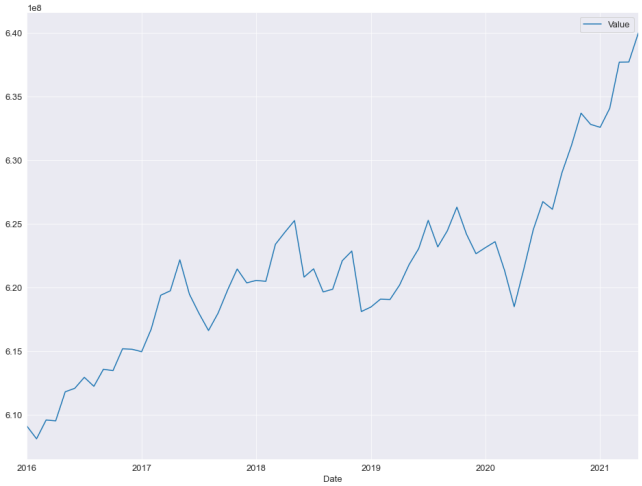


Figure 2.1: Example of time series

theory of stochastic processes.
Let (Ω, F, P) be a probability space, where Ω denotes a sample space, F is an σ-algebra of subsets of Ω and P is a measure of probability ($P : \Omega \to [0, 1]$).
Given a probability space $(\Omega; F; P)$, a time series can be considered as a finite realization of a stochastic process. **[18]**

Def 1. *A **stochastic process** is a set of random variables $(y_t)_{t\in T}$ defined on (Ω, F, P) at values in $(\mathbb{R}, B(\mathbb{R}))$, where T is a set of indices.*

So, a stochastic process is a family of random variables ordered concerning a parameter. Since a historical series is a finite set of observations related to a certain phenomenon, ordered in time, then it can be seen as a finite realization of a stochastic process.
It is possible to differentiate between these two definitions. While a time series is a collection of real numbers, ordered according to the time variable, which constitutes a finite part of a realization of a stochastic process, on the other hand, a discrete parameter stochastic process is characterized by its family of finite distributions, i.e. by the knowledge of all possible probability distribution functions.
If T is an interval of $\mathbb{R}$, then the stochastic process is said to be continuous and is denoted by $\{y(t) : t \in T\}$; if $T \subseteq N(\mathbb{Z})$, then the stochastic process is called discrete and is denoted by $\{y(t) : t \in T\}$. For simplicity we will describe only discrete time stochastic processes.
Fixed $t \in T$, $Y_t(\cdot)$ is a random variable called the state of the process at time t; fixed $\in \Omega$, the application $t \in T \to Y_t(\omega)$ is a trajectory (or realization) of the process, i.e., a time series (denoted by $Y(\omega)$).

Def 2. *Let T be the set of all vectors $\{\boldsymbol{t} = (t_1, ..., t_n)' \in T^n : t_1 < ... < t_n, n = 1, 2, ...\}$. Let's call (finite-dimensional) **distribution functions** of the stochastic process $\{Y(t) : t \in T\}$ the functions $\{F(t), t \in T\}$ defined for $\boldsymbol{t}=$*

$(t_1, ..., t_n)'$ *as*

$$F_t(y) = P(Y_{t_1} \leq y_1, ..., Y_{t_n} \leq y_n), \quad y = (y_1, ..., y_n)' \in \mathbb{R}^n$$

If the distributions of $(y_{t_1}, ..., y_{t_n}), \forall (t_1, ..., t_n) \in T, \forall n \in \mathbb{N}, \forall \mathbf{y} = (y_1, ..., y_n) \in \mathbb{R}^n$ is compatibly specified and the distribution functions $F_t, \forall t \in T$ is known, then it's known probabilistically the process $\{Y_t\}_{t \in \mathbb{R}}$.
The main goal of time series analysis is to identify an appropriate stochastic process that has trajectories that fit the data, to make predictions. It is necessary to restrict our attention to a class of stochastic processes that allows us to identify univocally the process, that is the model. So here it's presented a very important class of stochastic processes is that of stationary processes.
A time series is *stationary* when at least the mean and the variance of data don't depend on the time index.
There are two types of stationary process:

Def 3. *A time series* $\{y_t\}_{t \in T}$ *is said to be strongly stationary if* $\forall k \in \mathbb{R}, \forall t_1, ..., t_n \in T, \forall n \in \mathbb{N}, \forall$ $\boldsymbol{y} = (y_1, ..., y_n) \in \mathbb{R}^n$

$$F_{Y_{t_1}, ..., Y_{t_n}}(y_1, ..., y_n) = F_{Y_{t_1+k}, ..., Y_{t_n+k}}(y_1, ..., y_n)$$

Consequently, the joint distribution of a series that meets this definition is not altered by a time slip k. However, this is a very difficult condition to verify in practice.

Def 4. *A process* $(y_t)_{t \in T}$ *is said to be weakly stationary of order* k *if* $\mathbf{E}[Y_t] = \mu$ *(constant and independent from i) and* $\mathbf{E}[Y_{t_1} Y_{t_2}] = \mathbf{E}[Y_{t_1+k} Y_{t_2+k}], \forall t_1, t_2 \in T, \forall h \in \mathbb{R}$

In this case, the statistical moments of the process up to k depend only on time differences and not upon the time of occurrences of the data being used to estimate the moments. So this kind of stationarity requires that the first two moments are invariant (because the stationarity of the mean value implies the stationarity of the variance), concerning time.
Generally, in banking and finance, the time series is assumed to be weakly stationary and, from now, we will consider it to be in this case. When this requirement is not met by the empirical conditions, it is sufficient to fall back to the stationary case where $Z_t = Y_t - Y_{t-1}$ is taken as the series. By iteratively repeating this process, the requirement for stationarity is satisfied.
The study of time series can be divided into analysis and forecasting. The analysis is used to understand what are the relevant aspects of a phenomenon that has already occurred, which can be functional to the forecast.
Figure 2.2 shows the trend of a stationary series, which has an expected null value that can be observed visually.
The models that tend to be used to predict future values are built from the structure of the series in the past, net of accidental variations, which will be distinguished from the structure to avoid random fluctuations influencing the future trend.
Let's define also the concept of *lag*: the first lag of a time serie y_t is y_{t-1}, while the k^{th} lag is y_{t-k}[**8**].
If density functions have moments, it is possible to define that the mean is

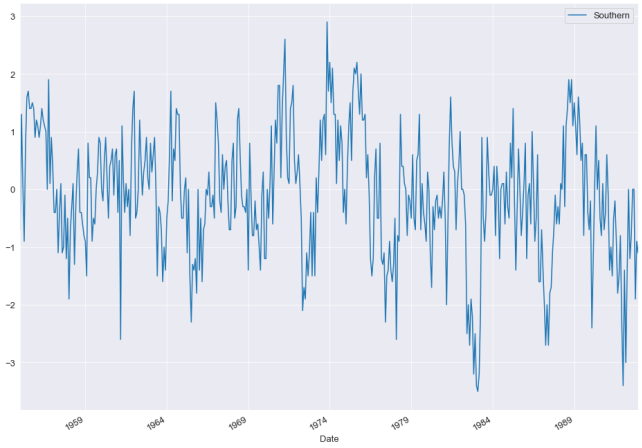


Figure 2.2: Example of stationary series

$E(Y_t) = \mu_t$, the variance is $Var(Y_t) = \sigma_t^2$, and the covariance is $Cov(Y_t, Y_{t-k}) = \gamma_{k,t}$.
In the case of stationary stochastic processes, the mean and variance will take on finite and constant values, μ and σ^2 respectively. However, the covariances at lag k are defined as

$$\gamma_k = \mathbf{E}[(y_t - \mu)(y_{t-k} - \mu)]$$

also known as autocovariances since they are not a function of time i but of k, i.e. such that $\gamma(k) = \gamma_k$.
Now let's introduce the concept of autocorrelation, also called serial correlation

Def 5. *"Autocorrelation measures how a time series is related to a time-shifted version of itself. The autocorrelation of y_t at lag k is*

$$\rho_k = \frac{Cov(y_t, y_{t-k})}{\sqrt{Var(y_t)Var(y_{t-k})}} = \frac{\gamma_k}{\gamma_0}$$

where γ_0 is the variance of the time series with itself, starting with $\rho_0 = 1$ ".[12]

Given a set $\{y_t\}_{t=1}^n$, it is possible to estimate ρ_k as

$$\hat{\rho}_k = \frac{Cov(y_t, y_{t-k})}{Var(y_t)} = \frac{\sum_{t=1}^{n-k}(y_t - \bar{y}_0)(y_{t+k} - \bar{y}_k)}{\sqrt{\sum_{t=1}^{n-k}(y_t - \bar{y}_0)^2 \sum_{t=1}^{n-k}(y_{t+k} - \bar{y}_k)^2}}, \qquad 0 \leq k < n-1$$

where $\bar{y}_0 = \frac{1}{n-k}\sum_{t=1}^{n-k} y_t \bar{y}_k = \frac{1}{n-k}\sum_{t=k+1}^{n} y_t$ is the mean of the sample under examination.
The autocorrelation function identifies internal similarities in the time series. "If a series has a pronounced periodicity but it does not have a pronounced trend, then $\hat{\rho}_k$ will have a high value at period; otherwise, if there is a pronounced trend, the autocorrelation estimate becomes quite high everywhere. It is, therefore, necessary that the trend be mild so that the function $\hat{\rho}_k$ is useful"[**13**].
$\hat{\rho}_k$ is always between -1 and 1, and is worth $\hat{\rho}_k \sim 1$ if the time series tends, approximately, to repeat itself i.e. $y_t \sim y_t + k$. Peaks in the autocorrelation function thus indicate a presence of a seasonal component in the time series.

2.2 Time series analysis

A historical series can be thought of as the composition of several components:

- the *trend-cycle* can be broken down, in turn, into the trend, that is generally is generally a long-term monotonic trend movement, which highlights a structural evolution of the phenomenon due to causes that act systematically on the same; and cyclical fluctuations, originated by the expansion and contraction of the economic environment in which the phenomenon under review is located. This component varies slowly over time and essentially determines the level of the series;
- the *seasonality* indicates whether there are one or more periodic components, which are found the same or almost at a fixed distance in time; it is present in the case of interim time series, such as monthly or quarterly;
- the *irregular component* is a disturbance component that includes residual variations not explained by the previous components, which oscillations are typical of a short period.

Here in the analysis, we can see the decomposition of the Italian granted loans of the *Famiglie* category. There are two types of approaches to the analysis

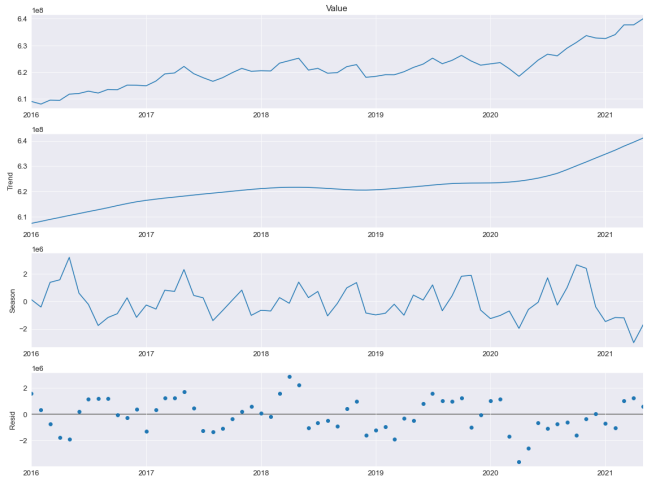


Figure 2.3: Decomposition of a non-stationary time series

of time series: the *classical approach* and the *modern approach*. The classical one assumes that the process represented by the series includes a deterministic part, which allows estimating the virtual components defined above, and a random disturbance component. The decomposition method is the only one among those that will be reported in this chapter that has an approach of classical type. The modern approach instead assumes that the series has been generated by a stochastic process with correlated components that can be described with appropriate probabilistic models.
to make statistical predictions, it is necessary to understand which of the alternative techniques to choose; therefore, it is appropriate to know the assumptions

that form the backbone of the various methods.

The steps through which a time series forecast is made are:

1. collection of data and verification of their quality and correctness. It is necessary to evaluate well the provenance and definitions of the available data and their comparability over time. It is necessary to correctly prepare the data from a data typology point of view
2. preliminary analysis of the trend and structure of the historical series: analysis of the series from a graphic point of view and with descriptive indices to highlight the possible presence of the oscillations of interest, trend, cycle, or seasonality
3. choice and estimation of the most appropriate model for the estimate of the components of the series
4. evaluation of the goodness of the model and its use to fine forecasting

The first step that is useful for understanding data and analyzing a time series is the display of graphs, showing the trend and the possible presence of fluctuations.
Generally, one starts with the construction and observation of a graph covering the whole series, also called long period or time plot, in which the values of the observed phenomenon Y are reported, inordinate, at each time t, along the x-axis.
If the level of the series remains more or less the same over the observed period
If in the observed period the level of the series remains more or less the same, that is the series is stationary on average, the spline of the time plot should oscillate around a constant value equal to the average of the series; on the contrary, if the series is evolutionary, that is not stationary, the time plot highlights the increasing or decreasing linear or non-linear trend.

2.3 Models

In the case of analysis considered in this thesis, stochastic time series with discrete values will be studied, where there is a trend for a time interval of 5 years, measured at regular intervals every month. This section is dedicated to the explanation of the different models usually applied to time series to obtain a prediction of the data in the future.
Let's start with the decomposition method.

2.3.1 Decomposition method

Many forecasting methods rely on the fact that, if a systematic pattern exists, it can be identified and separated from any accidental fluctuations by methods of equalization or smoothing of the time series data. The effect of smoothing is to eliminate random disturbances so that, once the pattern is identified, it can be projected into the future to produce the forecast.
The decomposition method, as noted above, uses a classical approach that assumes, therefore, that the series is composed of variations that are systematic or

deterministic, such as trend, cycle, and seasonality, and of disturbance or random oscillations. If therefore, there are patterns that repeat and it is assumed that this behavior continues to occur over time, it is possible to use them to predict future values.
It is necessary therefore to estimate the components of the trend $\mathcal{T}$ and the seasonality s, considering a model of the type $Y_t = f(\mathcal{T}_t, s_t, \epsilon_t)$.
These components are mainly composed according to an additive model or a multiplicative form. So, on the basis of the previous definitions, it is possible to write the historical series $y_1, ..., y_n$ as the so-called *additive form* of decomposition:

$$y_t = \mathcal{T}_t + s_t + \epsilon_t$$

where the series $\mathcal{T}_1, ..., \mathcal{T}_n$ is the trend, $s_1, ..., s_n$ is the seasonality, while the series $\epsilon_1, ..., \epsilon_n$ is that of the residuals. An additive model is appropriate when "the magnitude of the seasonal oscillation does not vary with the level of the series".
If, on the other hand, "the seasonal fluctuation increases (or decreases) proportionally as the level of the series increases (decreases), then a *multiplicative model*, as shown below, is more appropriate"[**14**]:

$$y_t = \mathcal{T}_t \cdot s_t \cdot \epsilon_t$$

It will be mainly treated the analysis only for the additive model, as this can also be applied to the multiplicative model, after a suitable logarithmic transformation of the data:

$$log(y_t) = log(\mathcal{T}_t) + log(s_t) + log(\epsilon_t)$$

Many economic series exhibit seasonal fluctuations that increase as the level of the series increases; therefore, in economics, the multiplicative model finds wider application.
This procedure is more of a time series analysis tool, and therefore requires some refinement to produce the forecast.
Considering for simplicity only the additive case, the classical decomposition is conducted by performing the following steps:

1. *Calculation of the trend-cycle of first approximation.* This is an instrumental step that does not produce a definitive estimate of the trend-cycle component. The trend-cycle of first approximation is calculated with a moving average centered at k terms. To study the jumps that are created, the idea is to first start by *identifying the trend.* To do this it is necessary to average the series, using the so-called moving averages smoothing of order m. Moving averages are a smoothing technique that has to analyze data points by creating a series of averages of different subsets of the full data set.

   $$\mathcal{T}_t = \frac{1}{m} \sum_{i=-k}^{k} y_{t+i}$$

   where $m = 2k + 1$. A m-term moving average, m odd, loses $\frac{(m-1)}{2}$ terms at the beginning and as many terms at the end of the series. Otherwise with m even, the time t and $t + 1$ don't correspond to a real observed

value, so it's necessary to center the moving average.
The loss of the first terms is of little importance, while the loss of the most recent terms has important consequences for the forecasting operation. A possible solution is to make moving averages at the extremes with a smaller number of terms. In general if it is considered that the periodicity is fairly uniform over time along the series, then one chooses k large; if, on the other hand, the periodic structure changes over time, then one chooses q small.
The search for the trend means to search for a component of the series known and if you want to find a mean value of the series, which when $t = 1, ..., n$ represents the trend, you can calculate the symmetrical average around t, of the type

$$\hat{\mathcal{T}}_t = \frac{y_{t-k_1} + ... + y_{t+k_1}}{2k_1 + 1}$$

where k_1 is an arbitrary positive integer value.
There is however an effect of recursive data processing: the local trend follows with a certain delay the data. This occurs because the local model, e.g. at the time t, is affected only by data prior to t and not by future data.

2. *Detrending.* This step is the simple subtraction of the trend from the time series $\{y_t\}$, i.e. series in which the periodic and predictable part has been removed. Let's consider the historical series $\{y_t\}_{t=1,..,n}$, thus the *detrended series* for the additive form is

$$z_t = y_t - \hat{\mathcal{T}}_t = s_t + \epsilon_t$$

In the case of the multiplicative model, the adjusted series is posed as

$$z_t = \frac{y_t}{\hat{\mathcal{T}}_t}$$

In both cases the new series $\{z_t\}_{t=1,..,n}$ will has a value close to zero.

3. *Estimation of the seasonal component.* To compute $\hat{s}_t$, let's introduce the mixed seasonality series and error for the additional model $SE_t = y_t - z_t$ (analogously for the multiplicative model, that uses the division). In the classical approach it is assumed that the seasonal oscillation is constant from year to year, so, with monthly data $s_t = s_t + 12 = s_t + 24 = ...$ The seasonality coefficient s_m for month m $(m = 1, ..., 12)$ is calculated by making the arithmetic mean of the SE_t terms where $t = m, m + 12, m + 24, ...$Then the seasonality estimate for January $\hat{s}_1$ is given by the arithmetic mean of the SE values referring to January.

4. *Derivation of the seasonally adjusted series.* The same process could be done for the seasonally adjusted series that, in the case of an additive model, is derived as

$$d_t = y_t - \hat{s}_t$$

after the estimation of s_t, and it's possible to do the same thing for the multiplicative case. The series d_t thus contains the cycle-trend pattern

and the effect of disturbance.
The trend of seasonally adjusted data should show a fairly smooth line, devoid of those regular and marked fluctuations typical of the seasonality of the series. Seasonal adjustment using the additive model is not always satisfactory: there is a tendency to seasonalize too much in some periods of the year, where it is possible to notice an inversion of the peaks, and too little in others, where a very high peak remains. The reason lies in the fact that the additive model implicitly assumes that the range of variation of seasonal fluctuations within the year remains constant, otherwise it will be noticed instead the presence of a certain cyclicity of the same ones.

5. *Estimation of the entire systematic component of the series.* By means of the seasonality and trend-cycle estimates we obtain the estimate $\hat{y}_t$, which contains only the systematic pattern of the series, where:

$$\hat{y}_t = \hat{\mathcal{T}}_t + \hat{s}_t$$

Otherwise for the multiplicative case it has to do the product between the two values[**15**].

This method is to the base of the models of decomposition.
"While classical decomposition is still widely used, it is not recommended, as now there are several much better methods. Some of the problems with the classical decomposition are summarized below.
Although the classical decomposition model is still widely used, however, it is not recommended"[**13**]. The first reason is that the trend estimation lacks the first and last observations, as well as the residual component estimation, and it greatly flattens the price trend at rapid increases or decreases. In addition, the seasonal component in this method is repeated from year to year. This may be considered reasonable for many series, but for some longer series, it is not, as this decomposition fails to capture seasonal changes over time. In addition, unusual changes, influenced by external components not easily predicted, are hardly highlighted.

There are some variants to the model of simple decomposition: here we report the X11 decomposition and the STL decomposition.

X11 decomposition

"The X11 method is an iterative method, which is used to decompose quarterly or monthly type data. More frequent data, like daily or hour data, are not appropriate for this method.
It is based on classical decomposition it handles both additive and multiplicative decomposition but includes many extra steps and features to overcome the disadvantages of the classical decomposition, discussed above. The aspect that complements this method over the classical is the existence of a trend estimate for all observations, including endpoints. In addition, the seasonal component can vary slowly over time. X11 also has some sophisticated methods for handling variation in trading days, vacation effects, and the effects of known predictors. Vacations occur at the same time each year, so they are included in the estimates of the seasonal component. Another advantage is that X11 is relatively

robust to outliers and level shifts
The process is entirely automatic and tends to be very robust to outliers and level shifts in the time series.[**13**]

STL decomposition

STL stands for "Seasonal and Trend decomposition using Loess", while Loess is a method for estimating nonlinear relationships.
STL is a decomposition method, robust and versatile, similar to X11. It has features that improve it such as handling seasonality, as it can be used on daily or hourly data. The user can control the smoothing of the trend and change the rate of change of the seasonal component over time.
This is also a method that is robust to outliers; in fact, it acts so that occasional unusual observations will not affect the trend-cycle estimates and seasonal components. However, they will affect the residual component.
STL only provides facilities for additive decompositions, however, it's possible to obtain a multiplicative decomposition by taking the logarithms of the data.

