University POLITEHNICA of Bucharest
Faculty of Electronics, Telecommunications and Information Technology

Financial Time Series Forecasting Using Spiking Neural Networks

Diploma Thesis

**Submitted in partial fulfillment of the requirements
for the Degree of *Engineer***
in the domain *Electronics Engineering, Telecommunications and Information Technology*
Study program: *Technology and Telecommunication Systems*

Thesis advisor
Prof. Corneliu Burileanu, PhD
Ana-Antonia Neacșu

Student
Roxana-Maria Iliese

Year 2020

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **Tc**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **ILIESE T. Roxana-Maria , 441C-TST.**

1. Titlul temei: Predictia seriilor temporale financiare cu ajutorul retelelor neurale de tip spiking

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) și specificații de proiectare:
Proiectul de diploma consta in prezicerea seriilor temporale financiare cu ajutorul retelelor neurale de tip spiking.
Obiectivul principal il reprezintă dezvoltarea si antrenarea unor arhitecturi de rețele inspirate din modelul uman ce introduc și conceptul de timp. Se va face o analiza statistica a seriilor temporale financiare, dat fiind faptul ca acestea au o evolutie partial haotica, depinzand de foarte multi factori, unii dintre ei fiind chiar necunoscuti.
Validarea performantelor sistemului dezvoltat se va face pe o baza de date reala, in contextul prezicerii evoluției prețului unor actiuni ale unor companii listate la bursa de valori. Se vor compara rezultatele obtinute de algoritm cu preturile reale ale respectivelor actiuni, la un anumit moment de timp, urmarindu-se o diferenta cat mai mica intre cele doua valori. In urma analizei rezultatelor obtinute de algoritm, utilizatorul va putea lua o decizie cat mai corecta in privinta investitiilor sale de pe piata de actiuni.

3. Resurse folosite la dezvoltarea proiectului:
Python, Tensorflow, Keras, Numpy, Matplotlib

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
DEPI, PC, POO

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2019-11-29 21:56:09

Conducător(i) lucrare,
Prof. dr. ing. Corneliu BURILEANU

Student,

Director departament,
Conf. dr. ing. Eduard POPOVICI

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **2b72bc75ad**

Statement of Academic Honesty

I hereby declare that the thesis *Financial Time Series Forecasting Using Spiking Neural Networks*, submitted to the Faculty of Electronics,Telecommunications and Information Technologies, University POLITEHNICA of Bucharest, in partial fulfillment of the requirements for the degree of *Inginer* in the domain Electronics and Telecommunications, study program *Tehnologii si Sisteme de Telecomunicatii* is written by myself and was never before submitted to any faculty or higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand that plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations or measurements I performed, together with the procedures used to obtain them, are real and indeed come from respective simulations or measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, June 2020.
Student: Roxana-Maria Iliese

Contents

List of Figures

List of Tables

List of Abbreviations

ACF = Autocorrelation Function
ANN = Artifical Neural Network
AR = Autoregressive
ARCH = Autoregressive Conditionally Heteroskedastic
ARIMA = Autoregressive Integrated Moving Average
ARMA = Autoregressive Moving Average
CH = Chaterring
FS = Fast Spiking
FX = Foreign Exchange
GARCH = Generalised Autoregressive Conditionally Heteroskedastic
IB = Intrinsically Bursting
IF = Integrate-and-Fire
LIF = Leaky Integrate-and-Fire
LSTM = Long Short Term Memory
LTD = Long-Term Depression
LTP = Long-Term Potentiation
LTS = Low-Threshold Spiking
MA = Moving Average
MSE = Mean Squared Error
PACF = Partial Autocorrelation Function
PSP = Postsynaptic Potential
RMSE = Root Mean Squared Error
RS = Regular Spiking
RZ = Resonator
SARIMA = Seasonal Autoregressive Integrated Moving Average
SNN = Spiking Neural Network
STDP = Spike-Time Dependent Plasticity
TC = Thalamo-cortical

Thesis Motivation

Peter Bernstein wrote in his monumental book *Against the Gods* that what distinguishes the thousands of years of history from what we think as modern times is not what we might be inclined to think of: the progress of science, technology, capitalism, and democracy. The revolutionary idea that marks the boundary between modern times and the past is our dexterity when dealing with risk: the notion that the future is more than a whim of the gods. The process of mastering the risk gave humans the confidence that men and women are not passive before nature. Before human beings discovered a way across that boundary, the future was a mirror of the past and the nebulous domain of oracles and psychics who held a monopoly over the knowledge of anticipated events.