These methods are also used to forecast the seasonal component by repeating the previous year and seasonally adjusted data using the non-seasonal time series method. It is usually used to understand the data and be able to easily visualize it and then build a forecasting model"[**13**].

2.3.2 ARMA process

ARMA processes constitute a family of stochastic processes. Before analyzing their characteristics, it is necessary to recall some concepts.

Moving average (MA-q)

Now let's consider the stochastic process $\{y_t\}_{t=1}^{T}$. The moving average is a simple model that smoothes the historical series. It predicts the future value of the time series by using past errors as the explanatory variables.
If the series is composed only of trend and residual components, the moving average eliminates the effects of disturbances.
If the seasonal phenomenon of period q is also present in the original series, then a moving average of amplitude q can eliminate seasonality as well. In the two cases, the moving average aims to isolate the trend-cycle.
This procedure is called moving average because each successive average is calculated by removing the oldest value and inserting a new one. It is a local fitting method in that it creates a series of smoothed values of length equal to the original series, each corresponding to observation time t.
As the terms increase, the spline joining the points identified by the moving averages becomes smoother. Consequently, assuming to average over k values, i.e. that is the average over the period, let's denote the moving average of order q as:

$$\hat{y}_t(q) = c + \epsilon_t + \theta_1\epsilon_{t-1} + ... + \theta_q\epsilon_{t-q}$$

where θ_t is the weight to be attributed to the t-th observed value and ϵ_t is the *white noise*.
Let's talk about the process $\{\epsilon_t\}_t$: this is called white noise when (ϵ_t) is a

succession of identically distributed uncorrelated random variables with zero mean and positive variance σ_Y^2. The variance of the error term will only change the scale of the series, not the patterns. So the white noise is a stochastic process with at least the first two moments constant in time, so the process is stationary but does not leave the process memory of itself. If these characteristics are respected, it corresponds to a random signal that has equal intensity at different frequencies, thus giving it a constant power spectral density. For the white noise series, we expect each autocorrelation to be close to zero.
An MA(q) process, then, is a process obtained as a combination of several elements of the same white noise that exhibits persistence characteristics that are more pronounced the higher its order.
Let's examine its moments

$$\mathbf{E}(\hat{y}_t) = \mathbf{E}[\sum_{i=0}^{q} \theta_i \epsilon_{t-i}] = \sum_{i=0}^{q} \theta_i \mathbf{E}(\epsilon_{t-i}) = 0$$

so a process MA has average 0. This feature could make these processes seem difficult to apply to real situations, since, in general, it is not said that the observed time series oscillate around the value 0. However, the limitation is solvable because for each process y_t for which $\mathbf{E}(y_t) = \mu_t$ we can always define a new process $z_t = y_t - \mu_t$, which will have a zero mean. If z_t is stationary in covariance, then it is enough to study z_t and then add back the mean to get y_t. Having obtained this result, the variance can simply be written as

$$Var(y_t) = \mathbf{E}(y_t^2) = \mathbf{E}[(\sum_{i=0}^{q} \theta_i \epsilon_{t-i})^2] = \mathbf{E}[(\sum_{i=0}^{q} \theta_i^2 \epsilon_{t-i}^2 + \sum_{i=0}^{q} \sum_{j \neq i} \theta_i \theta_j \epsilon_{t-i} \epsilon_{t-j})]$$

which can be simplified, since the expected value of the second term is null, as

$$Var(y_t) = \sum_{i=0}^{q} \theta_i^2 \mathbf{E}(\epsilon_{t-i}^2) = \sum_{i=0}^{q} \theta_i^2 \sigma^2 = \sigma_2 \sum_{i=0}^{q} \theta_i^2$$

which is a finite value when q is finite.
Finally it is possible to calculate the autocovariance of order k as

$$\mathbf{E}(y_t y_{t+k}) = \mathbf{E}[(\sum_{i=0}^{q} \theta_i \epsilon_{t-i})(\sum_{j=0}^{q} \theta_j \epsilon_{t-j+k})] = \sum_{i=0}^{q} \theta_i (\sum_{j=0}^{q} \theta_j \mathbf{E}(\epsilon_{t-i} \epsilon_{t-j+k}))$$

due to $\mathbf{E}(\epsilon_{t-i}\epsilon_{t-j+k}) = \sigma^2$ for $j = i+k$, otherwise it is equal to 0, the expression is rewritten as

$$\gamma_k = \mathbf{E}(y_t y_{t+k}) = \sigma^2 \sum_{i=0}^{q} \theta_i \theta_{i+k}$$

Autoregressive model (AR-p)

"Another important process is the Autoregressive model of order p. The processes in this model provide a more intuitive representation than that obtained with MA processes since the idea is that the level of the series at the time i is a linear function of its past values, plus a white noise"[**13**].

It comes said autoregressive model because the forecast is obtained by using a linear combination of the p past observations of the series

$$\hat{y}_t(p) = \sum_{i=1}^{p} \beta_i y_{t-i} + \epsilon_t + c$$

where $\beta_1, \beta_2, ..., \beta_p$ are the parameters to estimate, c is a costant term and ϵ_t is the white noise. Fixed the order p, which will indicate the order of the autoregressive model, the value of the parameters of the model is obtained solving the following program of estimation of the least-squares.
Then the identification of the AR(p) model is done by iteration: starting with $p = 1$, the model parameters are calculated by solving the program, the prediction error is calculated, and a test, which verifies if the process is white noise. If the test fails we increase the order by 1.
Since the model is iterative, let's start from AR(1) series. To compute the moments of the process, first, assume the stationarity of the process. Let's start from the expected value, that it's supposed to be equal to the constant value μ:

$$\mu = \mathbf{E}(y_t) = \beta_1 \mathbf{E}(y_{t-1}) + \mathbf{E}(\epsilon_t) = \beta_1 \mu$$

This expression is valid for:

- $\mu = 0, \ \forall \beta_1$
- $\beta_1 = 1$, the series y_t is equivalent to a random walk and the mean is indeterminate.

The variance of this process, instead, could be written as:

$$Var(y_t) = \mathbf{E}(y_t^2) = \mathbf{E}[(\beta_1 y_{t-1} + \epsilon_t)^2] = \beta_1^2 Var(y_t) + \sigma^2 + 2\beta_1 \mathbf{E}(y_{t-1}\epsilon_t)$$

The last element is equal to zero because it is a linear combination of autocovariances of a white noise, so it's deducted the value of the variance $\frac{\sigma^2}{1-\beta_1^2}$, which is stable in time only if $|\beta_1| < 1$.
The autocovariance for the first iteration ($k = 1$) is

$$\gamma_1 = \mathbf{E}(y_t, y_{t-1}) = \mathbf{E}[(\beta_1 y_{t-1} + \epsilon_t) y_{t-1}] = \beta_1 Var(y_t)$$

but, for a generic order k is

$$\gamma_k = \mathbf{E}(y_t, y_{t-k}) = \mathbf{E}[(\beta_k y_{t-1} + \epsilon_t) y_{t-k}] = \beta_k \gamma_{k-1} \quad \Rightarrow \quad \gamma_k = \beta_1^k \frac{\sigma^2}{1 - \beta_1^2}$$

The autocorrelations are an index of memory of the process: the larger β_1 is, the more the autocorrelation grows. It is therefore possible to deduce that this parameter characterizes the persistence.

ARMA(p,q)

ARMA stands for AutoRegressive Moving Average, in fact it's the combination between the moving average model MA(p) and the autoregressive model AR(q),

where these two models are the boundary cases.
It can be written as

$$\hat{y}_t = \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \sum_{k=1}^{p} \beta_k y_{t-k} + c$$

where

- p = order of the autoregressive part;
- q = order of the moving average part.

The stationarity condition is satisfied if the polynomial characteristic of the component AR has solution greater than 1 in modulus." If the series is not stationary (the mean and variance are not constant over time) it is performed a transformation of the data. In this way is obtained a stationary series, a random walk"[**13**]. The procedure proposed by Box and Jenkins is iterative and is used for the identification, estimation, and verification of an ARMA model and aims to build a model that fits the time series observed and represents the generating process of the series itself. Through the analysis of the graphs of the series, it is easier to identify any outliers.
The p, q values of the model are identified through the analysis of the partial and total autocorrelation functions, and the ARMA model parameters are estimated through the maximum likelihood or least-squares method.
Finally, there is model verification: a check is made on the residuals of the estimated model to see if they are a sample realization of a white noise process with Gaussian components. If the estimated model passes the verification phase then it can be used for the decomposition and/or the predictions. Otherwise, the identification, estimation, and verification steps are repeated, proceeding iteratively.

2.3.3 Exponential smoothing

"Exponential smoothing is a technique used for forecasting, proposed in the late 1950s by Brown, Holt, and Winters"[**13**].
"The forecast produced using exponential smoothing methods is made through the weighted averages of prior observations, whose weights decrease exponentially as the observations become older: the more recent the observation, the higher the associated weight. This is often better than moving average models that allocate the same weight for each period. This structure generates reliable predictions quickly and for a wide range of time series.
The procedures of the most important exponential smoothing methods and their application in the prediction of time series with various characteristics will be reported below. Method selection is generally based on the recognition of key components of the time series, such as trend or seasonal components, and how these enter the smoothing method (e.g., additive or multiplicative).
Exponential smoothing attempts to make the original series smooth in the same way that the moving average does, and uses the smoothed data obtained to predict future values"[**16**].

Simple Exponential Smoothing

The simplest of the exponential smoothing methods is called simple exponential smoothing (SES).
This method is mostly used when the data that does not have a clear trend or seasonal pattern, because it doesn't recognize trends or seasonal components.
SES is then used to predict future data when the data does not have a specific slope and does not follow a pattern due to seasonal factors. In fact, single exponential smoothing estimates only the level component.
The forecast value at time $t+1$, given the observations of the series $y_1, ..., y_t$, is given by

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \dots$$

where α is the smoothing parameter, with $0 \leq \alpha \leq 1$, which is the weight of each parameter. If $\alpha = 0$, relevance is given to the historical data, i.e., the predicted future values are the average of the historical data, while if $\alpha = 1$, the values to be predicted are the results of the recent observation, i.e., it is the most recent values that influence the forecast.
To chose the optimal smoothing parameter may estimate the parameter by minimizing the sum of squared error.
The formula of the SES could be written also using the component form. Here it's included only the level component l_t.
The component form emphasizes the recursiveness of the formula, and it can be rewritten with two equations:

$$\begin{aligned} &\textit{forecast equation} && \hat{y}_{t+h|t} = l_t \\ &\textit{smoothing equation} && l_t = \alpha y_t + (1-\alpha) l_{t-1} \end{aligned}$$

The level l_t is the smoothed value of the series at time t . If $h = 1$ it gives the fitted values, while if $h = T$ "it gives the true forecasts beyond the training data".[**13**]

Triple Exponential Smoothing

"Triple exponential smoothing is an extension of exponential smoothing also called Holt-Winters exponential smoothing. This forecasting method is used when the data shows trend and seasonality. In fact, it explicitly adds support for seasonality to time series that have a systematic trend or seasonal component that do not increase over time"[**16**].
"The implementation of the method consists of choosing the initial values, analyzing their sensitivity to events as outliers, choosing the smoothing parameters, and normalizing the seasonal indices"[**17**].
The basic equations for this method are given by:

$$\begin{aligned} &\textit{forecast equation} && \hat{y}_{t+h|t} = l_t + h\mathcal{T}_t + s_{t+h-m(k+1)} \\ &\textit{overall smoothing equation} && l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + \mathcal{T}_{t-1}) \\ &\textit{trend smoothing equation} && \mathcal{T}_t = \beta(l_t - l_{t-1}) + (1-\beta)\mathcal{T}_{t-1} \\ &\textit{seasonal smoothing equation} && s_t = \gamma(y_t - l_{t-1} - \mathcal{T}_{t-1}) + (1-\gamma)s_{t-m} \end{aligned}$$

where:

- $\hat{y}_{t+h|t}$ is the forecast at h periods ahead

- l_t is the smoothed series at time t
- $\mathcal{T}_t$ is the trend factor at time t
- s_t is the seasonal smoothed series at time t
- m is the frequency of the seasonality
- k is the integer part of $\frac{h-1}{m}$, it ensures that the estimates of the seasonal index come from the final year of the sample
- α, β, γ are the smoothing parameters
- t is an index denoting a time period, $1 \leq t \leq T$

α, β, γ are constants that have to be optimal. These parameters must be estimated in such a way that the MSE of the error is minimized. When these parameters are estimated, it's possible to estimate the prediction $\hat{y}_{t+h|t}$.

2.3.4 Linear regression

One of the simplest models to use as a probabilistic and predictive approach for predicting stock price is linear regression. Although it is an elementary method, it is often used because it is easy to implement and accurate from a predictive analysis point of view.
Multiple linear regression is a statistical method that allows us to model the relationships between two or more continuous quantitative variables:

- the forecast variable y is regarded as the response, outcome or dependent variable,
- the k predictor variables $x_1, ..., x_k$ are the explanatory or independent variables. Each variable has to be numerical.

The regression model could be written in the following way:

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + ... + \beta_k x_{k,t} + \epsilon_t$$

where the coefficients $\beta_1, ..., \beta_k$ measure the marginal effect of each predictor in the model, and the error ϵ_t, which captures anything that affect y_t other than x_t.
Using this model, there are some assumptions to make about the errors $\epsilon_1, ..., \epsilon_T$:

- "they have mean equal to zero, otherwise the forecast will be biased
- they are not autocorrelated, otherwise the forecast will be inefficient
- they are unrelated to the predictor variables
- usually is normally distributed with constant variance σ^2" **[13]**

The focus is to estimate the best parameters $\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_k$ to obtain a good prediction $\hat{y}_t$. The most effectively way of choosing them is by minimizing the sum of the squared errors, so such that minimize

$$\sum_{t=1}^{T} \epsilon_t^2 = \sum_{t=1}^{T} (y_t - \beta_0 - \beta_1 x_{1,t} - ... - \beta_k x_{k,t})^2$$

Obtained the estimated parameters, the predictions of y can be written

$$\hat{y}_t = \hat{\beta}_0 + \hat{\beta}_1 x_{1,t} + \hat{\beta}_2 x_{2,t} + ... + \hat{\beta}_k x_{k,t}$$

Let e_t , for $t = 1, .., T$, the so-called residuals, defined as

$$e_t = y_t - \hat{y}_t$$

that are the differences between the observed values y_t and the corresponding fitted ones $\hat{y}_t$. After selecting the regression variables and fitting a regression model, it is necessary to plot the residuals to check that the model assumptions have been met. Several graphs should be produced to check different aspects of the fitted model and the underlying assumptions.
"When fitting a regression model to time series data, it is common to find autocorrelation in the residuals, since it is very likely that the value of an observed variable in the current period is similar to its value in the previous period. In this case, the estimated model violates the assumption of no autocorrelation in the errors, and our predictions may be inefficient"[**13**]. In truth, in this case, a model's predictions are not considered to be wrong, but they will have wider prediction intervals than necessary. Therefore we should always look at an ACF plot of the residuals.
"Another useful autocorrelation test in the residuals designed to account for the regression model is the Breusch-Godfrey test, also called the LM (Lagrange Multiplier) test for serial correlation. It is used to test the joint hypothesis that there is no autocorrelation in the residuals up to some specified order. A small p-value indicates that there is significant residual autocorrelation in the residuals.
It is useful to view the histogram of the residuals to check if they are normally distributed. This analysis is not essential for forecasting, but it makes calculating the forecast intervals much easier"[**13**].

2.4 Assessment of model goodness

A common way to summarise how well a model fits the data is via the coefficient of determination, or R^2 . This can be calculated as the square of the correlation between the observed values and the predicted values. It can also be calculated as,

$$R^2 = \frac{\sum_{i=1}^{k}(\hat{y}_{i,t} - \bar{y})^2}{\sum_{i=1}^{k}(y_{i,t} - \bar{y})^2}$$

"It is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. Whereas correlation explains the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of one variable explains the variance of the second variable. If the predictions are close to the actual values, we would expect R^2 to be close to 1"[**5**].
However, it is not usually used because it doesn't measure how the model will forecast future data and, in addition, R^2 does not allow for degrees of freedom,

so adding any variable tends to increase the value of R^2, even if that variable is irrelevant.
To solve this problem, it's designed the so-called adjusted R^2

$$R^2_{adj} = 1 - (1 - R^2)\frac{T-1}{T-k-1}$$

where T is the number of observations and k is the number of predictors. This is an improvement on R^2 because if a predictor is added, it will no longer increase. The best model will be the one with the largest value of R^2_{adj} .

The choice of the model is fundamental to obtaining a reliable forecast. In the field of time series analysis for forecasting purposes, there are two types of evaluation to determine which of the models best fits its situation: first, you can estimate, based on the chosen model, the theoretical values of the series and then compare the estimated data $\hat{y}_t$ with the observed values y_t. In this way it is verified like the model is adapted to the data and succeeds to reproduce them.
The person who has to make a prediction then chooses between the methods just shown by comparing them. This strategy of comparing different methods can be carried out in two different ways.
The first is to calculate the sum of the least-squares of the residuals of the methods. In this case, we speak of the goodness of fit and in formal terms the model residual $\hat{\epsilon}_t$ is:

$$\hat{\epsilon}_t = y_t - \hat{y}_t$$

When h values are predicted in the future, to check that the predicted values are reliable, the goodness of prediction is measured, which in formal terms indicates a prediction error at time $t + h$:

$$\hat{\epsilon}_{t+h} = y_{t+h} - \hat{y}_{t+h}$$

Obviously, the prediction error, as defined above, can only be calculated when data for future times are available. to assess the predictive capabilities of the model, it is useful to assume that the model's prediction errors behave similarly over time.
Let's therefore introduce as evaluation measures the mean square error (MSE), which is a measure of the variability of the estimates provided by an estimator that corresponds to:

$$MSE = \frac{1}{N-h}\sum_{i=h+1}^{N} \hat{\epsilon}_i$$

As the precision of the estimates increases, so does the efficiency of the estimator, in the sense that the degree of uncertainty in the estimates obtained from a sample survey decreases. a sample survey.[**18**]
The MSE calculation is one of the most user-friendly techniques that allows you to quickly and easily check which of the models best approximates and predicts the data.

The second way is cross-validation. This principle is based on dividing the time series into two parts: the first part, normally the oldest data, is called the

training set; the second part, the most recent data, is called the *test set*. The models to be compared are then applied to the training set to find the optimal parameters of the respective methods. Then, using them in the formulas of the respective models, the prediction of the next period, which is the test set, is made. In this way, the model is built on the available data, and then its performance is examined with regard to how well it manages to predict future data, which do not coincide with those on which it was generated.

2.4.1 Forecast uncertainty

When applying a prediction method from a time series $y_1, ..., y_n$, a value $\hat{y}_{n+1}$ is obtained. This is called point estimate of the value at time $n+1$. On the other hand, one can speak of interval estimation which evaluates the uncertainty of the point estimate. In general, we set a *confidence level*, for example 95%, and we have to calculate the δ such that at 95% the value at time $n+1$ will fall within the interval

$$[\hat{y}_{n+1} - \delta, \hat{y}_{n+1} + \delta]$$

This estimate can be made more flexible by finding δ_- and δ_+ . For example, if it's established that 5% has to be divided into equal parts, 2.5% each, it's required that 2.5% is the probability of having values less than $\hat{y}_{n+1} - \delta_-$, 2.5% is the probability of having values greater than $\hat{y}_{n+1} + \delta_+$.**[19]**

Chapter 3

Artificial Intelligence

Artificial Intelligence is a branch of computer science that enables the programming and design of both hardware and software systems, whose characteristics are considered typically human, such as visual, spatio-temporal, and decision-making perceptions.
Artificial Intelligence was born to hand in hand with the advent of computers in the 1950s, but remained unexplored until the early 1990s because, until then, the computing power of the first computers was insufficient.
In recent years, Artificial Intelligence has become the subject of intense interest and central to the debate on data analysis and study, as it is considered one of the most promising fields of the moment.
It deals with the problems people have daily, ranging from interpreting language to distinguishing what is inside an image.
In themselves, these systems, which are called intelligent, are not intelligent models, but rather capable of learning. They are created in such a way to acquire this ability, i.e. they are programmed with algorithms whose steps allow the model to react appropriately to the data received by modifying the functioning algorithm itself.
This chapter provides the essential context for artificial intelligence, machine learning, and deep learning.
Figure 3.1 graphically represents the relationship between the three different branches, Artificial Intelligence, Machine Learning, and Deep Learning. The image is taken from [**3**].