The beginning of forecasting as we know it today, dates back to the year 1654, when the famed French mathematician Blaise Pascal was challenged to solve a puzzle by Chevalier de Mere, a french nobleman with a taste for both gambling and mathematics. The puzzle in question tackles the division of the awards of an unfinished game of chance between two players when one of them is ahead. The discovery of Pascal and Pierre de Fermat established the foundation of the theory of probability. Their solution to the puzzle implied that people could for the first time make decisions and forecast the future with the help of numbers.

It is undeniably easy to understand the humans' desire to predict the future - besides the will to control certain outcomes, it is the endless fascination of defying the nature. The games of chance, along with the stock market and the bond market, are natural laboratories for the study of risk and prediction, as they lend themselves so easily for quantification; their language is the language of numbers. It is not hard to recognize that forecasting financial data is probably one of the most challenging predictions we strive to make, partly due to the multitude of factors we are not aware of and partly because of the great deal of risk and reward associated with it.[2]

This thesis will analyse the ability to predict the kind of data which has been intriguing people since 1602, when the Dutch East India Company officially became the world's first publicly traded company by releasing shares on the Amsterdam Stock Exchange. The prediction will be performed with the help of artificial neural networks.[3]

Chapter 1
Time Series

1.1 Introduction

A time series is a chronological or time-oriented sequence of observations $(x_t)_{1\leq t\leq N}$, each one being recorded at a specific time t. The variable of interest, X, could be anything from a stock price, the wind speed in a certain direction at a location, or the number of people who cross a country boarder. Each of these values could be collected on a daily, weekly, monthly, quarterly or annual basis, but any reporting interval could be used.[4] For instance, earthquake activity data is reported in milliseconds.[5]

To illustrate this, figure 1.1 shows the daily new cases of COVID-19 in the world, for a time period of 65 days: from the 10^{th} of March 2020 until the 13^{th} of May 2020. Thus, the set T_0 of times at which observations are made has 65 elements {(10 March), (11 March),... ,(13 May)}. We can see from this graph that each time point, t has a corresponding value, (x_t) which in this case is an integer, representing the number of people tested positive for the virus in that specific day.

There are two main motivations for studying time series:

1. To develop an understanding of the structure and underlying forces that produced the observed data.

2. To find a model that fits our data and proceed to make forecasts and even control the behaviour of future data points. [6]

However, in order to achieve the last, it is almost critical to master the first. As a consequence, the next section will briefly present some properties and characteristics of time series, which help us to understand better the data we have collected and draw insightful conclusions in order to predict and manipulate future events.

1.2 Covariance and Correlation

Covariance is a measure of the joint variability of two random variables, X and Y say. It is defined as follows:

$$\begin{aligned} cov(X,Y) &= E\{(X - E\{X\})(Y - E\{Y\})\} \\ &= E\{XY\} - E\{X\}E\{Y\} \end{aligned} \tag{1.1}$$

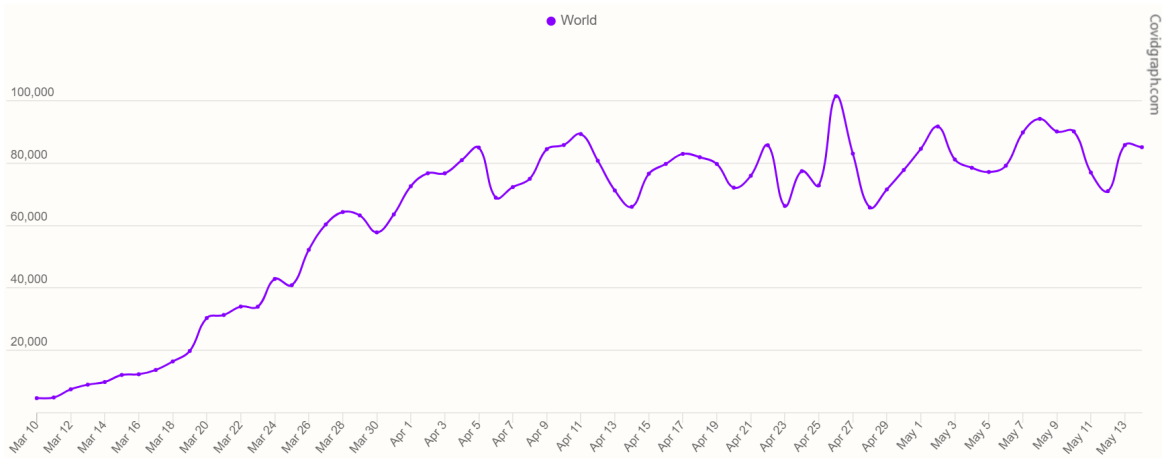

Figure 1.1: Daily New Cases of COVID-19 in the World, 10 Mar. 2020-13 May 2020 [1]

From the definition we highlight three cases:

- when X is above the mean and then Y also tends to be; in this case we say to have a *positive covariance*
- when X is above the mean and Y tends to be below its mean; this is the case of *negative covariance*
- there is no relationship of this type between X and Y ($E\{XY\} = E\{X\}E\{Y\}$), so we have *zero covariance*

Therefore, this gives us a measure of linear dependency between two random variables.

Correlation is a normalized measure of covariance.

We define correlation ρ as:

$$\rho = \frac{cov(X,Y)}{\sqrt{var(X)var(Y)}} \tag{1.2}$$

From the Cauchy-Schwartz inequality we conclude that $-1 \leq \rho \leq 1$.

In the context of time series analysis, we will be interested in looking at these two measures, covariance and correlation, within the random process X_t, at two different points in time. This gives us a good overview of the relationship between two random variables, X_{t_1} and X_{t_2}. [7]

1.3 Seasonality

Many time series exhibit seasonality. In this context, we refer to seasonality as periodic fluctuations that appear in our set of values. For example, ice-cream sales tend to peak in the summer, compared to other months of the year. This trend is mostly present in economic time series and it less popular among engineering or scientific data sets. [6]

Seasonality is a very useful kind of information we can gather from our data, as it allows us to conclude that certain patterns will repeat in the future, and, more importantly, *when* they will occur.

Based on a database which includes observations made on a daily basis of the minimum temperatures in Melbourne, Australia [8], figure 1.2 illustrates the phenomenon of seasonality