Machine learning is a sub-field of AI, so it is one of the methods of approaching artificial intelligence, and it has been applied to many different fields, from biology to financial markets.
Machine learning overturns the usual concept of programming, where the individual steps of the algorithms created are known: it is based on retrieving a lot of data, creating a large number of algorithms, and creating functions to understand and comprehend the data we have, excluding irrelevant ones. The aim is to improve the results over time and obtain a system that can decide on its own based on the available data.
"Machine learning refers to the various mechanisms and algorithms that enable a machine to improve its capabilities and performance over time. The machine, therefore, will be able to learn to perform certain tasks by improving, through experience, its capabilities, responses, and functions"[**22**].

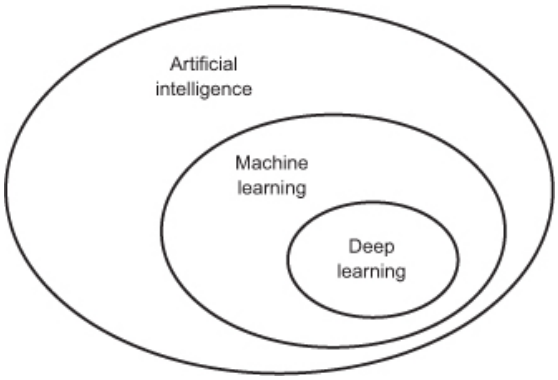


Figure 3.1: AI, Machine and Deep Learning

At the basis of machine learning are a series of different algorithms that, starting from primitive notions, will know how to make a specific decision rather than another or perform actions learned over time.
The learning techniques used are trained on large amounts of data to make the algorithm capable of classifying more specific individual patterns. Based on the patterns recognized by the model and trained on the dataset, the algorithms will be able to make predictions.
"The learning models can be divided into three different categories based on how the machine learns and accumulates data and they are supervised, unsupervised and reinforcement learning"[**23**].
Furthermore, there are also two more complex types of learning, that are ensemble learning and deep learning.

Supervised Learning

The models that belong to this typology are built starting from the memorization of data and information acquired with the experience of the algorithm. The machine can classify a problem based on and drawing on the data stored and processed over time. The machine's only objective is to be able to choose the best response to the input it is given.
"Through Supervised Learning, we try to build a model from labeled training data, with which we try to make predictions about unavailable or future data. Each piece of the data that's passed to the model during training is a pair that consists of the input object or sample along with the corresponding label or output value. So the model is learning how to create a mapping from given inputs to particular outputs based on what it's learning from the labeled training data"[**24**].
Supervised Learning, therefore, means that in the set of samples or datasets, the desired output signals are already known because they are previously labeled. In this type of learning, based on labels of discrete classes, we will therefore have a task based on classification techniques.
The algorithms that make use of supervised learning are used when there is the necessity to make inductive hypotheses, that is, hypotheses that can be obtained by scanning a series of specific problems to obtain a suitable solution to a problem of a general type.

Then the supervised are divided into two other sub-categories of extraction methods which are *Classification* or *Regression*, depending on the variable if cardinal or quantitative numerical type.
With classification, the goal, "based on the analysis of previously labeled data, is to be able to predict the labeling of future data classes. Labels are unordered discrete values that can be considered as belonging to a group of a class".[**4**]
In contrast, in linear regression the output is a continuous value, so the machine's task is not to find the result but to find a relationship between the input and output values.
Finally, the model will classify the output obtained and then determine the error by looking at the difference between the value it predicted and the actual label.

Unsupervised Learning

The unsupervised learning foresees instead that the information inserted inside the machine is not codified, that is, the machine can draw on certain information without having any example of its use and, therefore, without knowing the expected results depending on the choice made. This is because, unlike supervised data, these data are unlabelled or unstructured. Therefore, the machine itself will have to catalog all the information in its possession, organize it and learn its meaning, its use, and, above all, the result to which it leads. This type of learning does not have a variable label.
"Essentially, with unsupervised learning, the model is going to be given the unlabeled data set and it's going to attempt to learn some type of structure from the data. It extracts the useful information or features from the data analyzed and it's going to be learning how to create a mapping from given inputs to particular outputs based on what it's learning about the structure of this data without any labels"[**25**].
The models that belong to this subdivision are the *Clustering* models or *Association Rule* models. One of the most popular applications of this kind of learning is the use of clustering algorithms. Clustering is "an exploratory technique that allows the aggregation within groups, called clusters, of data for which there is no previous knowledge of belonging to groups". The result is large datasets in which the data has similar elements. Within each cluster, we will find data with many similar characteristics. Clustering is "an excellent technique that allows us to find relationships between data".[**4**]
Learning without supervision offers more freedom of choice to the machine that will have to organize the information intelligently and learn which are the best results for the different situations that arise.

Reinforcement Learning

The learning by reinforcement represents probably the most complex system of learning, which foresees that the machine is equipped with systems and instruments able to improve its learning and, above all, to understand the characteristics of the surrounding environment. The objective of this type of learning is "to construct a system (agent) that through interactions with the environment improves its performance".[**4**]
This type of learning arises to improve the functionality of the system and occurs through the introduction of reinforcements, that is, reward signals. This

reinforcement is not given by labels or correct truth values but is a measurement of the quality of the actions undertaken by the system. For this reason, it cannot be equated with supervised learning.
Currently, with the widespread use of Deep Learning algorithms, this type of learning is back in fashion.
Initially, the moves will be completely random and without logic. From the moment in which the system receives positive feedback, it will receive a greater weight and, consequently, positive reinforcement on that action. Conversely, in the case of a negative action, the value of the weights on that action will decrease. Therefore, as a consequence of these reinforcements, the system gives more weight to the moves that have brought it greater benefits and tend to replicate the same behavior on future moves. This type of learning is typical of driverless cars, which, thanks to a complex system of supporting sensors, can drive along city and country roads, recognize obstacles, follow directions, and much more.

Ensemble Learning

Ensemble techniques combine individual machine learning models to improve the stability and predictive power of the model.
This model aims to use the positive and efficient aspects of different models because some of them do well in modeling one aspect of the data, while others do better in modeling another. The combined strength of the models offsets individual model variances and biases. In this way, ensemble learning provides a composite prediction where the final accuracy is better than the accuracy of individual models.
Ensemble methods can be divided into two sub-groups: the Sequential ensemble methods and Parallel ensemble methods. In the first case base, learners are generated consecutively and its main motivation is to use the dependence between the base learners. The overall performance of a model can be boosted. Instead, the parallel methods are applied wherever the base learners are generated in parallel and the basic motivation is to have independence between them.
"Model averaging is an approach to ensemble learning, where each ensemble member contributes an equal amount to the final prediction. For example in the case of regression, the ensemble forecast is calculated as the average of the member predictions; in the case of predicting a class label, the prediction is calculated as the mode of the member predictions. The class probability is calculated as the argmax of the summed probabilities for each class label.
A limitation of this approach is that each model has an equal contribution to the final model. A possible solution to this problem is to use the weighted average ensemble, that is an extension of the model averaging, where the contribution of each member to the final prediction is weighted by the performance of the model"[**26**].

Let's introduce the idea behind *Bagging* and *Boosting* and their main differences. "While the training stage is parallel for bagging (i.e., each model is built independently), boosting builds the new learner in a sequential way".[**21**]
A bagging method, also called bootstrapping, combines the results of multiple models to get a generalized result from a single model. It reduces the variance of an estimate by taking the mean of multiple estimates. There are three steps

to perform it:

1. Create randomly sampled data sets of the original training data;
2. Build and fit several classifiers to each of these diverse copies;
3. Take the average of all the predictions to make a final overall prediction.

Instead, boosting reduces bias by training weak learners sequentially, each trying to correct its predecessor. In general, boosting is a technique of changing weak learners into strong learners. Now we have the main steps for this method:

1. Train a classifier H1 that best classifies the data concerning accuracy;
2. Identify the regions where H1 produces errors, add weight to them and produce a H2 classifier;
3. Aggregate those samples for which H1 gives a different result from H2 and produces H3 classifier.

3.1 Deep Learning

Deep learning is a particular Machine Learning technique, applicable in certain contexts, which is based on software that manages to mimic the functioning of neurons.
To understand how deep learning works, it is necessary to start with **Neural Networks**.
The concept behind Neural Networks is the reproduction of typical human reasoning but done by machines. This means that intelligent systems, with the acquisition of more and more algorithms, are moving in the direction of imitating humans in more and more situations. These algorithms can make decisions based on both background knowledge and knowledge gained through the algorithm's experience.
Deep learning represents this idea of successive layers of representations. "The number of layers that contribute to a data model is called the depth of the model. In deep learning, these layered representations are (almost always) learned through models called Neural Networks, structured in layers on top of each other"[**28**].
Deep learning creates multi-level learning models, where higher concepts are learned from lower levels.

Artificial Neural Networks are information processing systems that use a mathematical model to simulate the structure and behavior of biological Neural Networks.
By biological Neural Networks, we mean the connected groups of nerve cells, while by artificial Neural Networks we mean mathematical constructs, initially designed to reproduce the mechanism with which they carry out the activities of reasoning and learning.
An artificial Neural Network attempts to replicate the behavior of this very powerful machine by creating a large number of connections between very simple elements, the artificial neurons, and realizing a distributed structure characterized by a high degree of parallelism and, above all, able to learn and provide a

response given a starting situation that has never been seen before.

The definition of artificial Neural Network is supplied by Professor Robert Hecht-Nielsen who defines a Neural Network as "a processing system consisting of a series of simple and highly interconnected processing elements, which process information through their dynamic state response to external inputs".
The simple, highly interconnected processing elements are neurons, also referred to as nodes or units in the Neural Network. Neurons are usually arranged in layers, which can be of three types:

1. Input Layer: a layer designed to receive information from outside to learn to recognize and process the same information received;

2. Hidden Layer: they connect the input layer with the output one and help the Neural Network to learn the complex relations analyzed by the data. Often the hidden layers are more than one;

3. Output Layer: the final layer that shows the result of what the program has been able to learn.[**38**]

Each level is made up of thousands of artificial neurons, indicated in the figure by dots, which interconnect between the various levels, in the representation above they are indicated by arrows.
Most Neural Networks are fully connected, i.e. every neuron belonging to the hidden layer is connected to every neuron in the output layer.
Each connection between neurons is associated with a weight that determines the importance of the input value. The initial weights are set randomly. Each neuron, instead, has an activation function that allows defining output for a neuron given a series of input data analyzed by it. Once a set of input data has passed through all the layers of the Neural Network, it returns the output data through the output layer.
Each neuron performs a single elementary operation if the total amount of signal it receives exceeds a certain activation threshold and, once the total amount of signal it receives exceeds a certain activation threshold and, once active, it, in turn, emits a signal that is transmitted along the communication channels to the other units to which it is connected.
In this context, learning means finding a set of values for the weights of all layers in a network, such that the network will correctly map example inputs to their associated labels.

3.1.1 Deep Learning applied to time series

Time series forecasting is a particularly challenging problem "compared to other types of classification and regression problems because, in addition to the inherent difficulties of forecasting, there is the complexity of inter-temporal dependence between variables"[**27**]. As mentioned in the previous chapter, the problem of time series forecasting was limited to essentially linear or linearizable methods, such as ARMA. But classical methods have limitations:

- They do not support missing data within the series, and in the case it is therefore necessary to use interpolation methods to overcome the problem;

- They have more difficulty in capturing non-linear relationships;
- Support mainly univariate models;
- They are often limited to a single-step prediction horizon.

We are helped by the Deep Neural Networks that, as we will see, are better able to compensate for this lack of the classic methods, exploiting their superior capacities in mapping input-output functions, even if these are highly nonlinear. Time series can be extremely different from each other and therefore over time different types of Neural Networks have emerged to accommodate these differences. In this section, we will review the most widely used approaches to time series prediction using deep learning algorithms. The objective to be achieved through the use of time series prediction models is to predict future values $y_i(t)$, where i denotes a given entity at time instant t. In the specific case that we will analyze in the next chapter, with each different logical grouping i we indicate the respective location and sector of customers of the loan granted at time t.

The simplest forecasting models are called one-step-ahead, as they predict a single value at time t+1, and have the following form

$$\hat{y}_i(t+1) = f\Big(y_i(t-k:t), x_i(t-k:t), s_i\Big)$$

where " $\hat{y}_i(t+1)$ is the model forecast, $y_i(t-k:t) = [y_i(t-k), ..., y_i(t)]$, $x_i(t-k:t) = [x_i(t-k), ..., x_i(t)]$ are observations of the target and exogenous inputs respectively over a look-back window k, s_i is static metadata associated with the entity, and f is the prediction function learnt by the model".**[37]**

3.2 Types of Deep Neural Networks

Different Neural Network structures are used depending on the learning method used and the purpose of the application. The main categories of Neural Networks are listed below:

- **Perceptrons**: are the simplest form of Neural Network originally designated a network consisting of a single neuron, altered by weighting and a threshold value;
- **Feed Forward Networks**: these can only carry information in one processing direction, so backward connections or connections to the same layer are not allowed. The networks can be single-layer, i.e. consisting only of input and output layers, or multi-layer with several hidden layers;
- **Recurrent Networks**: the output values of a layer of a higher layer are used as input to a layer of a lower layer. These interconnections between layers allow the system to create a memory in the recurrent networks there are feedback connections (usually to neurons of the same layer, but also backward). This greatly complicates the flow of information and training, requiring the behavior to be considered at multiple instants in time behavior in several temporal instants. These networks are more suitable for the management of sequences because they have a memory effect (of the

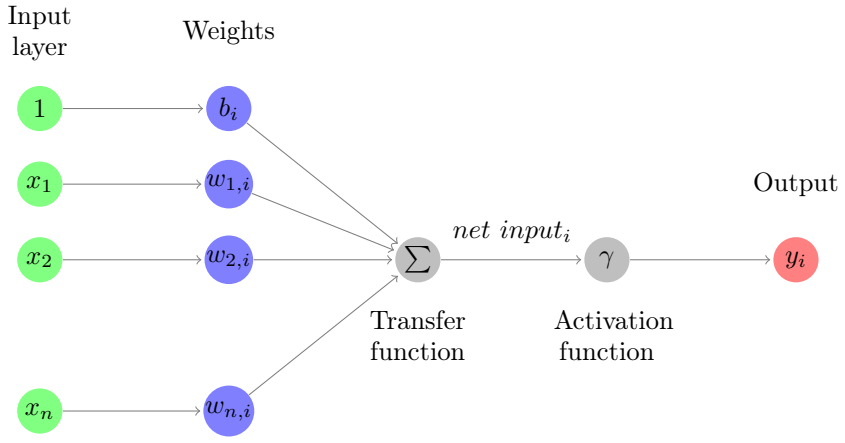
short term) that at time t makes available the information processed at $t-1$, $t-2$, etc. A particular type of recurrent network is LSTM (Long Short Term Memory);

- **Convolutional Networks**: these networks are a subset of the multilayer networks. They consist of at least five layers. Pattern recognition is performed on each layer, where the result of one layer is transferred to the next layer. This type of Neural Network is used for image recognition.

Among the various typologies of Networks, some are better than others can face the characteristics of the problem relative to the prediction of the financial historical series: Multilayer Perceptron, Recurrent Neural Networks, Convolutional Neural Networks.[**31**]

3.2.1 Perceptron

Before introducing the different types of deep learning architectures, let us recall some basic structures that are common and underlie them all." First we introduce the Perceptron, which is the basic element of many deep learning algorithms, inspired by the functionality of biological neural circuits, and for this reason it is also called a neuron"[**30**].
The perceptron is the unit that makes up a Neural Network. Each layer of the network contains a certain number of perceptrons, that receive a certain number of input signals and produce an output that will be sent to the perceptrons of the following layer if we are in an intermediate stage, or it will be the output of the network if we are in an xoutput layer. The architecture of a perceptron is shown in the figure below.



The above graph is a basic perceptron diagram. The objective of a perceptron is to calculate the output of the i-th perceptron

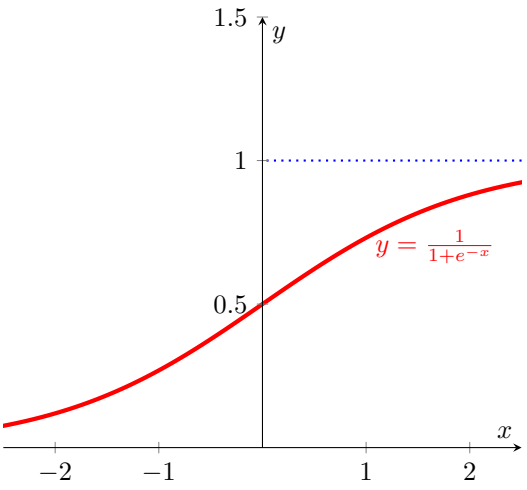
$$net\ input_i = \sum_{j=1}^{n} x_j w_{j,i} + b_i$$

that is the sum of the n inputs, each of them multiplied by the corresponding weight w_i, to which the known term b_i, or bias, of the perceptron is added. The

bias is an additional weight that is considered linked to a fictitious input with value always 1; this weight is useful to calibrate the optimal working point of the neuron.
The result is subsequently fed into an activation function. The activation function is a fundamental aspect of a Neural Network, as it is what enables the reproduction of non-linear features of the functions to be mapped, and it can be written as

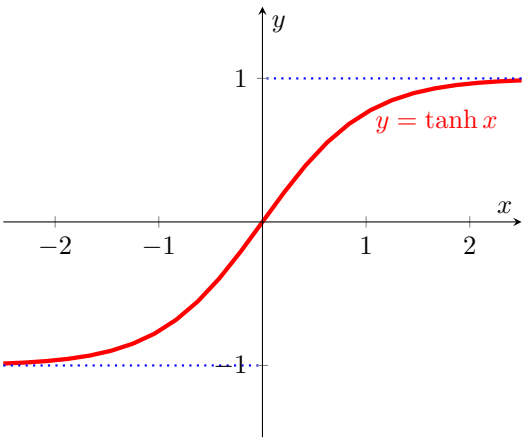
$$y_i = \gamma(net\ input_i)$$

It is a mathematical non linear function that is usually represented by functions such as the step function, the sigmoid function, the hyperbolic tangent function or the rectifier function.
The *sigmoid function* has a codomain in a continuous interval between 0 and 1. It is usually used to identify a binary variable, but the continuity of the function can give a probabilistic meaning to the output, which instead of representing the true value can represent the probability that the predicted value is equal to 1. Graphically it is displayed as:

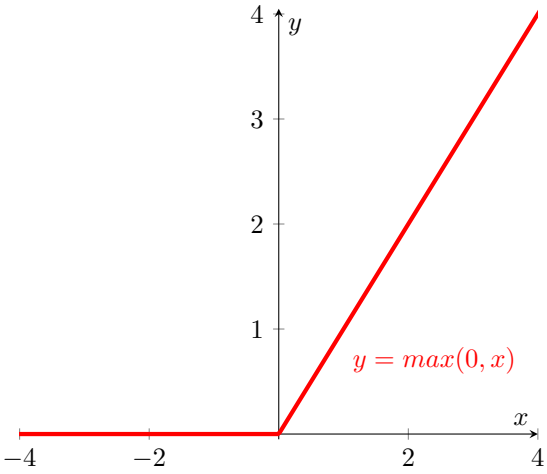


The *hyperbolic tangent* is in the form similar to the sigmoid, but the codomain is a continuous interval between -1 and +1. Analytically, it is expressed as follows:

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



The *rectifier function* (or ReLU) returns the value 0 if the input is less than 0, or the bisector line of the first quadrant if the input is greater than 0.
The rectifier function is the most used function in Neural Networks because it has been empirically demonstrated to be superior in the algorithm of setting the weights of the network, overcoming the so-called gradient vanish problem, which we will see later.



So the activation function plays an extremely important role, as without it a Deep Neural Network would be no better than a network consisting of only two layers, the input, and the output layer. It, therefore, determines the characteristics of the output and how the network learns.**[29]**

The link between the input and the output, that is, the function of transfer of the network, is not programmed but results from a process of learning that is usually, as we have seen, supervised, unsupervised, or reinforcement learning. In supervised learning, the network can learn from the training set to infer the relation that ties the inputs. A back propagation algorithm trains the network through the data in such a way as to modify the weights and other parameters of the network itself in such a way to minimize the error of prediction relative

to the training set. If the training done on the test set is successful, the network learns to grasp the unknown relation that links the input variables to the output ones, and is, therefore, able to make predictions even when the output is not known a priori.
The unsupervised training is instead based on algorithms that modify the weights of the network referring only to a data set that includes only the input variables. Instead, the reinforcement learning algorithms try to determine a policy aimed at maximizing the sum of the incentives received by an agent, endowed with perceptive capacity, where during its exploration of the problem and that gradually modifies such actions according to the consequences produced.