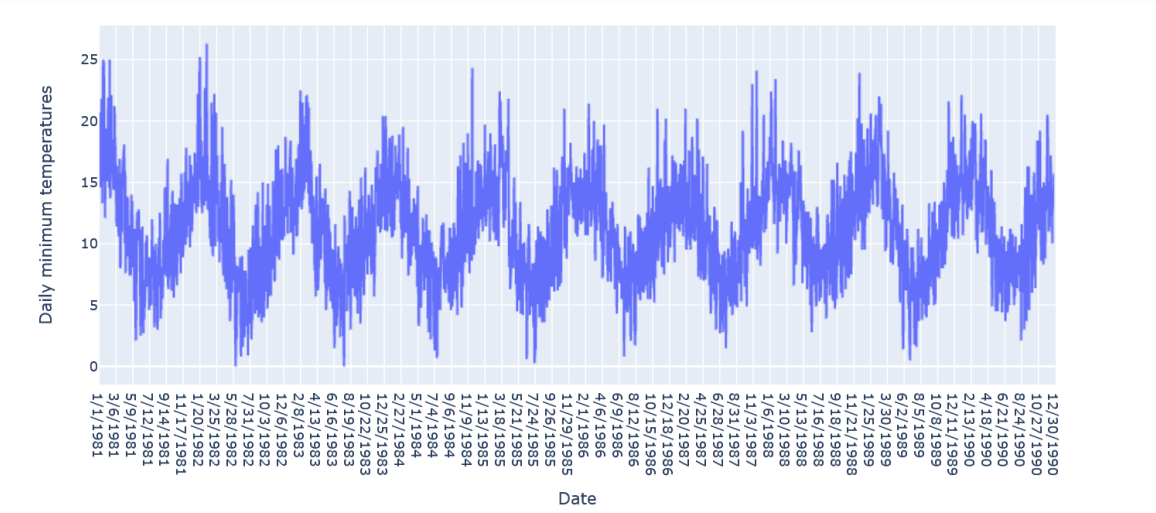

Figure 1.2: Daily minimum temperatures in Melbourne, Australia, 1981-1990

in time series. We can see that temperatures between 0°C and 5°C occur between the months May and September. This pattern tends to repeat itself every year, between the same months. The same is true for the months January, February and March, when the lowest temperatures are 20-25°C.

1.4 Stationarity

We say that a time series is *completely* or *strictly stationary* if the joint cumulative distribution function of $\{X_{t_1}, X_{t_2}, ..., X_{t_n}\}$ is the same as the joint cumulative distribution function of $\{X_{t_{1+\tau}}, X_{t_{2+\tau}}, ..., X_{t_{n+\tau}}\}$.

$$F_X(x_{t_{1+\tau}}, .., x_{t_{n+\tau}}) = F_X(x_{t_1}, .., x_{t_n}) \tag{1.3}$$

This means that a completely stationary process has a probabilistic structure that is invariant under a shift in time.

We say that a time series is *second-order* or *weakly stationary* if the following conditions are met:

1. Its *expected value* or *mean* is a constant independent of time:

$$E\{X_t\} = \mu \tag{1.4}$$

2. Its *variance* is a constant independent of time:

$$var\{X_t\} \equiv \sigma^2 \quad (= E\{X_t^2\} - \mu^2) \tag{1.5}$$

3. The *covariance* between X_t and $X_{t+\tau}$ is a constant independent of time:

$$cov(X_t, X_{t+\tau}) \equiv \gamma_\tau \tag{1.6}$$

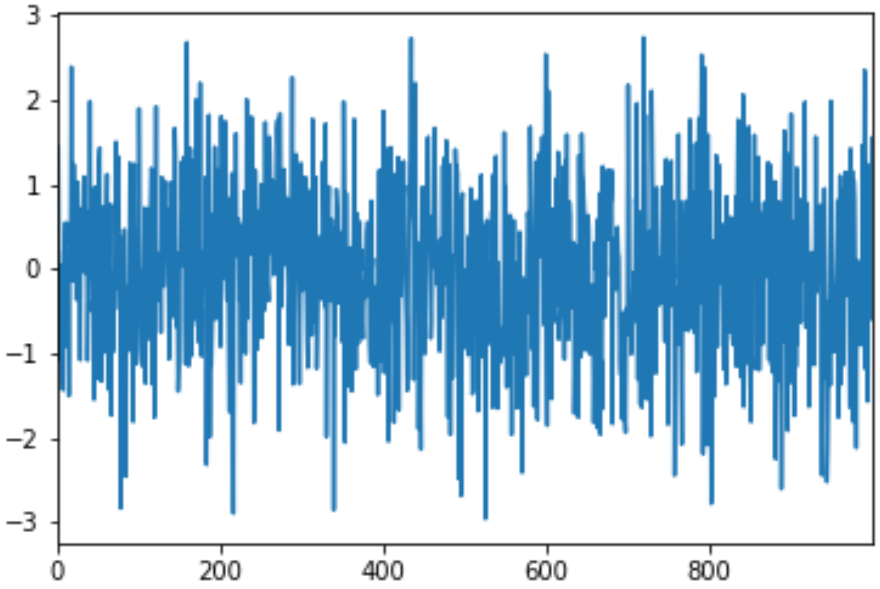

Figure 1.3: Stationary Process Example- Gaussian White Noise

In other words, a process is stationary if its statistical properties (mean, variance and covariance) do not change over time.

We want our processes to be stationary because it is easier to work with them and fit a model. As its parameters are constant, we can say with high probability that it will follow the same pattern in the future.

The next two sections will discuss and illustrate examples of stationary and non-stationary time series.