Then we will explain the supervised learning algorithm known as back propagation, which is the most widely used. First of all, it is necessary to define a metric that indicates the goodness of the model, an indicator whose parameters will then be modified to make the network as accurate as possible. For the Neural Networks, this indicator is represented by the **loss function**, that given the predicted value and the objective value returns the distance between the two. The objective is that this value is as low as possible.
The loss function most used in Neural Networks is the mean squared error.
In general any loss function can be expressed as

$$\eta(\hat{y} - y) = \hat{y} - y = \gamma(x, w, b) - y$$

where $\hat{y}$ is the output of the Neural Network and y is the actual value.
To minimise the η function, it will therefore be necessary to act on the choice of weights and biases, as the inputs are given.
Choosing as loss function the sum of the squares of the errors and considering the output vector $y = [y_1, y_2, ..., y_m]$, the error (for pattern x) is:

$$\eta(\hat{y} - y) = J(x, w) = \frac{1}{2} \sum_{j=1}^{m} (\hat{y}_j - y_j)^2$$

which quantifies how much the output produced for pattern x deviates from the desired one. The dependence on the w-weights is implicit in $\hat{y}$.
The error $J(w)$ over the whole training set is the average of $J(x, w)$ over all the patterns x belonging to the training set. This value can be reduced by changing the weights w in the opposite direction to the gradient of J. In fact, the gradient indicates the direction of greatest growth of a function (or more variables) and by moving in the opposite direction we reduce (at most) the error.
As we will see later when the minimization of the error occurs through steps in the opposite direction to the gradient the back propagation algorithm is also called gradient descent. There are also second-order minimization techniques that can accelerate convergence (applicable in practice only to networks and training sets of small/medium size).

Back Propagation Algorithm

The back propagation algorithm is a widespread technique used in the training of feed-forward Neural Networks for supervised learning.
First of all, the network weights must be initialized, usually randomly or based on predefined distributions. Initially, it is not possible to set the weights equal

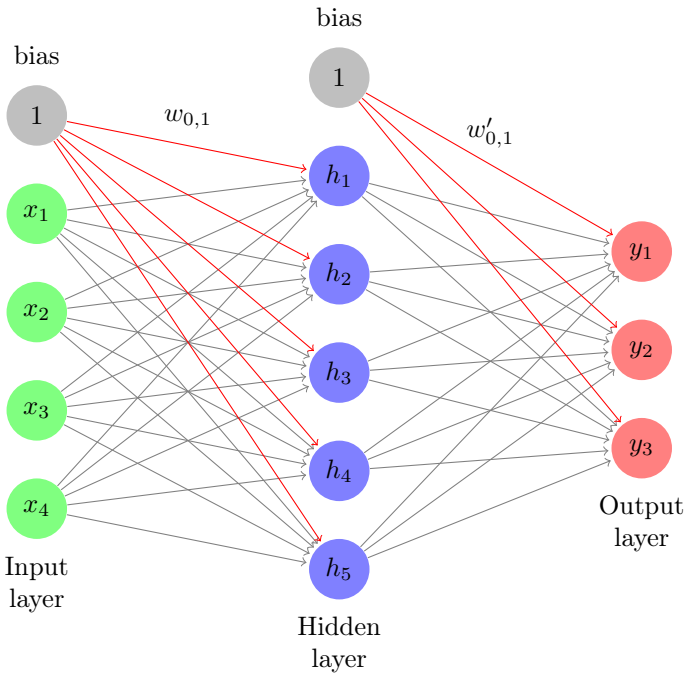
to 1 because the network would not succeed in nonlinear learning, so usually, we opt for constant values like 0.1 or 0.01.
The second step is to carry out an initial training run of the network. A complete training run occurs when the network completely scans the training set. Each cycle defined in this way is called an **epoch**.
In reality, the update of the weights does not necessarily take place at the end of each epoch but it is possible to specify it as a network parameter and it is called a batch. In any case, at the end of the first update, the network calculates the value of the cost function based on the expected values produced. [**31**]
Then the weights are calculated, obtained by minimizing the loss function. To minimize, the partial derivative of the loss function is calculated for each of the weights and the bias. The partial derivative indicates the tangent at the calculated point and represents the direction in which the displacement is to be made. The algorithm for calculating this direction is the so-called **Gradient Descent Algorithm**.
However, the deviation does not have to be equal to the whole value of the derivative, but only to a fraction, which is called the **learning rate**. The choice of this parameter determines the speed of the learning: in fact, a high rate determines fast learning; vice versa a low rate of learning indicates that the network will take more time to reach the objective.
Thus, the new weights will be calculated as follows:

$$w_i(t) = w_i(t-1) - \alpha \frac{\partial \eta}{\partial w_i}$$

where $w_i(t)$ and $w_i(t-1)$ are the new and the current estimates of the parameter w_i, and α is the learning rate. The slope of the loss function at $w_i(t-1)$ is given by $\frac{\partial \eta}{\partial w_i}$.
Once the new parameters have been calculated, return to step two, then recalculate the outputs with the new weights and recalculate the new error, and so on in a cycle of improvement until the chosen number of epochs has been completed.

Multilayer Perceptron

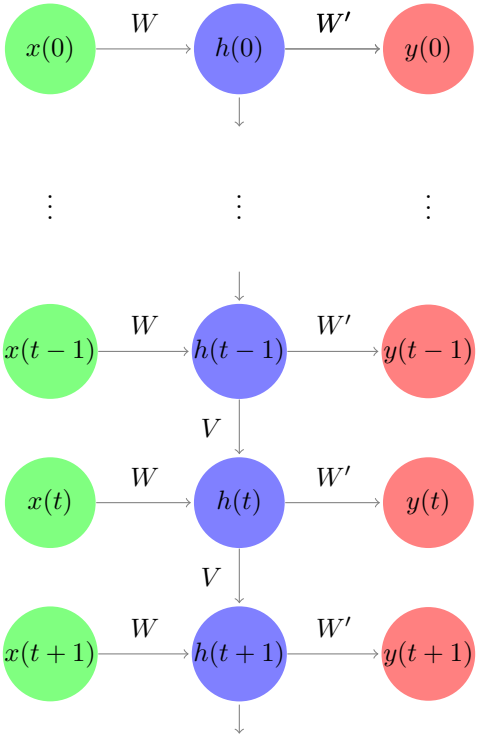
Deep Learning can therefore be defined as an artificial Neural Network that is composed of at least two hidden layers. In reality, Deep Learning applications tend to contain many more layers.
A Multilayer Perceptron (MLP) is a feedforward network with at least 3 layers, of which at least 1 is hidden, and with non-linear functions.
It is a network able to map in a very precise and accurate way a function of input-output, also nonlinear, even if the inputs and the outputs are multivariate and it supports a horizon of multi-step prediction. Despite their great effectiveness in the function mapping phase, MLPs suffer from some limitations concerning time series problems, as they do not have a suitable structure to capture the intertemporal dependencies that characterize them, which makes them not particularly effective in this type of context. [**31**]



The above example shows a 3-level simplified network which, in this case, has 4 input neurons, 5 neurons for the hidden layer, and 3 output neurons.
This type of network is defined as a feed-forward network since the flow of information is mono-directional. Therefore, the schematic information is fed into the network from the input layer, then it is split into the hidden layers and finally arrives at the output layer. Furthermore, this schematic network is fully connected, as each input of the previous layer sends a signal to all outputs of the next layer.

3.2.2 Recurrent Neural Network

The Recurrent Neural Networks (RNN) are a type of Deep Network not feed-forward, so there is back-propagation of some signals. The recurrent Neural Networks differ from those previously shown for the fact that they have a memory level, therefore, it is the type of network more used in literature to treat sequential data such as the historical series.
This is due to the existence of cycles, in fact, the output values of a higher-level layer are used as input data for a lower-level layer.
Consequently, what characterizes this type of network is the storage in one of the hidden layers of a state variable, which takes into account the history of the data passed during the mapping of the function. **[29]**
In the following figure, there is a basic scheme of an RNN in different time instants, where we want to highlight how the feedback loop is deployed along the time sequence, thus highlighting the correlations between future and past values.



In the figure there is a network formed by input states, hidden states and output states, whose weights are indicated with W,W',V. The output $y(t)$, i.e. at the instant t, is a function of the input $x(t)$ and partly of the value $V(t)$ that, due to the effect of the cycle, corresponds to the previous output $y(t-1)$. But the output $y(t-1)$ in its turn depended on the value of the input $x(t-1)$ and of $V(t-1)$ that, again for effect of the cycle, corresponded to the still previous output $y(t-2)$ and so on.
Each block corresponds to a cycle as described in the case of the MLP.
Specifically, at each instant of time t the output of the hidden layer is given by

$$h(t) = \gamma\Big(Wx(t) + Vh(t-1) + b_1(t)\Big)$$

where γ is the activation function, $x(t)$ is the current input, $h(t-1)$ is the output of the hidden layer at the previous instant and $b_1(t)$ is the bias. The output of the network at instant i will therefore be

$$y(t) = \gamma\Big(W'h(t) + b_2(t)\Big)$$

[29]
A typical problem of recurrent networks is the so-called vanishing or exploding gradient.
Also in the case of the RNNs, during the training phase of the Neural Networks, the method of the back propagation is used.
Two different problems can arise based on the chosen function of activation. If sigmoidal or hyperbolic tangent functions are used, the values of their gradients

remain in the range $[0, 1]$. Consequently, since the back propagation algorithm requires the gradients to be multiplied in a chain along with the layers, the product of a large number of values between 0 and 1 can lead to the overall value decreasing rapidly along the chain of neurons (*vanishing gradient*).
If, on the contrary, linear functions such as the ReLU or the pure linear are used, the values of the gradients can be also greater than 1, and then the overall value can grow enormously along the chain of neurons, making the value of the gradient explode (*exploding gradient*).[**32**]
To avoid running into these problems, there exist two special cases of a recurrent network, called LSTM and GRU.
Therefore, in general, recurrent networks also involve links backward or towards the same level. The two models we are going to look at specifically involving backward or same-level links. Let's begin with the most used one, the LSTM technique.

Long Short Term Memory

The structure of a neuron that belongs to this typology keeps track during training of both more recent data (Short Term) and data located on more distant time instants (Long Term). The output of the hidden layer is calculated in a more complex way: it is composed of structures called cells, in which the input and output signals are back-propagated in special perceptrons, as well as the measure given by the weight of the more remote states. These perceptrons inside the cell are called **gates** that are sigmoid functions with output in $[0, 1]$, which extract the value of the current input concerning the past of the series, to pass some or all of the information. A value of zero means "filtering out information completely, while a value of one means passing on information completely. The structure is called long-term memory because it uses short-term memory processes to create a longer memory".[**6**]
There are three types of gates: the input gate, the forget gate, and the output gate. All of them have the focus to allow blocking or passing the value.
For LSTM to preserve the memory of a long time ago, in addition to having the hidden layers, LSTM has layers called cell state to prevent old information from fading too soon.
Let's look at what these four main components mean:

- *Input Gate*: the objective is to acquire new information $x(t)$. There are two functions for acquiring new information: $i(t)$ and $\tilde{c}(t)$. The $i(t)$ concatenates the previous hidden vector $h(t-1)$ with the new information $x(t)$. The $\tilde{c}(t)$ has a similar behaviour. There is an element-wise multiplication between these two values, to obtain the state of the cell $c(t)$.

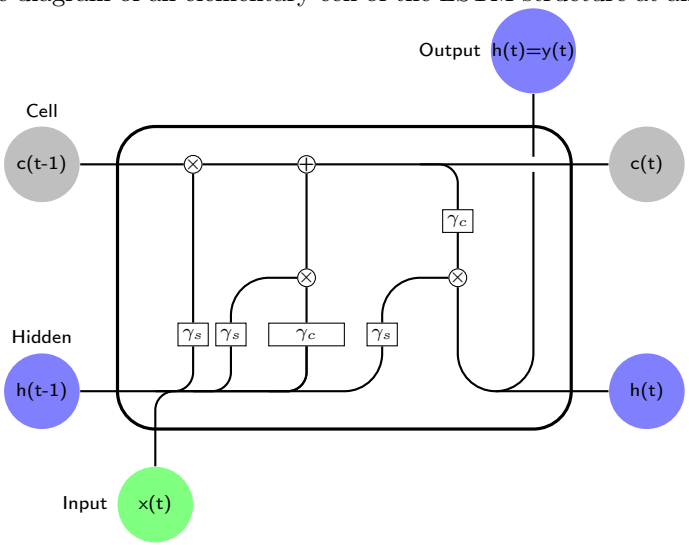
- *Forget Gate*: The forget gate $f(t)$ is very similar to $i(t)$ in the input gate. It controls the limit up to which a value is retailed in memory.

- *Cell state*: calculates a multiplication per element between the previous cell state $c(t-1)$ and forgetting gate $f(t)$. It then adds the results from the input gate $i(t)$ to $\tilde{c}(t)$.

- *Output Gate*: $o(t)$ is the output port at time step t The hidden level $h(t)$ goes to the next time step or rises to the output as $y(t)$. Note that the

output port $o(t)$ is not the output $y(t)$, it is simply the port to control the output.[**36**]

Finally, the hyperbolic tangent function returns the hidden layer $h(t)$ that will be given as input to the downstream perceptron to calculate the output.
The diagram of an elementary cell of the LSTM structure at any time t follows:



The equations which explain an LSTM unit with forget gate at the time t are:

$$
\begin{aligned}
f(t) &= \gamma_s\Big(W_f x(t) + V_f h(t-1) + b_f\Big) \\
i(t) &= \gamma_s\Big(W_i x(t) + V_i h(t-1) + b_i\Big) \\
o(t) &= \gamma_s\Big(W_y x(t) + V_y h(t-1) + b_y\Big) \\
\tilde{c}(t) &= \gamma_c\Big(W_c x(t) + V_c h(t-1) + b_c\Big) \\
c(t) &= f(t) \circ c(t-1) + i(t) \circ \tilde{c}(t) \\
h(t) &= y(t) \circ \gamma_c\Big(c(t)\Big)
\end{aligned}
$$

where:

- $x(t) \in \mathbb{R}^d$ is the input vector to the LSTM unit;
- $f(t) \in (0,1)^h$ is the forget gate's activation vector;
- $i(t) \in (0,1)^h$ is the input gate's activation vector;
- $o(t) \in (0,1)^h$ is the output gate's activation vector;
- $h(t) \in (-1,1)^h$ is the hidden state vector, that is the output vector of the unit. It's a short-term state (or memory);
- $\tilde{c}(t) \in (-1,1)^h$ is the cell input activation vector, which represents candidate for cell state;

- $c(t) \in \mathbb{R}^h$ is the cell state vector, a long-term memory;
- γ_s is the sigmoid functiona andγ_c is the hyperbolic tangent function .

We have to find the weight matrices $W \in \mathbb{R}^{h \times d}, V \in \mathbb{R}^{h \times h}$ for the respective gate, and bias vector $b \in \mathbb{R}^h$, which need to be learned during the training.
"The forget gate $f(t)$ decides if the input information has to be thrown away or kept. It receives the information of the current input $x(t)$ and the previous feedback output $h(t-1)$. To this information is applied a sigmoidal activation function that will return an output between 0 and 1. This output will be multiplied by the value of the previous state $c(t-1)$. Therefore, if the sigmoidal function returns a value close to 0, the previous state will tend to reset, i.e. it will be forgotten, while if it returns a value close to 1, the previous state will tend to remain the same and therefore be stored.
The input gate $i(t)$ similarly decides whether the input values $x(t)$ and $h(t-1)$ can be processed together with the previous state $c(t-1)$.
The processing of the inputs and the current state, i.e. the values that the forget and the input gate allow to pass, become the current state of the neuron $c(t)$.
Finally, the output gate, in a similar way to the other two gates, always exploiting the input values $x(t)$ and $h(t-1)$ decides if the current state $c(t)$ can be presented at the output and become the output $o(t)$ that also corresponds to the next feedback input $h(t)$".[**32**]

Gated Recurrent Units

GRU also aims to solve the problem of escape gradient. GRU does not have the cell state and output gate like those in LSTM. It, therefore, has fewer parameters than LSTM and uses hidden layers to transfer information. GRU calls its two gates the reset gate and the update gate:

- *Reset Gate*: this achieves what the input gate and the forget gate of LSTM achieve, in fact, it determines if the previous hidden state should be ignored. The gate $r(t)$ determines if the previous hidden state should be ignored. The gate $z(t)$ is generated for the update gate $\hat{h}(t)$.
- *Update Gate*: This part multiplies $r(t)$ and $h(t-1)$. The multiplication means how much of $h(t-1)$ will be kept or ignored. This creates a time $\hat{h}(t)$ to be used for updating $h(t)$.
- *Update Degree*: This part calculates the weighted average between $h(t-1)$ and $\hat{h}(t)$, based on the weight $z(t)$. If $z(t)$ is close to zero, past information contributes little and new information contributes more.[**36**]

The equations that represent this kind of Neural Network are:

$$
\begin{aligned}
r(t) &= \gamma_s\Big(W_r x(t) + V_r h(t-1) + b_r\Big) \\
z(t) &= \gamma_s\Big(W_z x(t) + V_z h(t-1) + b_z\Big) \\
\hat{h}(t) &= \gamma_c\Big(W_h x(t) + V_h h(t-1) r(t) + b_h\Big) \\
h(t) &= (1 - z(t)) \circ \hat{h}(t) + z(t) \circ h(t-1)
\end{aligned}
$$

where the new parameters are:

- $r(t)$ is the reset gate's activation vector;
- $z(t)$ is the update gate's activation vector;
- $\hat{h}(t)$ is the candidate activation vector;
- $h(t)$ is the update degree, which is the output vector;

It can therefore be said that GRU is a simplified version of LSTM, which tries to maintain its advantages, reducing parameters and complexity. In fact in this case there is only one memory state h(t) and only one gate (with inverted logic) to quantify how much to forget and how much to add.

3.2.3 Convolutional Neural Network

The Convolutional Neural Networks also belong to the class of feed-forward networks and are inspired by the organization of the visual cortex of the human brain.
This type of Neural Network was created to counteract networks that are too large and reduce their size in cases where they are impossible to train (this is the case, for example, when the network is fully connected). Therefore, they reduce "the number of parameters to learn by limiting the number of connections of the neurons in the hidden layer to only some of the input neurons"[**33**].
They were born to face problems of Computer Vision and pre-processing of the images, but in recent years they are finding application in many other fields; in fact, the basic idea is that an object is decomposed through simplifications that allow the recognition of the object in question.
The difference concerning the classic feed-forward networks is represented by the presence of the convolution layers, which carry out a very important job as they extract, through the use of filters, the characteristics of the images whose content is to be analyzed.
The convolution product is mathematically defined as follows:

$$X(t) * g(t) = \int_{-\infty}^{\infty} X(s)g(t-s)ds$$

where the function $X(t)$ represents a *signal* that we want to analyze, for example, an image or, in our case, a time series; the second function $g(t)$, which is called *filter*, runs over it and its function is to identify particular structures or patterns within the first signal. So the function of this network is precisely to recognize recurrent patterns within the input signal, and it is based on this that it is possible to exploit it in the prediction of historical series.[**29**]
The basic architecture of a CNN consists of an input block, one or more hidden layers, which perform calculations through activation functions, and an output block that performs the actual classification. In Figure 3.2 it's shown the complete architecture of this kind of Neural Network, taken from [**30**].

"In CNN architectures, there are several layers: convolutional, max-pooling, dropout, and the fully connected Multilayer Perceptron layer. The first layer is the convolutional layer, which consists of a filtering operation. It deals with the elaboration of a limited portion of the input where, in our case, we adopt a historical series imagining it as a one-dimensional image"[**34**].The single processed portions, obtained from the scanning mode with the convolution filter

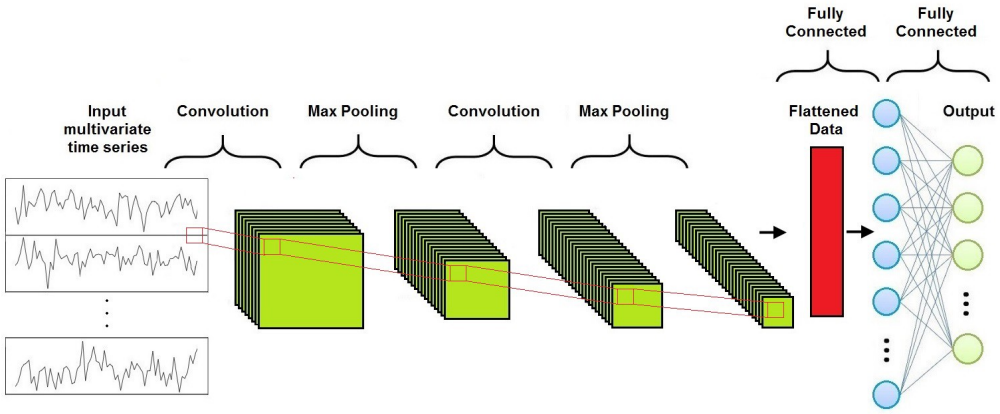


Figure 3.2: CNN architecture for a time series classification

on the whole series, are then given in input to a layer of Pooling. This phase aggregates the input through a sub-sampling; in fact, the Pooling, as well as the convolutional layer, acts through a kernel that scans the sub-portions of the original input, reducing it through the average operation (Average Pooling) or extracting the maximum values (Max Pooling). "Local pooling such as average or maximum pooling takes an input time series and reduces its length by aggregating over a sliding window of the time series. With global pooling, the time series will be aggregated over the entire resulting time dimension into a single true value. In other words, this is similar to applying local pooling with a sliding window length equal to the length of the input time series. Global pooling is usually adopted to drastically reduce the number of parameters in a model, thus decreasing the risk of overfitting."**[35]**
"In addition to pooling layers, some deep learning architectures include normalization layers to help the network converge quickly. For time series data, the batch normalization operation is performed on each channel, thus preventing internal shifting of covariates through a mini-batch of time series training."
Finally, downstream of the network, there is a Fully Connected layer that synthesizes the processing of the individual filters into a final result.
In the case of time series, convolution can be seen as the application of a filter that has only one dimension (time), as opposed to images that have two (width and height). It makes sense to consider the filter as a generic non-linear transformation of a time series. In practice, if we make a convolution between a filter of length 4 and a univariate series, "by setting the filter values to be equal to $[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$ the convolution will result in applying a moving average with a sliding window of length 4. A general form of applying the convolution for a centered time stamp t is given in the following equation:

$$C(t) = \gamma\Big(g * X(t - \frac{l}{2} : t + \frac{l}{2}) + b\Big)$$

where C denotes the result of a convolution applied on a univariate time series X of length T with a filter g of length l, a bias parameter b and a final non-linear function γ".**[35]** The result obtained by applying a filter to a univariate time series X is another filtered time series C.

The values assigned to the filter do not have to be chosen, but the optimal values established according to the input dataset should be learned. A filter is considered optimal when it allows an easy classification between the classes of the dataset.

Chapter 4

Data Analysis

In this chapter, we are going to tackle the problem of forecasting financial time series. The problem we are going to tackle is regression. This means that a continuous value is assigned to an input pattern.
Solving a regression problem corresponds to finding a function that approximates the input/output pairs provided during the training phase.
Data collection is the first step: the main characteristic they must have is to be collected at a regular, pre-established frequency.
The downloaded data was taken from the site *https://it.finance.yahoo.com/*: this is a global financial portal that provides analysis, streaming, quotations and graphs, technical data and instruments regarding global financial markets, and allows to visualize and obtain all the historical data relative to all the quotations of the securities present on the Stock Exchange. The stock taken as an example is Intesa San Paolo S.p.A., over the period from 1 September 2017 to 31 August 2021, with a daily frequency of detection.

4.1 Dataset description

First of all, we will analyze the main characteristics of the series taken into consideration, to build a structure and a model suitable for the type of starting date.
This step is extremely important because, depending on the composition of the data, they will fit differently to the constructed models.
The first aspect to consider is the definition of inputs and outputs. It is necessary to be clear about this aspect which is crucial to structure the model correctly. We stress that as input we do not mean the set used to train the model, but we mean the variables we want to analyze to carry out the prediction of the output. When we speak of univariate analysis for the input or output, we are indicating that a single variable is considered; whereas if the analysis is multivariate the number of variables is greater than one.
In the case in question, we will have as input a multivariate analysis with 7 elements and as output a single variable.

The input data are composed of the following variables:

- **Date** (d): reference date;

- **Open** (o): opening value on the corresponding date;
- **High** (h): highest value of the index on the corresponding date;
- **Low** (l): lowest value taken on the corresponding date;
- **Close** (c): index closing value on the corresponding date;
- **Adj Close** (a): close price after taking into account factors such as dividends, share splits or even the possible issue of new shares;
- **Volume** (v): volume traded on the corresponding date.

The input dataset is read from the csv file, which containing our data; then it's stored in a local dataframe and its main characteristics could be seen in the structure in Figure 4.1.

There are no null or missing values in the dataset, but we can immediately see

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-09-01	2.8500	2.8620	2.8420	2.8420	2.043724	61133251
1	2017-09-04	2.8240	2.8420	2.8180	2.8360	2.039409	36965318
2	2017-09-05	2.8340	2.8520	2.8020	2.8160	2.025027	59599727
3	2017-09-06	2.7960	2.8320	2.7880	2.8120	2.022151	66275247
4	2017-09-07	2.8220	2.8340	2.8000	2.8060	2.017836	91483360
...	...	...	...	...	...	...	...
1008	2021-08-25	2.3800	2.3900	2.3695	2.3900	2.390000	48254761
1009	2021-08-26	2.3780	2.3860	2.3590	2.3640	2.364000	73692888
1010	2021-08-27	2.3600	2.3800	2.3545	2.3780	2.378000	44028348
1011	2021-08-30	2.3825	2.3840	2.3620	2.3690	2.369000	31114337
1012	2021-08-31	2.3720	2.4015	2.3625	2.3975	2.397500	92717758

Figure 4.1: The dataset contains 1013 rows and 7 columns

that the four years under analysis include only weekly data.
So the input at day t is generically denoted as:

$$\mathbf{x}_t = (d_t, o_t, h_t, l_t, c_t, a_t, v_t)$$

which is a vector of seven elements, which for simplicity we will denote as x_t.
The input information set will be used to detect the evolution of the output variable, identified in the daily closing value of the indices. Considering as input the data collected for t previous days, the objective of this analysis is to obtain the closing value c_i on a forecasting horizon of a single day. Therefore, we want to obtain the value c_{t+1}.
Assuming n trading days are considered, the input-output table of the model will be as follows:

$$
\begin{aligned}
\textit{Input} &\Rightarrow \textit{Output} \\
(x_0, x_1, ..., x_t) &\Rightarrow c_{t+1} \\
(x_1, x_2, ..., x_{t+1}) &\Rightarrow c_{t+2} \\
&\dots \\
(x_{n-t}, x_{n-t+1}, ..., x_{n-1}) &\Rightarrow c_n
\end{aligned}
$$

During a data analysis, it is important to ensure that you use the correct data types, so let's check the form of our variables in Figure 4.2.
The Date variable is used for date and time values, whereas the Pandas dtypes

```
Date          datetime64[ns]
Open                 float64
High                 float64
Low                  float64
Close                float64
Adj Close            float64
Volume                 int64
dtype: object
```

Figure 4.2: Data type of each object

float64 and int64 indicate floating point numbers and integer numbers respectively. The following statistical measures can be seen for each column using the describe-function of DataFrame of the pandas library:

- count: number of samples
- mean: the mean of this attribute among all samples
- std: the standard deviation of this attribute
- min: the minimal value of this attribute
- 25%: the lower percentile
- 50%: the median
- 75%: the upper percentile
- max: the maximal value of this attribute

From Figure 4.3 we can immediately see that the data are not standardized. Furthermore, it is interesting to note that the maximum values of high and low are very close to each other, as are the minimum values. This behavior is easily explained by the fact that the price of this stock does not undergo sudden changes. As a result, the high and low in a day are usually not very far apart.

According to the following code, it is possible to obtain a dataset showing the monthly average of the opening and closing stock prices.

```
1 df['Date'] = pd.to_datetime(df['Date'])
2 monthly_mean= data.groupby(df['Date'].dt.strftime('%B'))
3         [['Open','Close']].mean()
4 months = ['January', 'February', 'March', 'April', 'May', 'June',
```

	Open	High	Low	Close	Adj Close	Volume
count	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1.013000e+03
mean	2.232518	2.256072	2.206338	2.231057	1.874188	1.154766e+08
std	0.441718	0.440379	0.442272	0.442093	0.295098	5.530659e+07
min	1.337800	1.350800	1.306200	1.320000	1.119756	3.111434e+07
25%	1.928600	1.952000	1.906000	1.926400	1.653295	7.829352e+07
50%	2.208000	2.233000	2.178500	2.205500	1.881988	1.008558e+08
75%	2.445000	2.460000	2.416500	2.445500	2.096938	1.350347e+08
max	3.201000	3.230000	3.161500	3.210000	2.485500	4.517761e+08

Figure 4.3: Description of the variables

```
5          'July', 'August', 'September', 'October', 'November', 'December']
6 monthly_mean = monthly_mean.reindex(months, axis=0)
```

Date	Open	Close
January	2.315755	2.318967
February	2.408995	2.412333
March	2.276071	2.270738
April	2.258809	2.257743
May	2.227299	2.219412
June	2.124051	2.122731
July	2.139157	2.138670
August	2.113793	2.109257
September	2.262751	2.264898
October	2.177797	2.172599
November	2.234676	2.238222
December	2.277207	2.273493

Figure 4.4: Average opening and closing stock prices

From the Figure 4.4 it's possible to observe that the average monthly opening and closing values do not change over time.
Instead, in Figure 4.5 we plot the stock high and low prices, grouped by month. From this graph, however, it is not possible to see the variation and seasonality of prices by month. It is necessary to analyze these data through other methodologies.

Through the OHLC graph, we can analyze more specifically the behavior of the stock price.
The OHLC chart, whose acronym stands for open, high, low, and close, in Figure 4.6, describes for each date d the values of open, high, low, and close. It is possible to distinguish positive trends, shown in green, from negative trends, shown in red. Each daily block is made up of a minimum and a maximum value: if the open value is greater than the close value, the sample point is said to be decreasing, while if during the day the value increases, we will have an increasing trend.
Through the plotly library, it is possible to view the index trend over time,

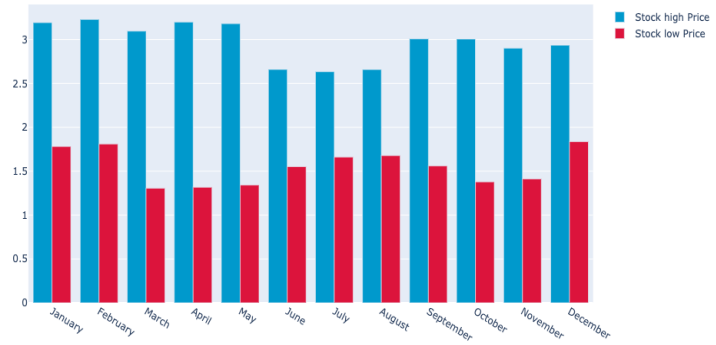


Figure 4.5: Stock high and low prices, by month

observing maximum, minimum, opening, and closing levels.

```
import plotly.graph_objects as go

fig = go.Figure(data=go.Ohlc(x=df['Date'],
                      open=df['Open'],
                      high=df['High'],
                      low=df['Low'],
                      close=df['Close']))

fig.update(layout_xaxis_rangeslider_visible=False)
fig.show()
```

The result obtained through this code is shown in Figure 4.6.
From the figure, it is clear that there were two moments when the share price collapsed: the first in October 2018, due to a convergence of negative factors in the Italian stock market, and the second in March 2020, caused by the Covid-19 outbreak. Before this date, Intesa shares were performing very well, where climbing and had taken on a more than positive trend. A more gradual negative trend is in correspondence with the second Italian wave of Covid, which occurred in the summer of 2020.

Let's take a closer look to the graph in Figure 4.7. The vertical height of a bar indicates the volatility of the index during the day: the greater the length of the bar, the more volatile the value of the asset and therefore unstable in the market. The whole length of the bar coincides with the difference between the maximum and minimum values reached during the day.
The horizontal bars indicate the values of opening and closing of the Stock: when the two points are close, it means that the price has made notable progress neither in one direction nor in the other. This behavior also shows indecision in the asset's behavior.
As you can tell from what has been said, if there are more green bars instead of red, then the trend will typically be bullish. A series of many green bars

Figure 4.6: OHLC

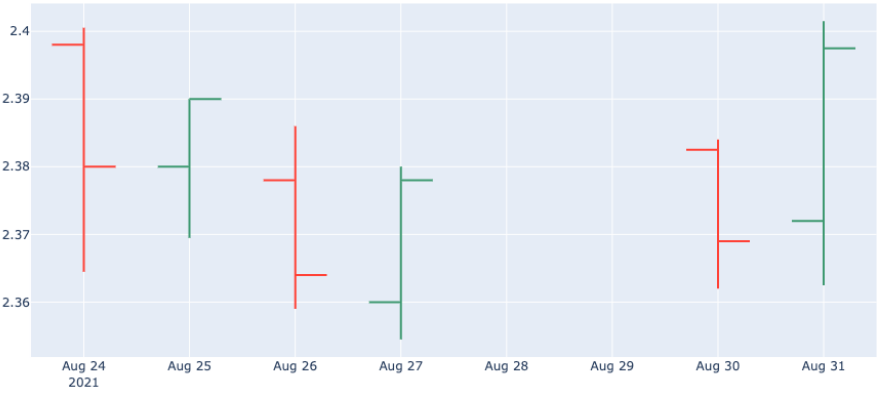


Figure 4.7: Closer look to an OHLC chart

indicate a positive trend. [**5**]

4.1.1 Normalization

As in most machine learning problems, it is often necessary to use normalization methods in deep learning problems. This is due to the need to compare units of measurement that are different from each other.
For this purpose, there are data scaling techniques, which transform the data into a more comprehensible format and such that they are on the same level of importance. In addition, normalization makes the data more suitable for con-

vergence and comparison, speeds up the learning process, which is therefore for trend prediction.
This step is essential to avoid errors, which would make the model unsuitable for good prediction.

There are many methods of normalization. The one we will use is called *Min-Max Scaling*. The formula that allows this scaling is:

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

In the case under analysis it has been used because it ensures that all features have the exact same scaling and the data belong to a range of oscillation that does not contain outliers. It also handles outliers correctly.
The data is scaled to a fixed interval, usually from 0 to 1.

```
data= df.values[:, 1:5]
y= data[1:, 3]
X= data[:1012,:]

y= (y- min(y))/(max(y)- min(y))
X= (X- np.min(X,axis=0))/(np.max(X, axis=0)-np.min(X,axis=0))
```

X is an array that contains the first 1012 vectors, excluding the last one, which is made up of the first 5 column values of the dataset. Instead, y contains the last 1012 closing values of the dataset. These arrays are subsequently standardized. The values are then reproportioned and, considering the entire dataset, the minimum value will take on the value 0, while to the maximum one it will be assigned the value 1.

4.1.2 Training and validation set

The transformed dataset must be divided into training sets and validation sets. The training set is the first part of the time series, which will be used to train the model. It is used to allow the model to set the network parameters.
Therefore, the training set will be submitted to the model with both the inputs and the corresponding outputs.
The validation set, or test set, is composed of a part of data that the model has never analyzed. This data will only feed the model as input, so that we can compare the output, i.e. the expected values, with the actual values we know about. In this way, it will be possible to evaluate the effectiveness of the model.

As we have previously introduced, supervised learning is an algorithm that can be used as soon as you have a training set of data. It is therefore necessary to have an example of the input-output function that we wish to map, with a set of input samples and corresponding outputs.
After this learning/training phase, the samples excluded from the analysis are sent to the model and a prediction is made based on the learned mapping. The network is then trained by a back propagation algorithm, which uses the sample data to set the network weight and bias appropriately, minimizing the prediction error.

These steps allow us to transform a historical series into a supervised learning problem, and thus allow us to solve prediction problems such as the one under analysis.
Supervised learning assumes that one wants to map a function. The problem with time series is that they are not in the form of a function. A time series is characterized by a precise temporal dependence, i.e. the observations that compose it have an inescapable order. Therefore the sequence in which the values follow each other is fundamental and it is not possible to mix the observations, otherwise one would obtain a completely different and meaningless series.
The most common way to get around this obstacle is to use a *Sliding Windows Algorithm*: the aim is to transform the dataset in such a way that, even if mixed, it represents our historical series equally well.
This method allows us to reorganize the starting dataset to make it suitable for supervised training. The basic principle behind this algorithm is scrolling through the whole time series.
For this reason, it's not possible to split the dataset randomly, because the sequence of events is important for time series.
So let's start taking 70% values for the training part and the remaining one for testing.

```
training_size=int(len(X)*0.70)
test_size=len(X)-training_size
train_X, test_X=X[0:training_size,:],X[training_size:len(X),:1]
```

In the same way, placing the array as a one-dimensional array, it works also on y.
The respective dimensions of $train\_X$, $test\_X$, $train\_y$, $test\_y$ are:

```
(708, 4), (304, 1)
(708,), (304,)
```

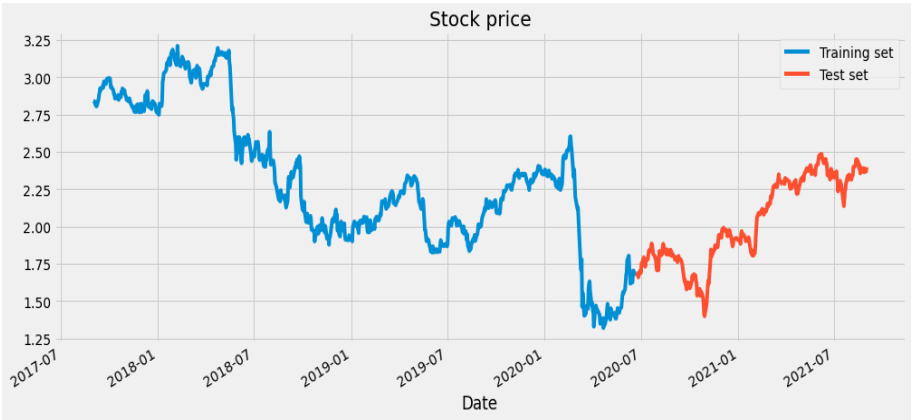


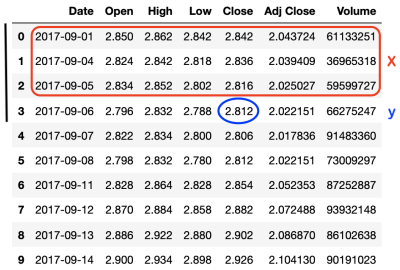
Figure 4.8: Close price divided into training and test sets

In this way, in the case where the task is to predict the next value in the future, the new dataset will be composed of the data contained in all the windows, which will represent our input, with the corresponding observation just after, i.e. the output.

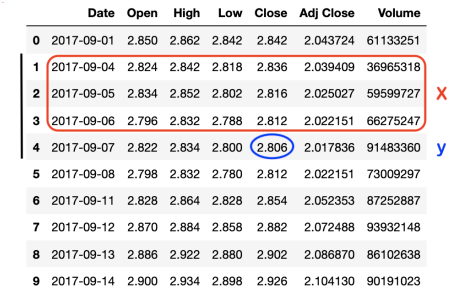
Once the model is built, if we wanted to predict the next value of a time series, it would be sufficient to provide the model with the content of the last observed time window (i.e. the one whose last observation represents the current value) to obtain the prediction.

For example, suppose we take a sliding window with numerosity of 4:

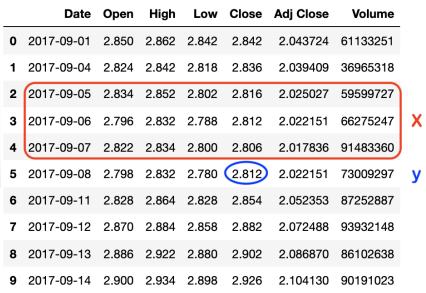
- 3 input vectors, which contains all the variables at time t;
- 1 output value, which is the closing value at time $t + 1$.



	Date	Open	High	Low	Close	Adj Close	Volume	
0	2017-09-01	2.850	2.862	2.842	2.842	2.043724	61133251	
1	2017-09-04	2.824	2.842	2.818	2.836	2.039409	36965318	X
2	2017-09-05	2.834	2.852	2.802	2.816	2.025027	59599727	
3	2017-09-06	2.796	2.832	2.788	2.812	2.022151	66275247	y
4	2017-09-07	2.822	2.834	2.800	2.806	2.017836	91483360	
5	2017-09-08	2.798	2.832	2.780	2.812	2.022151	73009297	
6	2017-09-11	2.828	2.864	2.828	2.854	2.052353	87252887	
7	2017-09-12	2.870	2.884	2.858	2.882	2.072488	93932148	
8	2017-09-13	2.886	2.922	2.880	2.902	2.086870	86102638	
9	2017-09-14	2.900	2.934	2.898	2.926	2.104130	90191023	



	Date	Open	High	Low	Close	Adj Close	Volume	
0	2017-09-01	2.850	2.862	2.842	2.842	2.043724	61133251	
1	2017-09-04	2.824	2.842	2.818	2.836	2.039409	36965318	
2	2017-09-05	2.834	2.852	2.802	2.816	2.025027	59599727	X
3	2017-09-06	2.796	2.832	2.788	2.812	2.022151	66275247	
4	2017-09-07	2.822	2.834	2.800	2.806	2.017836	91483360	y
5	2017-09-08	2.798	2.832	2.780	2.812	2.022151	73009297	
6	2017-09-11	2.828	2.864	2.828	2.854	2.052353	87252887	
7	2017-09-12	2.870	2.884	2.858	2.882	2.072488	93932148	
8	2017-09-13	2.886	2.922	2.880	2.902	2.086870	86102638	
9	2017-09-14	2.900	2.934	2.898	2.926	2.104130	90191023	



	Date	Open	High	Low	Close	Adj Close	Volume	
0	2017-09-01	2.850	2.862	2.842	2.842	2.043724	61133251	
1	2017-09-04	2.824	2.842	2.818	2.836	2.039409	36965318	
2	2017-09-05	2.834	2.852	2.802	2.816	2.025027	59599727	
3	2017-09-06	2.796	2.832	2.788	2.812	2.022151	66275247	X
4	2017-09-07	2.822	2.834	2.800	2.806	2.017836	91483360	
5	2017-09-08	2.798	2.832	2.780	2.812	2.022151	73009297	y
6	2017-09-11	2.828	2.864	2.828	2.854	2.052353	87252887	
7	2017-09-12	2.870	2.884	2.858	2.882	2.072488	93932148	
8	2017-09-13	2.886	2.922	2.880	2.902	2.086870	86102638	
9	2017-09-14	2.900	2.934	2.898	2.926	2.104130	90191023	

In the first figure, the first iteration takes place: in this case we call X the input matrix composed of 3 input vectors (x_0, x_1, x_2) and y the output value c_3. During the second iteration $X = (x_1, x_2, x_3)$ and $y = c_4$, wherease at the third iteration $X = (x_2, x_3, x_4)$ and $y = c_5$.
This same procedure is repeated for the entire length of the training set. So, for example, at the n-th iteration we will have $X = (x_{n-1}, x_n, x_{n+1})$ and $y = c_{n+2}$.