1.4.1 White Noise Process

The white noise is a classic example of a stationary process. Also known as a purely random process, the white noise is defined as a sequence $\{\epsilon_t\}$, consisting of *independent random variables* with constant mean ($E\{X_t\} = \mu$) and constant variance ($var\{X_t\} = \sigma^2$).

The fundamental property of a white noise is that no matter how *close* any two of its values are in time, they are statistically *independent*.

There are several types of white noise processes, including those generated by the following distributions: Gaussian, exponential, uniform and truncated Cauchy.

Figure 1.3 shows a white noise plot generated from a gaussian distribution. As it can be noticed, the mean is zero (more precisely, -0.013222; this is due to the fact that we used only 1000 samples), and the standard deviation is 1 (more precisely, 1.003685).

1.4.2 Non-stationary Processes

We have defined a stationary process as a process whose statistical properties are time-independent. Thus, a non-stationary process is a process whose mean, variance and standard deviation do change over time. As a consequence, we will observe trends, seasonal effects or other time-dependent structures in our data; see for reference: [7], [9] [10], [11], [12]

Figure 1.4 illustrates a non-stationary process. It is based on a database which collected data from January 1949 until December 1960, at a monthly rate. [13] It is clear that the number of passengers, although shows some volatility, presents an upward trend, which may

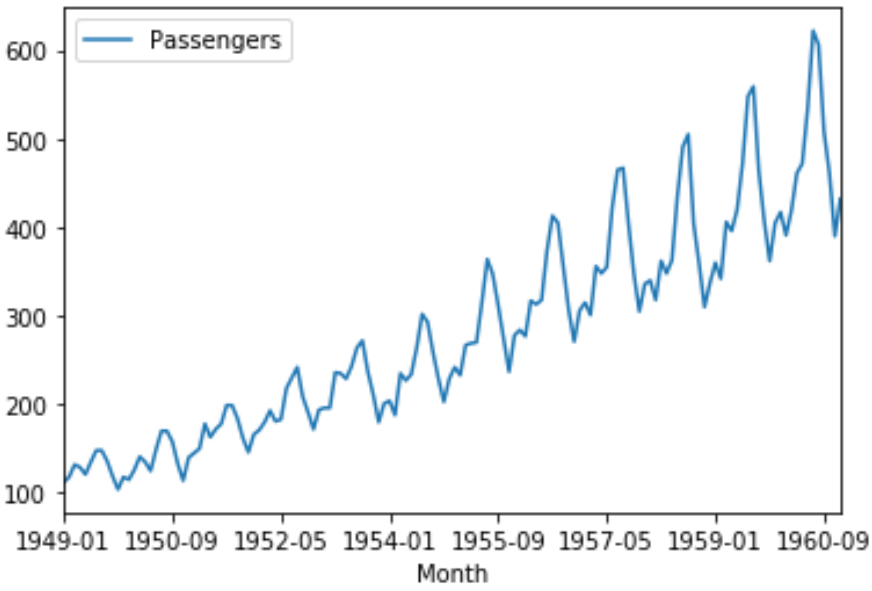

Figure 1.4: Non-Stationary Process Example – Monthly Airline Passengers, 1949-1960

be explained by the growing popularity and affordability of airplane travel. It can also be seen that the mean number of passengers grows over time, as it does the deviation from the mean. We have discussed some of the most important tools which help us characterize a time series. In the next following sections, we will see how time series are modelled.

1.5 Stochastic Processes

1.5.1 Moving Average (MA) Processes

Time series in social sciences often are the result of various underlying factors, events or shocks. When predicting a value in a time series, we often need to take into account other phenomena that could have led to a certain modification in our data.

We define a moving average process as follows:

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} \tag{1.7}$$

The equation 1.7 is called a q^{th} *order moving average process.* $\theta_1, \theta_2, ..., \theta_q$ from the equation 1.7 represent the moving average coefficients, $\epsilon_t, \epsilon_{t-1}, ..., \epsilon_{t-q}$ represent the shocks at time $t, t-1, ..., t-q$, respectively.

When we replace q in equation 1.7 with 1, the resulting process will be a *first order moving average process* or MA(1), that is, the simplest form of a MA process.

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon \theta_{t-1} \tag{1.8}$$

When *q=2*, we get a so called *second-order moving average process.*

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon \theta_{t-1} + \theta_2 \epsilon \theta_{t-2} \tag{1.9}$$

1.5.2 Autoregressive (AR) Processes

Sometimes, values in a time series are dependent on the values that came before them. Hence, having the observations made until the moment t, we can use them to forecast the $t+1$ value in our data set. An autoregressive process is defined as:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} ... + \phi_p X_{t-p} + \epsilon_t \tag{1.10}$$

The equation 1.10 is called a p^{th} *order autoregressive process.* $\phi_1, \phi_2, ..., \phi_p$ from the equation 1.10 are the portions of the previous values carried over to the current value, $X_t, X_{t-1}, ..., X_{t-p}$ are the values of our dataset at time $t, t-1, ..., t-p$, respectively. ϵ_t is an error term.

When we replace p in equation 1.10 with 1, we obtain a *first-order autoregressive process* or AR(1). This happens when the current value is a function of the first previous value only.

$$X_t = \phi_1 X_{t-1} + \epsilon_t \tag{1.11}$$

When $p=2$, we get a so called *second order moving average process.*

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \epsilon_t \tag{1.12}$$

Put differently, this model says that our next observation will be the weighted average of the last q observations. Perhaps the most naive approach when it comes to modeling time series, the autoregressive technique still gives us very useful results. [7],[14]

1.5.3 Autoregressive Moving Average (ARMA) Processes

As a combination of the last two models discussed, we say that series which present both moving average and autoregressive characteristics are called ARMA processes.

A formulation of an ARMA process is presented in Eq. 1.13

$$X_t = \epsilon_t + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \phi_1 X_{t-1} + \phi_2 X_{t-2} ... + \phi_p X_{t-p} \tag{1.13}$$

The eq. 1.13 is designated as ARMA (p,q), because the MA process has the order q and the AR process has the order p.

If we want to model ARMA(1,1), we can write:

$$X_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \phi_1 X_{t-1} \tag{1.14}$$

The ARMA model can be used to model time series data only when the data has a *stationary* behaviour. However, many time series from natural sciences, but especially those from finance and economics are non-stationary, as they present both trends and seasonal patterns. In order to apply a model that suits best this kind of the data, the following section will present the ARIMA processes. [7],[14],[15]

1.5.4 Autoregressive Integrated Moving Average (ARIMA) Processes

Autoregressive Integrated Moving Average (ARIMA) process are a generalised formulation of ARMA processes, which include the case non-stationarity. ARIMA models a non-stationary time series by applying finite differencing of the data points.

In order to define an ARIMA process, we will introduce the **lag operator, L**. The lag

operator L (also known as the backward shift operator), is defined as:

$$LX_t = X_{t-1} \tag{1.15}$$

We write AR(p) processes using the lag operator :

$$\epsilon_t = \phi(L)X_t \tag{1.16}$$

Similarly, MA(q) processes will be:

$$X_t = \theta(L)\epsilon_t \tag{1.17}$$

And ARMA(p,q) processes will have the following form:

$$\phi(L)X_t = \theta(L)\epsilon_t \tag{1.18}$$

Here, $\phi(L) = 1 - \sum_{i=1}^{p} \phi_i L^i$ and $\theta(L) = 1 + \sum_{j=1}^{q} \theta_j L^j$.
We can now define ARIMA(p,d,q) processes using lag polynomials as follows:

$$\begin{gathered} \phi(L)(1-L)^d X_t = \theta(L)\epsilon_t \quad \text{or} \\ \left(1 - \sum_{i=1}^{p} \phi_i L^i\right)(1-L)^d X_t = \left(1 + \sum_{j=1}^{q} \theta_j L^j\right)\epsilon_t \end{gathered} \tag{1.19}$$

In eq. 1.19 $p, d, q \in \mathbb{Z}^+$ and give the order of the autoregressive, integrated and moving average parts of the model respectively. It is worth noting here that the distribution designation $I(d)$, is the order of integration of the model. If $d = 0$ then the series is stationary and we have an ARMA process. If $d = 1$ the series requires first differencing to become stationary and we obtain ARIMA(p,1,q). Once the process has been made stationary, we can advance with the analysis.