The implementation of the *sliding_windows* algorithm is constructed using the following Python function:

```
def sliding_windows(X, y, time_step):
    X_new= np.zeros((X.shape[0] - time_steps +1, 20, X.shape[1]))
    y_new= np.zeros((y.shape[0] - time_steps +1,))

    for i in range(X_new.shape[0]):
        for j in range(time_steps):
            X_new[i, j, :]= X[i + j, :]
        y_new[i]= y[i + time_steps -1]

    return(X_new,y_new)
```

X and y will become an array of input vectors and the corresponding output, respectively, and will be converted into arrays.
Next, all you have to do is call the function on the training set and the test set in the main program, specifying the number of steps you want as input.

```
time_step=20
X_train, y_train = sliding_windows(train_X, train_y, time_step)
X_test, y_test = sliding_windows(test_X, test_y, time_step)
```

We take a window of size 20, which are the monthly working days.

4.2 Implementation of the Neural Networks

In this section, the different models analyzed in chapter 3 will be implemented. The construction of each of them will be explored.
First of all, it is necessary to introduce the Numpy library which is a well-known mathematical library that helps us in the vector and matrix problems typical of these application contexts.
At the base of the construction of all the structures of Neural Networks that we will use is Keras. It is an open-source library that is used to build Deep Learning models in Python, usually for supervised learning problems. As already mentioned, Keras is a back-end technology, supporting the Tensorflow library, which belongs to an even deeper level than Keras.
Keras is a simple, intuitive library that is easy to implement. It allows you to create complex models while keeping them robust and at the same time allows great flexibility due to its modular feature. "In fact, in Keras, the concept of a model is understood as a sequence or graph of individual modules that can be assembled to create new models with as few restrictions as possible. Neural Networks, cost functions, optimizers, initialization schemes, activation functions, regularisation schemes are all seen in Keras as modules and can be combined to create new ones."[**40**]

The most commonly used model in Keras is the sequential model, which implements a linear stack of Neural Network layers, but there is also the possibility to use the functional model.
We now introduce the main models that will be used during the analysis:

- **Sequential()**, which is the tool that allows us to insert the layers of a Neural Network, specifying each time the type of layer, whether input, hidden, or output. The sequential model is in fact a linear stack of layers, and to add layers via the .add() method.

- **Dense(n_perceptrons, activation_function)**, is a class that constitutes a layer composed of n neurons; it constitutes the classic layer of an artificial Neural Network in which the inputs are weighted and transferred through the activation function to the output together with the bias. The parameters to specify are respectively the number of perceptrons that compose the layer, and the function of activation.

- **Dropout(rate)** is a layer that is used to prevent the overfitting in the Neural Networks: it sets randomly a fraction of units of input on 0 at each update during the time of training. This simplifies the Neural Network and reduces training time.

- **Flatten()** converts multidimensional data into a single vector for processing. It is a layer used between the convolutional layer and the FullyConnected layer that aims to reduce the output features to a one-dimensional vector. For example, a 28 by 28 matrix would become a vector 784 numbers long after the application of the Flatten layer.

- **MaxPooling1D(pool_size, strides)** is the process of splitting the data into several regions, which extracts the maximum values of the characteristics of a cell and places them in an output matrix. This is done to reduce the amount of information to be processed and sent to the next layer. The input parameters are the kernel size of the layer (pool_size parameter) and the integer number representing the factor by which the downscaling (strides) is to be performed.

4.2.1 LSTM

Let's start with the Long Short-Term Memory (LSTM) network.
The implementation of the LSTM is characterized by two phases: the first is the application of different weight matrices to the input and hidden layers; these are then passed to four Neural Networks within the LSTM cell. The 4 matrices that are multiplied to the input are called kernels, while those multiplied with the hidden state are called recurrent_kernel.

As input we have the training dataset X that is reshaped in the format $[samples, time\_steps, features]$, to obtain an output y. By samples, we mean a fixed number of observations that are independent of the domain, i.e. the rows of data. "This batch of data defines how many patterns to process before updating the network weights. Time steps, on the other hand, are separate time steps of a given variable for a given observation. Features, on the other hand, are separate measurements observed at the time of the observation.
Thus each time step in the original sequence is a separate sample with a time step and a feature."[**26**]

The code that has been used to implement an LSTM-type network starts with the definition of X_train and X_test. The actual sequential model consists of the following steps:

```
import keras
from keras.models import Sequential
```

```
from keras.layers import LSTM, Dense, Dropout

LSTM_model = Sequential()
LSTM_model.add(LSTM(50, return_sequences=True,
            input_shape=(X_train.shape[1],X_train.shape[2],1)))
LSTM_model.add(Dropout(0.2))
LSTM_model.add(LSTM(50, return_sequences=True))
LSTM_model.add(Dropout(0.2))
LSTM_model.add(LSTM(50))
LSTM_model.add(Dropout(0.2))
LSTM_model.add(Dense(1))
```

The model expressed through the above code is a sequential type model to which several layers are added: the first one is an LSTM layer that has the following characteristics:

- $units = 50$ neurons, which are added and coincide with the dimensionality of the output space
- $return\_sequences$ = True, to stack the LSTM layers so that the next LSTM layer has a three-dimensional input sequence
- $input\_shape$ is the shape of the training dataset

By adding the Dropout layer, the 20% of layers will be dropped out.
The second and third LSTM layers are analogous to the first one, but it is not necessary to specify the shape of the input data. Also in these cases, 20% of the elements are dropped.
Finally, the last layer is the dense one that specifies output of one unit.

```
print(LSTM_model.summary())


Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 20, 50)            10400
_________________________________________________________________
dropout_9 (Dropout)          (None, 20, 50)            0
_________________________________________________________________
lstm_10 (LSTM)               (None, 20, 50)            20200
_________________________________________________________________
dropout_10 (Dropout)         (None, 20, 50)            0
_________________________________________________________________
lstm_11 (LSTM)               (None, 50)                20200
_________________________________________________________________
dropout_11 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
```

```
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
None
```

Using the following code, it is possible to extract all weights from the model:

```
for x in LSTM_model.layers[0].weights:
    print(x.name, '-->', x.shape)
```

whose output is

```
lstm/lstm_cell/kernel:0 --> (1, 200)
lstm/lstm_cell/recurrent_kernel:0 --> (50, 200)
lstm/lstm_cell/bias:0 --> (200,)
```

It's possible to see that the 4 kernel matrices and the 4 recurrent_kernel matrices are stored in 1 single monolithic matrix each (concatenated on the column axis), so the size is 50*4=200. The same applies to the bias term.
It is also important to specify the difference between the *recurrent_activation* and *activation* parameters: "the default value of the recurrent_activation parameter has a value of "sigmoid" and is applied to the input, forget and output gate as suggested above in the formula; activation, on the other hand, has a default value of "tanh" and is applied to the cell state and hidden state".[**39**]

In the previous lines, we have defined the type of model of the Neural Network; at this point, it is necessary to understand if it is suitable for the training of the network.
For this purpose, we will use the function compile on the sequential model defined before and we will use it as optimizer 'adam', an algorithm that is an extension of the stochastic gradient descent.
As a metric to evaluate the quality of the training we will use the MSE: the lower the value, the better the training; accuracy is not used as an evaluation metric because our case is not a classification problem, but a regression problem.
Furthermore, we will also use 'mean_squared_error' for the loss function, which is used in regression cases.

```
LSTM_model.compile(optimizer='adam', loss='mean_squared_error',
                   metrics=['mean_squared_error'])
```

At this point the structure of the Neural Network is ready and it is sufficient to launch the learning phase. During the training phase of the model, the patterns contained in the training set are provided, so that the model can adapt its parameters in search of the optimal objective function. However, one single step is not sufficient to obtain an optimal result; several cycles of training set processing and parameter adaptation are required.
Among other things, due to the limits imposed by the memory and the computing power available for such processing, it is usually not possible to process the entire training set in one go.
When the entire training set has been submitted to the model, then there is an epoch. The training phase usually ends after several epochs. But not always by increasing the number of epochs does the model improve.
As mentioned above, the training set may be too large to process all at once.

Therefore, the training set can be divided into uniform subgroups, called batches.
The number of examples contained in each batch is called batch size.
Hence the definition of iteration, which is the number of batches needed to complete an epoch.
The number of epochs and the batch size affect the training speed of a model, but also the way it is refined.
If the batch size is too small, usually less than 10, then performance will not be optimized: if the batch size is small, there will be too little data to process and the benefits of the processor architecture will not be fully exploited.
Conversely, if the batch size is too large, there may be a problem with memory exhaustion or a more pronounced tendency to overfit.
Batch sizes are usually 32, 64, or 128, powers of 2, for memory allocation reasons.[**32**]

In our case, this phase stops the training after 30 epochs with batches of 64 elements at a time. LSTM_model.fit() "fits training data. For supervised learning applications, this accepts two arguments: the data X_train and the labels y_train".

```
history=LSTM_model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    verbose=1, epochs=30, batch_size=64)
```

```
Epoch 1/30
11/11 [==============================] - 0s 31ms/step - loss: 0.0042 -
        mean_squared_error: 0.0042 - val_loss: 0.0014 -
        val_mean_squared_error: 0.0014
Epoch 2/30
11/11 [==============================] - 0s 24ms/step - loss: 0.0038 -
        mean_squared_error: 0.0038 - val_loss: 0.0015 -
        val_mean_squared_error: 0.0015
Epoch 3/30
11/11 [==============================] - 0s 23ms/step - loss: 0.0044 -
        mean_squared_error: 0.0044 - val_loss: 0.0013 -
        val_mean_squared_error: 0.0013
Epoch 4/30
11/11 [==============================] - 0s 24ms/step - loss: 0.0039 -
        mean_squared_error: 0.0039 - val_loss: 0.0020 -
        val_mean_squared_error: 0.0020
Epoch 5/30
11/11 [==============================] - 0s 24ms/step - loss: 0.0041 -
        mean_squared_error: 0.0041 - val_loss: 0.0012 -
        val_mean_squared_error: 0.0012
....
Epoch 26/30
11/11 [==============================] - 0s 23ms/step - loss: 0.0037 -
        mean_squared_error: 0.0037 - val_loss: 0.0011 -
        val_mean_squared_error: 0.0011
Epoch 27/30
11/11 [==============================] - 0s 23ms/step - loss: 0.0037 -
        mean_squared_error: 0.0037 - val_loss: 0.0011 -
```

```
        val_mean_squared_error: 0.0011
Epoch 28/30
11/11 [==============================] - 0s 23ms/step - loss: 0.0037 -
        mean_squared_error: 0.0037 - val_loss: 0.0011 -
        val_mean_squared_error: 0.0011
Epoch 29/30
11/11 [==============================] - 0s 23ms/step - loss: 0.0034 -
        mean_squared_error: 0.0034 - val_loss: 0.0011 -
        val_mean_squared_error: 0.0011
Epoch 30/30
11/11 [==============================] - 0s 23ms/step - loss: 0.0036 -
        mean_squared_error: 0.0036 - val_loss: 0.0012 -
        val_mean_squared_error: 0.0012
```

Following these steps, output predictions can be generated for the input samples. Given a trained model, it is then possible to predict a new data set.

```
train_predict_LSTM=LSTM_model.predict(X_train)
test_predict_LSTM=LSTM_model.predict(X_test)
```

However, the values predicted through these formulas are standardised, as we had as input arrays that had been transformed through the $MinMaxScaler$ function.
It is therefore sufficient to apply the inverse transformation to the results obtained, through the following code:

```
train_predict_LSTM=scaler.inverse_transform(train_predict_LSTM)
test_predict_LSTM=scaler.inverse_transform(test_predict_LSTM)
```

Finally, it is possible to plot the predicted data and compare it with the source data to have a graphical understanding of how the model works.

```
seq_size=20

trainPredictPlot = np.empty_like(df['Close'])
trainPredictPlot[:,:] = np.nan
trainPredictPlot[seq_size:len(train_predict_LSTM)+seq_size,:] =
                    train_predict_LSTM

testPredictPlot = np.empty_like(df['Close'])
testPredictPlot[:,:] = np.nan
testPredictPlot[len(train_predict_LSTM)+(seq_size*2)+1:len(df['Close'])-1,:] =
                    test_predict_LSTM

plt.figure(figsize=(16,8))
plt.plot(scaler.inverse_transform(df['Close']))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.legend(['Original','Train','Test'])
plt.title('LSTM plot')
plt.show()
```

In the first part of the code we shifted the train and test predictions for plotting, to plot baseline and predictions.

In blue is the trend of the closing price of the share overtime of the starting

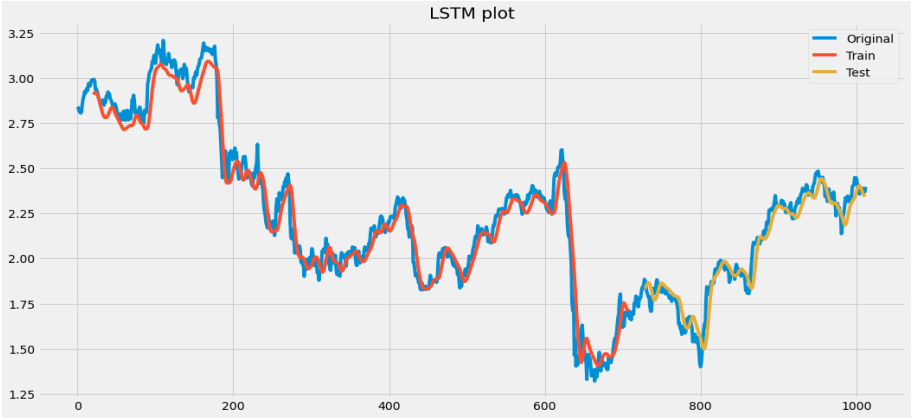


Figure 4.9: Plot shows how well the trend has been tracked by the prediction

dataset; it is plotted using the code on line 14.
In red is shown the prediction made on the training set X_train; note that in correspondence of the first twenty days of the series no predictions are made, because we have taken a time step equal to 20.
The prediction of the test set is shown in yellow and, as for the red line, starts from the twenty-first day.

It can be seen that the prediction made approximates well the original dataset. In the following chapter, we will show the results of the metrics applied to the models, to understand the goodness and accuracy of each of them.

GRU

In this section, we quickly see the behavior of the GRU model which, like the LSTM model, is a Recurrent Neural Network.
The steps will not be shown specifically as they are similar. They are two techniques that optimize the vanishing gradient problem, but what changes is how they "use previous information with gates to prevent vanishing gradients".[**41**]
The GRU model has a less variable gradient descent and is also faster as it has fewer parameters. Therefore, in general, the GRU model is considered to be a model with a simpler and leaner structure.

The Keras model that allows the GRU architecture to be built in the following code:

```
1 GRU_model=Sequential()
2 GRU_model.add(GRU(32,return_sequences=True,input_shape=(time_step,1)))
3 GRU_model.add(GRU(32,return_sequences=True))
4 GRU_model.add(GRU(32))
5 GRU_model.add(Dropout(0.20))
6 GRU_model.add(Dense(1))
7 GRU_model.compile(loss='mean_squared_error',optimizer='adam')
```

In the next chapter, we will compare all the models analyzed with each other to see which is the most efficient for our regression problem.

4.2.2 CNN

As previously explained the Convolutional Neural Networks was born for the elaboration of images or, more in general, of two-dimensional files. To exploit it on a historical series it is necessary to modify the convolutional hidden layer so that it can work on a mono-dimensional input.
What is instead necessary to operate on the data is a modification of the dimensionality of the input. In fact, in addition to the number of samples and their length (number of timesteps), that by construction are information already incorporated in the data (length and width of the input sequence), we need to incorporate further information on the number of characteristics, that is, on the number of input variables, being this a multivariate analysis. This is because it is necessary to send the number of features as a parameter to the convolutional layer.

In this specific case, we start from input data of multivariate type, unlike the LSTM case: in fact, the variables considered are Open, High, Low, Close at time t. While the single output variable is Close, at the time instant $t + 1$. These constitute the X and y arrays respectively, to which the normalization and then the sliding_windows algorithm is applied.
It is necessary to reshape X and y to have a format suitable for the convolutional model.
Again, we assign the first 70% to the training set and the remaining 30% to the test set, obtaining the following dimensions.

```
(709, 20, 4) (709,)
(284, 20, 4) (284,)
```

Also in this case we use the method Sequential(), constructor of an instance of the Neural Network. Then, with the add method of the class Sequential class, new layers can be added.
The model that we are going to implement is composed of two convolutional layers followed by a pooling layer, followed by a further convolutional layer, followed by a further pooling layer; downstream of the network, we insert the flatten layer for the reduction of the features, before two Fully Connected layers that will produce the final result. The lines of code to implement the structure are as follows:

```
Conv_model = Sequential()
Conv_model.add(Conv1D(filters=64, kernel_size=3,
                  activation='relu', input_shape=(seq_size, 4)))
Conv_model.add(Conv1D(filters=32, kernel_size=3,activation='relu'))
Conv_model.add(MaxPooling1D(pool_size=2))
Conv_model.add(Flatten())
Conv_model.add(Dense(32))
Conv_model.add(Dense(1))
```

The Conv1D model is effective for processing temporal patterns (unlike Conv2D which processes visual patterns). This type of model can be more efficient and

faster than Recurrent Neural Networks, such as LSTM. This is because they are much less expensive to process long sequences, as they shorten the sequence and extract useful information.
The structure of this sequential model consists of stacks of layers in which Conv1D and MaxPooling1D are superimposed.
Specifically, the function of activation used is the ReLU function, which accepts any real value as its input, but is activated only when these inputs are greater than 0.

Once the model has been built, the compile method must be invoked on it, whose optimizer and loss parameters allow the algorithm to be specified to reduce the loss at each iteration and the loss function to be used for reduction. The summary obtained with this model is the following:

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_4 (Conv1D)            (None, 18, 64)            832
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 16, 32)            6176
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 32)             0
_________________________________________________________________
flatten_2 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 32)                8224
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 33
=================================================================
Total params: 15,265
Trainable params: 15,265
Non-trainable params: 0
_________________________________________________________________
None
```

Once the template has been filled in, we are ready to start the training phase:

```
Conv_model.fit(X_train, y_train, validation_data=(X_test, y_test),
        verbose=1, epochs=30, batch_size=32)
```

The fit method will initiate the training and validation phases of the model using the following parameters:

- X_train: training set of 709 elements;
- y_train: training set of 709 labels;
- *epochs*: number of iterations, 30 in our example;
- $batch\_size$= size of the mini-batch of examples used for the gradient calculation;

- *validation_data*: validation set for model evaluation consisting of 284 elements and 284 labels.

```
Epoch 1/30
23/23 [==============================] - 1s 18ms/step - loss: 0.0859 -
val_loss: 0.0049
Epoch 2/30
23/23 [==============================] - 0s 5ms/step - loss: 0.0063 -
val_loss: 0.0028
Epoch 3/30
23/23 [==============================] - 0s 4ms/step - loss: 0.0047 -
val_loss: 0.0020
Epoch 4/30
23/23 [==============================] - 0s 4ms/step - loss: 0.0040 -
val_loss: 0.0014
Epoch 5/30
23/23 [==============================] - 0s 4ms/step - loss: 0.0027 -
val_loss: 0.0012
...
Epoch 26/30
23/23 [==============================] - 0s 3ms/step - loss: 9.1005e-04 -
val_loss: 5.9754e-04
Epoch 27/30
23/23 [==============================] - 0s 3ms/step - loss: 7.7060e-04 -
val_loss: 6.1791e-04
Epoch 28/30
23/23 [==============================] - 0s 3ms/step - loss: 8.3587e-04 -
val_loss: 7.5165e-04
Epoch 29/30
23/23 [==============================] - 0s 3ms/step - loss: 9.3447e-04 -
val_loss: 5.6893e-04
Epoch 30/30
23/23 [==============================] - 0s 3ms/step - loss: 7.6127e-04 -
val_loss: 0.0013
```

As in the case of the LSTM model, following the fitting of the model, the output values y_pred_Conv and $y\_pred\_Conv\_train$ are predicted: the .predict() function is applied to each object in the arrays (X_test and X_train respectively).
Even with the CNN model, it is necessary to transform the data so that it is no longer in the standardized format.
Now view it's possible to see the graph which shows us the difference between the original data and the data predicted through the convolutional model just described.