ARIMA (p,0,0) is an autoregressive AR(1) process and ARIMA(0,0,1) is a moving average MA(1) process.[7],[14],[15]

1.5.5 Seasonal Autoregressive Integrated Moving Average (SARIMA) Processes

The ARIMA model presented in the last section is for non-seasonal non-stationary data. However, if we wish to generalise it so that it covers also seasonal time series, we can use the Seasonal ARIMA (SARIMA) model. In this proposed model seasonal differencing of a convenient order is used in order to remove non-stationarity from the data. A first order seasonal difference is the difference between a data point and the corresponding data point from the preceding year and is determined as $z_t = X_t - X_{t-s}$. If our data is collected on a monthly basis $s = 12$, and for quarterly time series $s = 4$.
This model is commonly termed as the SARIMA $(p, d, q) \times (P, D, Q)^s$.

The mathematical definition of a SARIMA $(p, d, q) \times (P, D, Q)^s$ model using the lag operator is written as follows:

$$\Phi_P(L^s)\phi_P(L)(1-L)^d(1-L^s)^D X_t = \Theta_Q(L^s)\theta_q(L)\epsilon_t, \quad or$$
$$\Phi_P(L^s)\phi_P(L)z_t = \Theta_Q(L^s)\theta_q(L)\epsilon_t \tag{1.20}$$

where z_t is the seasonally differenced series. [7], [14], [15]

1.5.6 Autoregressive Conditionally Heteroskedastic (ARCH) Processes

In some time series, the variance changes frequently over time. In the context of a time series in the financial domain, this would be called increasing or decreasing volatility. In time series where the variance is incremented in an organized manner, such as an increasing trend, this property of the series is called *heteroskedasticity*. This means that we have a changing or unequal variance across the series. If the change in variance can be corellated over time, then it can be represented using an autoregressive process, such as Autoregressive Conditionally Heteroskedastic (ARCH). Explicitly, an ARCH method models the variance at a time step as a function of the residual errors from a mean process.[16]
Let us consider a time series X_t that has a variance (volatility) that changes over time,

$$X_t = \sigma_t \epsilon_t \tag{1.21}$$

with ϵ_t is a sequence of independent and identical distributed (iid) random variables with zero mean and unit variance. Here, σ_t represents the local conditional standard deviation of the process.
X_t is ARCH(q) if it satisfies equation 1.21 and

$$\sigma_t^2 = \alpha + \beta_{1,q}X_{t-1}^2 + ... + \beta_{q,q}X_{t-q}^2, \tag{1.22}$$

where $\alpha > 0$ and $\beta_{j,q} \geq 0$, $j = 1, ..., q$ (to secure that σ_t^2 is positive).[7]
For example, we can write ARCH(1) as follows:

$$\sigma_t^2 = \alpha + \beta_{1,1}X_{t-1}^2 \tag{1.23}$$

1.5.7 Generalized Autoregressive Conditionally Heteroskedastic (GARCH) Processes

Generalized Autoregressive Conditional Heteroskedasticity, (GARCH), is an extension of the ARCH model which integrates a moving average component together with the autoregressive component.[16] A GARCH model is one where the variance is a function of previous conditional variances as well as previous innovations. The fundamental formulation of a GARCH(p,q) model is [14]

$$\sigma^2 = \omega + \beta_{1,q}X_{t-1}^2 + ... + \beta_{q,q}X_{t-q}^2 + \gamma_{1,p}\sigma_{t-1}^2 + ... + \gamma_{p,p}\sigma_{t-p}^2 \tag{1.24}$$

For instance, GARCH(1,1) is modeled:

$$\sigma^2 = \omega + \beta_{1,1}X_{t-1}^2 + \gamma_{1,1}\sigma_{t-1}^2 \tag{1.25}$$

Chapter 2
Spiking Neural Networks

2.1 Introduction

Artificial Neural Networks (ANNs), are effective computational tools that were influenced by the structure and mechanisms of the human brain. They have been used to solve complicated problems, ranging from function estimation, pattern recognition and classification questions, which could not be solved by using other analytical tools. As our knowledge of the brain and how it refines information was enhancing, ANNs have emerged into more biologically realistic and powerful models, leading to the development of recurrent networks, probabilistic neural networks, dynamic neural networks and many other types of evolved neural networks.