```
trainPredictPlot = np.empty_like(df['Close'])
trainPredictPlot[:] = np.nan
trainPredictPlot[seq_size:len(train_predict_Conv)+seq_size] =
        train_predict_Conv

testPredictPlot = np.empty_like(df['Close'])
```

```
testPredictPlot[:] = np.nan
testPredictPlot[len(train_predict_Conv)+(seq_size):len(df['Close'])] =
        test_predict_Conv

plt.figure(figsize=(16,8))
plt.plot(df['Close'])
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.legend(['Original','Train','Test'])
plt.title('Convolutional plot')
plt.show()
```

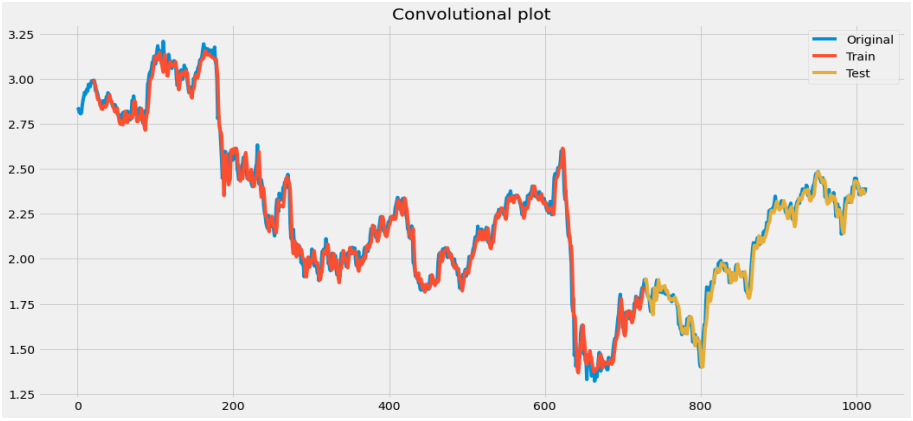


Figure 4.10: Plot shows how well the trend has been tracked by the prediction

4.2.3 MLP

The Multilayer Perceptron model we are going to analyze can, like CNN, have as input multivariate time series data. These are data that, in our case study, have 4 observations for each time step.
First of all, we have to set up the data for the regression.
Let's recall that the starting dataset contains 1013 rows:

- X is an array composed of the first 1012 elements (i.e. the last data collected is excluded), each of which is characterized by the 4 variables (Open, Close, High, Low);
- y is made up of the last 1012 elements (i.e. the first data collected is excluded), each of which has a size of 1 (variable Close).

Both arrays are normalized, as was done previously for the other models, and then split into a training set and testing set, before applying any algorithm.

```
from sklearn.neural_network import MLPRegressor

MLP_model= MLPRegressor(hidden_layer_sizes=(100, 50),
        activation='relu', solver='adam', verbose=True)
```

The parameters of the MLPRegressor function are:

- *hidden_layer_sizes*: "the i-th element represents the number of neurons in the i-th hidden layer;
- *activation*: the activation function is 'relu', the linear rectified unit function.
- *solver*: the solver for weight optimization: 'adam' refers, as we mentioned earlier, to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba. The default 'adam' solver works quite well on relatively large datasets (with thousands of training samples, but also more) in terms of training time and validation score."

```
MLP_model.fit(X_train, y_train)
```

```
Iteration 1, loss = 0.06953657
Iteration 2, loss = 0.02576766
Iteration 3, loss = 0.01744884
Iteration 4, loss = 0.01323221
Iteration 5, loss = 0.00610900
Iteration 6, loss = 0.00316600
Iteration 7, loss = 0.00247482
Iteration 8, loss = 0.00070633
Iteration 9, loss = 0.00066623
Iteration 10, loss = 0.00087667
Iteration 11, loss = 0.00066509
Iteration 12, loss = 0.00070267
Iteration 13, loss = 0.00045562
Iteration 14, loss = 0.00037360
Iteration 15, loss = 0.00034893
Iteration 16, loss = 0.00032559
Iteration 17, loss = 0.00034953
Iteration 18, loss = 0.00033008
Iteration 19, loss = 0.00032526
Iteration 20, loss = 0.00031213
Iteration 21, loss = 0.00030762
Iteration 22, loss = 0.00030341
Iteration 23, loss = 0.00030312
Iteration 24, loss = 0.00030152
Training loss did not improve more than tol=0.000100
        for 10 consecutive epochs. Stopping.
```

Here, after the prediction of the arrays and the transformation of the data into the original format (using the function *.inverse_transform*), we have also to reshape the data in the correct form. This is important to plot the prediction on the original data, using the following code:

```
train_predict_MLP = train_predict_MLP.reshape(train_predict_MLP.shape[0])
test_predict_MLP = test_predict_MLP.reshape(test_predict_MLP.shape[0])

trainPredictPlot = np.empty_like(df['Close'])
```

```
trainPredictPlot[:] = np.nan
trainPredictPlot[seq_size-20:len(train_predict_Conv)+seq_size-20] =
        train_predict_MLP

testPredictPlot = np.empty_like(df['Close'])
testPredictPlot[:] = np.nan
testPredictPlot[len(train_predict_Conv)+1:len(df['Close'])] =
        test_predict_MLP

plt.figure(figsize=(16,8))
plt.plot(df['Close'])
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.legend(['Original','Train','Test'])
plt.title('MLP plot')
plt.show()
```

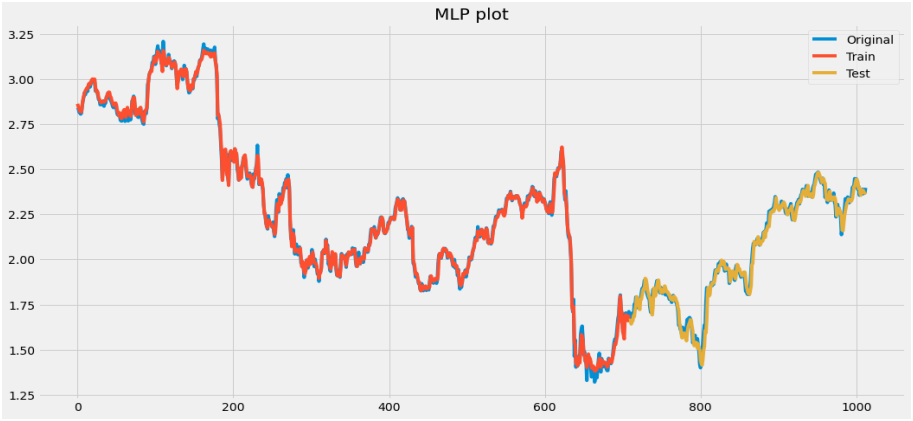


Figure 4.11: Plot shows how well the trend has been tracked by the prediction

4.3 Prediction of future values

The objective of this section is to predict the closing values of the stock price for the next 5 days, starting from the last date collected, which is 31st August 2021.
The timethat was used during the model training is 20 days. At this point, if we want to forecast the output for the next 5 days, we will use the values of the previous 20 days for the forecast.
Remembering that the length of the testing data is 304, then the last 20 days of input data needed are called x_input.

```
x_input=test_data[284 :].reshape(1,-1)

list_input=list(x_input)
list_input=list_input[0].tolist()
```

The data thus obtained is a normalized column array of the last 20 days.
In this case, we exploit the LSTM model and use the following code:

```
list_output=[]
i=0
while(i<5):
    if(len(list_input)>20):
        x_input=np.array(list_input[1:])
        print("{} day input{}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input=x_input.reshape((1,20,1))
        yhat=LSTM_model.predict(x_input, verbose=0)
        print("{} day output{}".format(i,yhat))
        print("-------------------------------------")
        list_input.extend(yhat[0].tolist())
        list_input=list_input[1:]
        list_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input=x_input.reshape((1,20,1))
        yhat=LSTM_model.predict(x_input, verbose=0)
        list_input.extend(yhat[0].tolist())
        list_output.extend(yhat.tolist())
        i=i+1
```

The output obtained is the following:

```
0 day input[0.56613757 0.55238095 0.55978836 0.55502646 0.57010582 0.5403142
 0.53946012 0.53798938 0.53576171 0.53284949 0.52944803 0.52573031
 0.52182215 0.51784289 0.5138616  0.50994277 0.50610721 0.50233972
 0.49866393 0.49504015]
0 day output[[0.49144542]]
-------------------------------------
1 day input[0.55238095 0.55978836 0.55502646 0.57010582 0.5403142  0.53946012
 0.53798938 0.53576171 0.53284949 0.52944803 0.52573031 0.52182215
 0.51784289 0.5138616  0.50994277 0.50610721 0.50233972 0.49866393
 0.49504015 0.49144542]
1 day output[[0.48788193]]
-------------------------------------
2 day input[0.55978836 0.55502646 0.57010582 0.5403142  0.53946012 0.53798938
 0.53576171 0.53284949 0.52944803 0.52573031 0.52182215 0.51784289
 0.5138616  0.50994277 0.50610721 0.50233972 0.49866393 0.49504015
 0.49144542 0.48788193]
2 day output[[0.4843475]]
-------------------------------------
3 day input[0.55502646 0.57010582 0.5403142  0.53946012 0.53798938 0.53576171
 0.53284949 0.52944803 0.52573031 0.52182215 0.51784289 0.5138616
 0.50994277 0.50610721 0.50233972 0.49866393 0.49504015 0.49144542
 0.48788193 0.48434749]
3 day output[[0.48086327]]
-------------------------------------
```

```
4 day input[0.57010582 0.5403142  0.53946012 0.53798938 0.53576171 0.53284949
 0.52944803 0.52573031 0.52182215 0.51784289 0.5138616  0.50994277
 0.50610721 0.50233972 0.49866393 0.49504015 0.49144542 0.48788193
 0.48434749 0.48086327]
4 day output[[0.4774149]]
-------------------------------------
```

Below is the output obtained from the code: note that in the first case ('0-day input') the input data are the twenty closing values of the test set from which we obtain the first predicted value for the date of 1 September 2021; in the second case ('1-day input') the data are composed of the last 19 of the test dataset and the first predicted value, i.e. '0-day output', to obtain in this way the price on September 2; in the same way we reason for the following days, where for example to predict the fifth day are taken the last 16 of the test set and the 4 outputs just predicted.
So the predicted future 5 days, in the normalized form, collected in the *list_output*, are:

```
[[0.5099427700042725], [0.5061072111129761], [0.5023397207260132],
[0.49866393208503723], [0.4950401484966278]]
```

Now the focus is to graphically represent the data obtained and display it in such a way that it follows the data collected until 31 August 2021.
First of all, we create the ranges used to create the plot:

```
last20days = np.arange(1,21)
future5days = np.arange(21,26)
```

Then, for now, let's display the data obtained by comparing and merging them exclusively with the last 20 days collected in the same plot.

```
df_Close=df['Close'].copy()
df_tol=df['Close'].tolist()
df_tol.extend(list_output)

plt.plot(last20days, scaler.inverse_transform(df_Close[993:]))
plt.plot(future5days, scaler.inverse_transform(lst_output))
```

Figure 4.12 shows how the trend in predicted data tends to be slightly negative. It is therefore expected that the investor who trusts this prediction will either decide not to invest any more capital or try to sell shares. The period under consideration is small and the decrease is not significant, but it is possible to understand the expected market trend from the data obtained.

This prediction can be made in the same way by taking as input a range of data that belongs to a larger period, for example, 100 working days. Having more data as input it is interesting to see a forecast of the stock price over a longer period, e.g. 15 days, i.e. three working weeks).
Applying the same code to this new data we get the Figure 4.12.
Looking at this figure, it is clearer that what might appear to be a negative trend in the closing price of the share, in truth has an almost constant trend over time.
We can therefore conclude that this stock does not undergo major fluctuations and is a fairly safe stock over the period following our analysis.

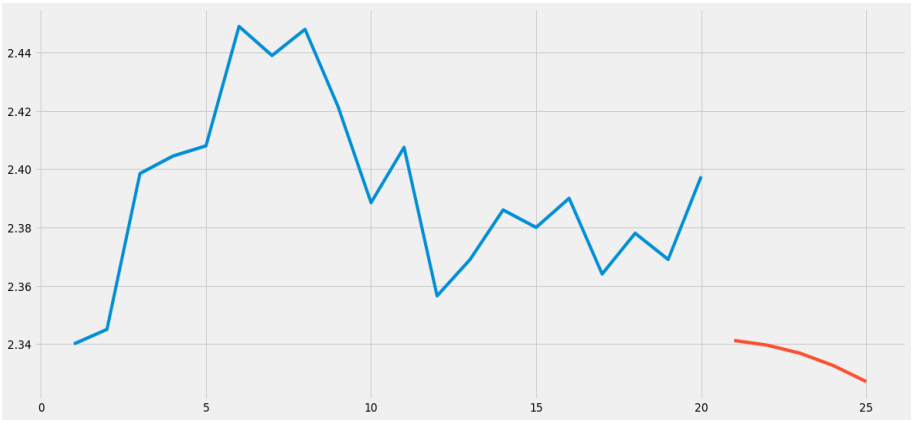


Figure 4.12: Predicted next 5 days close price

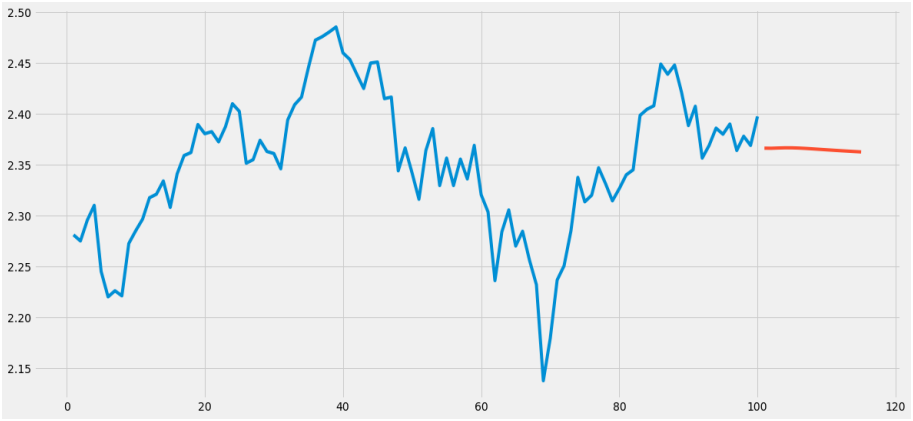


Figure 4.13: Predicted next 15 days close price

4.4 Evaluation techniques

We have looked at the theory behind the models, the collection and analysis of the data that was used, the preparation phases of each of the models, and their implementation.
At this point, it is necessary to be able to compare the models with each other and work out which one is the best fit for the data collected for prediction. Then all that remains is to test it to assess its actual effectiveness.

The learning algorithms used are regressors, i.e. they process the input dataset to produce a prediction model that can calculate the price from the characteristics of the dataset. In addition, the learning field is supervised.
What allows us to evaluate a trained model and its performance are the metrics. By correctly selecting an evaluation metric, or an equation used to objectively evaluate the performance of a model, we can get a good idea of how well the results produced by our model correspond to real-world observations. We can use the evaluation metric to determine whether improvements need to be made

to the model parameters or whether another modeling method needs to be considered.
The most commonly used metrics for analyzing the error of a regressor algorithm in supervised machine learning are based on the residuals, i.e. the differences between the predictions of the model and the correct responses called targets. The aim is to estimate a possible functional relationship existing between the dependent variable and the independent variables.

The concept behind the metrics that are used for regression analysis is as follows: the distance of the result obtained by the system from the expected result is measured, i.e. it is assessed how far or how close the response of the system is to the correct response.

The evaluation takes place by inserting input variables into the network and evaluating the output that the model returns. Consequently, it would be possible to insert any data into the input, as long as it has a structure compatible with that used to train the model.
This means that if one decides to predict the closing value starting, for example, from the data of the previous 20 days, the model expects a sequence of 20 numbers as input. But the variable it takes as an input is simply an array or vector, i.e. the model has no idea that it is a sequence. The difference is in how we implement the software that creates the model. So we could enter any vector with 20 random numbers and the model will still return an output.
This could be useful if you want to do behavioral simulations by creating ad hoc scenarios that you want to study by submitting them to the model.

4.4.1 Model evaluation metrics

The final part of constructing a regression model is evaluating how well the supervised model fits the data.
Metrics are those values that allow us to decide whether the model is ready for distribution or not.
Although choosing an evaluation metric seems like a simple task, there are many different metrics available. Each of them has characteristics that distinguish it and make it more or less suitable depending on the input data and the type of model being evaluated. The fact that the model being analyzed is a forecasting model, i.e. a regression model, excludes all those metrics, such as accuracy, that are suitable for classification models. This is due to how classification and regression problems present data in a completely different format, making it almost impossible to use a single equation for both types of problems.
Consequently, each type of machine learning problem has its own unique set of applicable evaluation metrics.

At this point, we will examine regression metrics and explore why we use them.
Each of the metrics described below quantifies the error associated with a model using a different method.
For regression to be useful, we need a method that we can use to determine the performance of a regression model. Evaluation metrics are used for this purpose,

providing a means to objectively evaluate the performance of a regression model by quantifying its goodness of fit.

Mean Absolute Error

MAE is the simplest form of regression metric that will be used. It is the average of the absolute error values calculated for each point in the dataset. The absolute error is the residual of the actual value and the predicted value. Thus positive and negative residuals do not cancel each other out.
It is calculated using the following equation:

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

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.
The resulting value represents the mean absolute error or the mean vertical distance between each pair of predicted and actual values when represented graphically. An ideal model, i.e. one whose prediction coincides with reality, produces an MAE of zero; therefore, the closer the observed MAE is to zero, the better the model fits the data.
The MAE calculation considers all penalties equally, regardless of whether the predicted value is less or more than the actual value. It also does not scale the amount of penalty applied to an error by its size. In fact, MAE may be an appropriate metric to use when there is no need to apply larger penalties for outliers or for datasets that contain few or no outliers.
The distinguishing feature of this metric is that each residual is proportional to the total error, i.e. all errors are treated equally.

To implement the code useful for calculating the above metrics, the *sklearn.metrics* library has to be used.

```
from sklearn.metrics import mean_absolute_error

MAE_train_LSTM = mean_absolute_error(y_train,train_predict_LSTM)
MAE_test_LSTM = mean_absolute_error(y_test,test_predict_LSTM)

MAE_train_GRU = mean_absolute_error(y_train,train_predict_GRU)
MAE_test_GRU = mean_absolute_error(y_test,test_predict_GRU)

MAE_train_Conv = mean_absolute_error(y_train,train_predict_Conv)
MAE_test_Conv = mean_absolute_error(y_test,test_predict_Conv)

MAE_train_MLP = mean_absolute_error(y_train,train_predict_MLP)
MAE_test_MLP = mean_absolute_error(y_test,test_predict_MLP)
```

This function has as input the first parameter y_i, and as second parameter the predicted value $\hat{y}_i$ from the respective model.
At this point we want to compare the results obtained for each model. To do this we plot the MAE values per model, tested on both the training set and the test set.

Testing on the training set means that, after training the model, testing is done by providing the framework with only the inputs that make up the training set, in order to evaluate the resulting outputs. This approach aims to evaluate how well the model is able to interpret the data it has been trained on.
On the other hand, testing on the testing set allows us to actually evaluate the prediction performance of the model, since it is data that the network has never had the opportunity to analyse before.

Figure 4.14 shows how the recurrent neural networks, i.e. the LSTM and

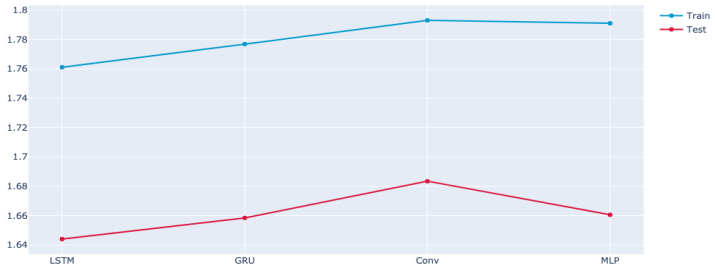


Figure 4.14: MAE results for the different models

GRU, achieve lower error rates than the convolutional network model and the multilayer perceptron.
The results obtained with the testing set are better than with the training set. This is a good thing, because if our model does much better on the training set than the testing set, then we are probably overfitting, i.e. there is overfitting. Overfitting occurs when the model captures noise along with the underlying model in the data.

Mean Squared Error

The MSE refers to the mean value of the squared error values calculated for each data point. The representative equation is the following:

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

It is an equation reminiscent of that seen above for MAE. The squaring of each bias term guarantees that the MSE will be greater than or equal to zero.
Since each bias value is squared before the sum of all terms is taken, the impact of larger observed errors on the total error value is exponentially greater than the impact of smaller observed errors. The larger the observed error value, the greater the penalty applied when calculating MSE. This is the main feature that distinguishes this metric from MSE, and for this reason, it is also a more popular regression evaluation metric. In particular, using MAE each residual is proportional to the total error, whereas with MSE the error grows quadratically. This means that MSE will have a higher total error due to the presence

of outliers than they would have in MAE. A higher MSE means that the model will be penalized for making predictions that differ significantly from the actual value. This means that a large difference between predicted and actual values will be more punished in MSE than in MAE.
Furthermore, MSE is less robust to outliers than MAE, as it squares the residuals.
Like MAE, an MSE value closer to zero indicates better model performance.

The function used to calculate the MSE belongs to the same library sklearn.metrics, and is applied to each model via the following code:

```
from sklearn.metrics import mean_absolute_error

MSE_train = mean_absolute_error(y_train,train_predict)
MSE_test = mean_absolute_error(y_test,test_predict)
```

Due to the squaring of residuals, the MSE will always be greater than the MAE. Therefore, we cannot compare the MSE with the MAE. However, we can compare these values with metrics from a different regression model. This will help us to choose which is the best model for the data. Again, we obtain

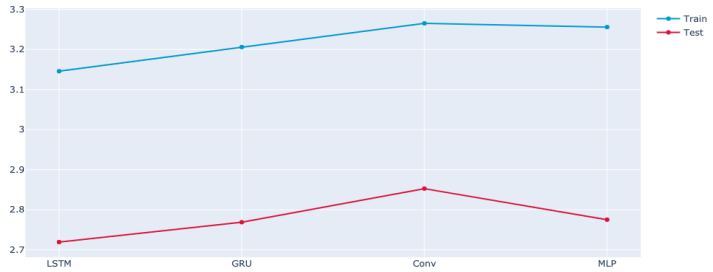


Figure 4.15: MSE results

similar results to the MSE. The values of the two metrics MAE and MSE do not coincide and are not comparable either. However, the important result from both evaluations is that the LSTM model is the most accurate in predicting future values, followed by the GRU model, which is also part of the recurrent neural networks.

Root Mean Squared Error

RMSE is one of the most popular evaluation metrics for regression problems. It is calculated by taking the square root of the MSE:

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

RMSE also represents the standard deviation of the residuals produced by our model. Since the mean value of the residuals is always zero, knowing the stan-

dard deviation of the residuals gives us an idea of how much our residuals are centered around zero. Like the other evaluation metrics we have discussed, a lower value of RMSE indicates better model performance.
Both RMSE and MSE are preferable metrics to use for datasets that contain several outliers. Applying an additional penalty for larger differences between predicted and actual values will help the model adapt to outliers during the fitting process.
MSE has a higher value because we are squaring it. To bring this value to the same scale as the forecast error, we use RMSE. This facilitates the interpretation. Since both MSE and RMSE square the residuals, they are equally affected by the outliers.
RMSE should be used over MAE or other evaluation metrics in situations where the observed data have a skewed conditional distribution. If MAE is used as an objective function that needs to be minimized when training a machine learning model, it will produce a biased fit that is closer to the median than would be obtained if RMSE were used.

The resulting value can be interpreted in the same units as the value we are trying to predict, which makes it easier to understand than other metrics. However, it is important to remember that RMSE values can only be compared between models that measure error using the same units.

Also in this third case, the results are consistent with those obtained through the previous metrics. Observe the values obtained in Figure 4.16.
The value of MAE is typically relatively close to the value of the root mean

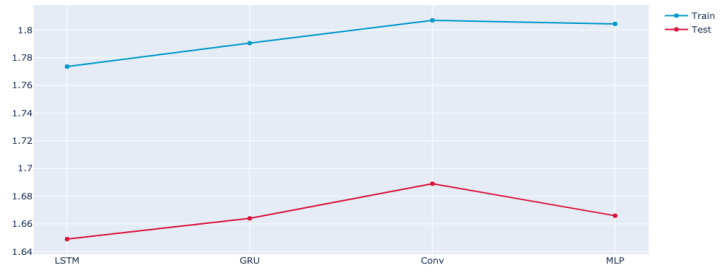


Figure 4.16: RMSE results

square error.

R-Squared

R-Square, also known as the coefficient of determination, "determines the proportion of the variance in the dependent variable that can be explained by the independent variable, known as the goodness of fit".[**7**]

This is calculated from the sum of the squares of the residuals divided by the sum of the mean total. The closer the value is to 1, the better the fit between

the prediction and the actual value. Despite this R^2 does not always guarantee whether a regression model fits the data adequately.
Regression models with a low R^2 can be good models: if the independent variable is statistically significant (related to the dependent variable), an important conclusion can be drawn between the relationships of the variables. The statistically significant coefficient represents an average change in the dependent variable given a one-unit shift in the independent variable.
However, for forecasts to be relatively accurate, a low R^2 can be a problem. A high R^2 is necessary for accurate predictions, but even a high R^2 does not always guarantee that the model is perfect.
Even when the model is perfect, i.e. R^2 is equal to 1 and the regression line fits the data perfectly, the model may still have difficulties with the test data resulting in high variance. The results obtained are not optimal, but confirm

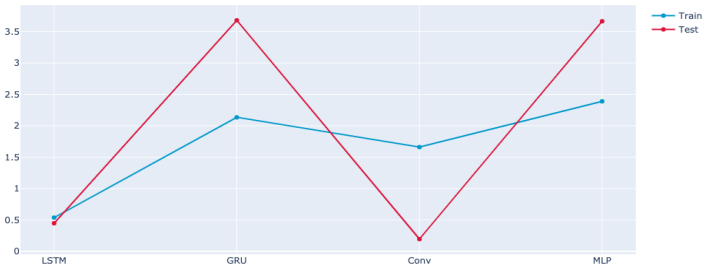


Figure 4.17: R^2 results

that the model that comes closest to the correct approximation is the LSTM model. In this case, it is followed by the convolutional network, in which a better value is obtained for the test set than for the training set. This could be due to an overfitting problem.

Chapter 5

Conclusions

In this last chapter, we will conclude the case under analysis. It will highlight strengths and limitations and how improvements could be made to the model.

The thesis focused on exploring some of the main technologies used in the field of artificial intelligence and machine learning. In particular, an attempt was made to follow a common thread to identify the links existing between the various technologies and the motivations that led scholars to identify new solutions.
The focus was on the method of supervised learning to explore the real motivations that lead to a neural model aimed at learning and then illustrating the evolution of the various approaches.
The notions learned were then used in a real case study such as stock-prediction to understand the behavior of neural networks in all their facets.

It is possible to state that the problem of stock price prediction has not been solved, i.e. the predicted prices are not reliable if one wants to know the exact closing stock price. In any case, interesting results were obtained that were hardly achieved using other more traditional models, such as the ARMA model or more classical machine learning models.
This is clearly because stock prices depend on more complicated factors than the simple time series of past prices.
However, the rationale for the approach used is based on the fundamentals of technical analysis, according to which it is believed that it is possible to obtain an estimate of future prices by analyzing past price trends because the latter already incorporates the market's information set.

The nature of the results obtained, which are not very accurate, may depend on whether the price function is time-varying. Incidentally, a function is called time-invariant (or stationary) if it remains constant over time.
In the case at hand, the non-stationary function changes at each instant of time. This implies that the model sets weight and bias based on a function that potentially changes instantly by instant.
Therefore, a network that is good today may have to be recalibrated tomorrow based on the new values.

The models analyzed and implemented have returned nothing short of astounding results in mapping the input-output function that was submitted to them. This may not be too useful for the problem we have addressed given the non-stationary nature of the series, but it does open up important possibilities for other financial case studies involving function mapping, such as the pricing of American options, exotic options, or even potentially cryptocurrencies.

The course of this research thesis has highlighted the modalities of use of the neural networks on time series, showing also some possible applications that can be carried out through the techniques of Artificial Intelligence, which are becoming more and more common today, which are becoming more and more commonly used nowadays.
Starting from the first chapters, we tried to provide basic knowledge that would give a good background on what we would see in terms of applications. Then, we tried to provide notions about the state of the art, i.e. the techniques that are traditionally used for time series analysis.
The main tasks on which the application study was focused were the training of the different models on the input dataset, collecting daily data from 1 September 2017 to 31 August 2021, and the consequent prediction of the following 5 days. The main objective is not to predict a closing value that is extremely accurate but to understand whether the stock price trend is bullish or bearish. These techniques are used today for research on online buying and selling sites. In general, they are used by companies to predict a specific piece of information, such as next month's sales, about the likely future value of a variable.
The tools that have been presented therefore serve for systematic processing of the available information and the results of the forecasts do not necessarily have to replace what the investor thinks but only help him to decide.
But the real aim has been to test these techniques to realize how much closer we are getting to the behavior of the human being and specifically the human brain. To do this, an analysis and testing study was carried out: this was precisely the core of our research.

In the experiment, an attempt was made to predict the future price based on the historical series of past prices, based on the fundamentals of technical economic analysis, according to which market information is already embedded in the past price series.
The experimental results showed that:

- in the case under analysis the LSTM model is the best performing network, but more generally recurrent neural networks seem more suitable for time series forecasting;
- the various models studied manage to be sufficiently flexible; in fact, the control of the stock price value due to Covid in March 2020 was well followed and the predictions on the training dataset did not deviate too much;
- Deep Learning models are much more sophisticated and accurate than more classical techniques such as exponential smoothing or the ARMA model, especially since the more data a model has as input, the more it learns and is flexible;

- you are more likely to get satisfactory results the more network configurations you can test, so the problem of processing time becomes very relevant here.

Neural networks can approximate financial time series, but when it comes to predicting future trends (both short and long term) they fail. One reason could be that price dynamics do not follow any recurring pattern, but depend solely on new information and its effects on the market. Another hypothesis is that other network configurations should be explored, considering data with greater time depth, and introducing cross-correlations into the information set, to take account of the interdependence between futures.
In conclusion, it can be said that, at least at a theoretical level, the work carried out has brought notable improvements to the previous trading system, validating the importance that the analysis of the historical data has towards the future of the prediction also to graphical level.

5.1 Possible improvements to the model

One aspect that has not yet been taken into account is the choice of the hyper-parameters of each layer, i.e. the number of perceptrons composing it, the characteristics of the activation function and the loss function, and, in the case of convolutional and pooling layers, the size of the scan kernel.
Unfortunately, there are no theorems or methods that allow us to know the optimal parameters of a network, so we tend to proceed by trial and error, trying out different configurations and choosing those that give the best results.
Thus, the main disadvantage of neural networks is that the optimal structure and parameters that the network must have to solve a given problem cannot be obtained through a formula or theorem.
It is only possible to obtain excellent results, but no matter how good they may be, there may always be a network of networks among the practically infinite possible configurations that give better ones.
It is possible to understand what the optimal hyper-parameters are by implementing an automatic structure that would allow us to try out different combinations of parameters.
Since the models obtained achieved good results across the evaluation metrics and the calculation time to test all combinations is very high, it was chosen to set the hyper-parameters to values suitable for a time series dataset, with a fairly low number of data.
In fact, it was possible to find the combination of Rolling Window length and number of nodes values that were best overall for each of the four models.
This can only be achieved by successive refinements, using a Feature Selection algorithm, thus testing the model several times with different combinations of variables to identify those that produce better indicators of performance.

One of the major risks involved in using these forecasting methods is that the stock price will undergo sudden changes due to external factors, such as the 2008 crisis or Covid. The model cannot know these things, so you have to learn to be flexible and update the situation as quickly as possible.
First of all, it is advisable not to take too long a time horizon, but rather a few days or weeks.

Another solution is to opt for a normalization method for calculating the percentage change in the real variables. One can capture the seasonality by taking into account certain "classifier" variables to try to make the network understand whether certain trends can be associated with the period in which they are detected.

5.2 Future Developments

Although the work carried out has considered various scenarios, it can certainly be improved and extended.
A possible experiment could be to optimize the predictive phase, considering different configurations of the parameters of each, attributing values other than the default ones.
One should also consider the idea of increasing or decreasing the width of the training window, to establish more precisely how many days are needed to predict the signal with maximum accuracy.
As far as technical analysis is concerned, its full potential could be exploited by evaluating the hypothesis of including other features from the same sector in the model. Therefore, the regressor could base its learning phase not only on historical prices but also on indicators of statistical, economic, and political imprint, already considered in many studies preceding this one.
The final objective that one wants to reach with time is to create a trading system that reflects as much as possible the real world of trading, which therefore allows one to understand the market trend based on past data.
Moreover, it is possible to modify the development of our problem, solving it using classifiers: in this case, we would not go to predict the closing value of the stock price, but the class of future days. The classes could be the chosen investment position, which could be long or short. When talking about trading strategies, these terms are nothing more than synonyms for "buy" and "sell".

Deep Learning is a fast-growing field that could revolutionize all data analysis and prediction systems. The potential is enormous and the fields of application are endless. In the financial sphere, it would be interesting to combine these data prediction models with techniques, such as sentiment analysis or text mining, which allow the acquisition of information from financial news sites. Based on this news, it would then be possible to draw up a sentiment profile of investors. These are then disciplines which, integrated with the structures just implemented, can process the information contained in texts, random and unstructured, transforming it into data in a structured and normalized format. A text can be used as an input signal to derive the author's opinion on the subject. Furthermore, it is possible to classify news based on the impact it may have on investors' sentiments and, consequently, how these may impact stock price fluctuations.

Another interesting application of this study is the field of cryptocurrencies. The problem that characterizes them and for which it is still difficult to make a price prediction is the very high volatility. This behavior has brought huge profits or huge losses to traders who have bet on them. Blockchain, a decentralized technology that allows cryptocurrencies to be produced and dis-

tributed, could also be facilitated by Deep Learning algorithms, for example by developing models that facilitate the traceability of transactions.

Bibliography

[1] *IAS 39 Financial Instruments: Recognition and Measurement* https://www.cpdbox.com/ias-39-financial-instruments-recognition-and-measurement/ (cit. on pp. 5)

[2] *Risk Management Strategies* https://www.avatrade.it/education/online-trading-strategies/risk-management-strategies (cit. on pp. 7)

[3] Francois Chollet, (2017) *Deep Learning with Python* https://books.google.it (cit. on pp. 28)

[4] Daniele Caldarini, *Machine learning and applications for industry* https://techblog.smc.it/en/2020-05-25/machine-learning-industry (cit. on pp. 30)

[5] https://www.investopedia.com (cit. on pp. 25, 53)

[6] Felix Gers, Fred Cummins, Santiago Fernandez, Justin Bayer, Daan Wierstra, Julian Togelius, Faustino Gomez, Matteo Gagliolo, and Alex Graves, (2015) *Understanding LSTM Networks* https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (cit. on pp. 42)

[7] Elias Eid, (2021) *Profiling Alloy Models* https://uwspace.uwaterloo.ca/bitstream/handle/10012/17341/Eid_Elias.pdf?sequence=3&isAllowed=y (cit. on pp. 77)

[8] Neelam Tyagi, (2021), *A Tutorial on Exponential Smoothing and its Types* https://www.analyticssteps.com/blogs/tutorial-exponential-smoothing-and-its-types (cit. on pp. 9, 12)

[9] Marcello Gomitoni, (2013), *Serie Storiche Finanziarie: Studio e Previsione* https://terna.to.it/tesi/gomitoni.pdf

[10] Alberto De Santis, *Serie Storiche* http://www.diag.uniroma1.it//~desantis/NOTE/timeseries.pdf

[11] Vincenzo Capasso, *Serie Storiche* http://www.mat.unimi.it/users/michel/\\statmat2/TIMEseries.pdf

[12] Anais Dotis-Georgiou, (2019), *Autocorrelation in Time Series Data* https://www.influxdata.com/blog/autocorrelation-in-time-series-data/ (cit. on pp. 13)

[13] https://otexts.com (cit. on pp. 13, 18, 19, 20, 22, 24, 25)

[14] Andrius Buteikis, *Time series with trend and seasonality components* http://web.vu.lt/mif/a.buteikis/wp-content/uploads/2019/02/Lecture_03.pdf

[15] Gianni Marliani, https://local.disia.unifi.it/marliani/stat_eco_A/seriestoriche2.pdf (cit. on pp. 18)

[16] Jason Brownlee, (2018), *A Gentle Introduction to Exponential Smoothing for Time Series Forecasting in Python* https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/ (cit. on pp. 22, 23)

[17] Vasilis Papastefanopoulos, Pantelis Linardatos, Sotiris Kotsiantis, (2020) *COVID-19: A Comparison of Time Series Methods to Forecast Percentage of Active Cases per Population* https://www.mdpi.com/2076-3417/10/11/3880/pdf (cit. on pp. 23)

[18] https://local.disia.unifi.it/marliani/stat_eco_A/lucidi_SS.pdf (cit. on pp. 9, 11, 26)

[19] Franco Flandoli, *Serie storiche e processi stocastici* https://people.dm.unipi.it/trevisan/didattica/455AA/Appunti\%20sull'analisi\%20statistica\%20delle\%20Serie\%20Storiche\%20(prof.\%20Flandoli).pdf (cit. on pp. 27)

[20] https://www.youtube.com/watch?v=lEfrr0Yr684&ab_channel=deeplizard

[21] https://quantdare.com/what-is-the-difference-between-\\bagging-and-boosting/ (cit. on pp. 31)

[22] V.A., (2017), *Data Analytics for Intelligent Transportation Systems* https://www.sciencedirect.com/topics/psychology/machine-learning (cit. on pp. 28)

[23] (2020), *THE THREE CATEGORIES OF ALGORITHMS WITHIN MACHINE LEARNING EXPLAINED* https://tmc-employeneurship.com/it/articles/the-three-categories-of-algorithms-within-machine-learning-explained (cit. on pp. 29)

[24] Aidan Wilson, (2019), *A Brief Introduction to Supervised Learning* https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590 (cit. on pp. 29)

[25] https://deeplizard.com/learn/video/lEfrr0Yr684 (cit. on pp. 30)

[26] Jason Brownlee, (2018), *How to Develop a Weighted Average Ensemble for Deep Learning Neural Networks* https://machinelearningmastery.com/weighted-average-ensemble-for-deep-learning-neural-networks/ (cit. on pp. 31, 58)

[27] Ratnadip Adhikari, R. K. Agrawal *An Introductory Study on Time Series Modeling and Forecasting* https://arxiv.org/ftp/arxiv/papers/1302/1302.6613.pdf (cit. on pp. 33)

[28] Diego Salinas, (2020), *Deep Learning Use Cases: Separating Reality from Hype in Neural Networks* https://towardsdatascience.com/deep-learning-use-cases-separating-reality-from-hype-in-neural\\-networks-9d31cc1bc746 (cit. on pp. 32)

[29] Marco Tortora, (2020), *Uso delle Reti Neurali per la previsione di serie storiche finanziarie* https://webthesis.biblio.polito.it/14856/1/tesi.pdf (cit. on pp. 37, 40, 41, 45)

[30] Marco Del Pra, (2020), *Time Series Classification with Deep Learning* https://towardsdatascience.com/time-series-classification-with-deep-learning-d238f0147d6f (cit. on pp. 35, 45)

[31] Davide Maltoni, *Reti Neurali* http://bias.csr.unibo.it/maltoni/ml/DispensePDF/8_ML_RetiNeurali.pdf (cit. on pp. 35, 39)

[32] Cristiano Casadei, (2020), *Le reti neurali ricorrenti* https://www.developersmaggioli.it/blog/le-reti-neurali-ricorrenti/ (cit. on pp. 42, 44, 61)

[33] John Gamboa, (2017), *Deep Learning for Time-Series Analysis* https://arxiv.org/pdf/1701.01887.pdf (cit. on pp. 45)

[34] Matthew Stewart, (2019), *Simple Introduction to Convolutional Neural Networks* https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac (cit. on pp. 45)

[35] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, Pierre-Alain Muller, (2019), *Deep learning for time series classification: a review* https://arxiv.org/pdf/1809.04356.pdf (cit. on pp. 46)

[36] dataman-git,(2020) *Una guida tecnica su RNN / LSTM / GRU per la previsione del prezzo delle azioni* https://ichi.pro/it/una-guida-tecnica-su-rnn-lstm-gru-per-la-previsione-del-prezzo-\\delle-azioni-207683008529222 (cit. on pp. 43, 44)

[37] Bryan Lim, Stefan Zohren,(2020) *Time-series forecasting with deep learning: a survey* https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2020.0209 (cit. on pp. 34)

[38] Lorenzo Govoni ,(2021) *Rete Neurale, Deep Learning e principali applicazioni* https://www.lorenzogovoni.com/deep-learning-e-applicazioni/ (cit. on pp. 33)

[39] Mohit Mayank, (2020) *A practical guide to RNN and LSTM in Keras* https://towardsdatascience.com/a-practical-guide-to-rnn-and-lstm-in-keras-980f176271bc (cit. on pp. 60)

[40] *Corso di Keras per machine learning* https://www.domsoria.com/2020/02/ (cit. on pp. 57)

[41] Nikolas Adaloglou, (2020) *RNN cells: analyzing GRU equations VS LSTM, and when to choose RNN over Transformers* https://towardsdatascience.com/rnn-cells-analyzing-gru-equations-vs-lstm-and-when-to-choose (cit. on pp. 63)