Even though ANNs have experienced numerous stages of evolution, for a long time there were no pursuits of classifying generations of neural networks. One possible explanation would be the difficulty of this task, because there have been multiple directions in which the ANN were developed and to define one advancement more compelling than another would be highly inaccurate. However, we can still find an conceptual advancement that could help us succeed in categorizing these generations, namely the development of the mathematically-defined activation function as the information processing mean of the artificial neuron. Based on this, we can distinguish the following categories:

1. The *first generation* of neurons proposed in the 1940s and 1950s did not exploit the temporal aspect of information processing. These models of neurons fired if their internal state (which is the weighted sum of the inputs of the respective neuron) exceeded a certain threshold. Yet, it was of no significance *when* that threshold was reached. From a biological point of view, this would mean that all inputs to a certain neuron contributed to its internal state at the exactly same moment (were synchronous) and thus could be directly summed. Also, the amplitude of the input did contribut to the external state, but this is not the case for biological neurons.

2. The *second generation* neurons were developed fro the 1950s to 1990s and determined their internal state in an analogous manner to their predecessors. However, unlike the previous generation, these neurons controlled their output using a mathematically defined activation function, generally a smooth sigmoid or radial basis function (RBF), rather than an established threshold value. It became achievable for these neurons to have a real-valued output. Although this model unquestionably more powerful than the one based on the neurons from the first generation and could more complex pattern recognition

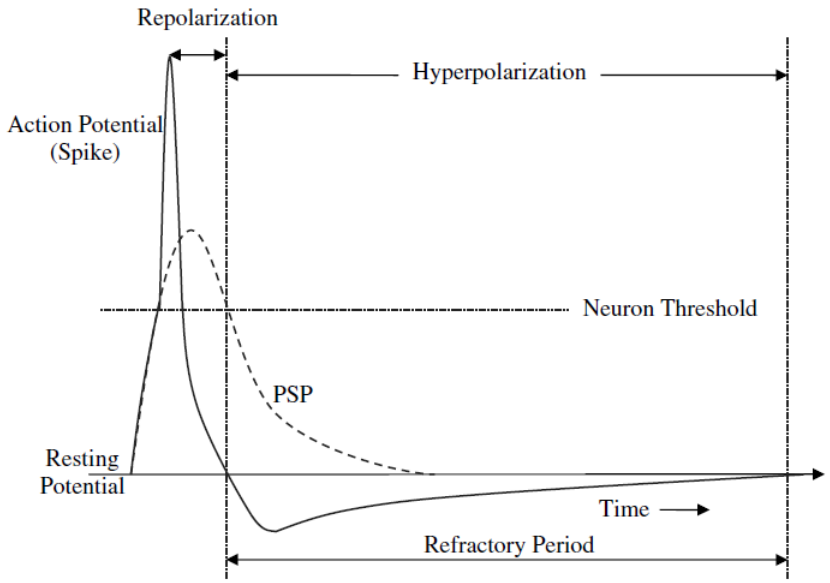

Figure 2.1: The Internal State of a Postsynaptic Neuron

problems (starting with the well-known XOR problem), its computational power still did not achieved its full potential. This was because it did not express the temporal information singular spikes.

3. In the last three decades, to surmount this deficiency, neurons that can interact via the rigorous timing of spikes have been developed and readjusted for ANNs. These neurons have been called spiking neurons. In the literature, these spiking neurons have been cited as *third generation* neurons. Akin to the first generation neurons, spiking neurons act as integrate-and-fire units and have an all or none output behaviour. However, the spiking neuron has an intrinsic dynamic nature defined by an internal state which modifies over time. Every postsynaptic neuron fires a spike or action potential at the time moment when its internal state exceeds the neuron threshold. The magnitude of the spikes (either input or output) encloses no information. This is a characteristic of the biological neurons as well. Instead, all information is contained in the timing of the spikes [17].

As a consequence, spiking neurons and spiking neural networks are of great interest, not only for the machine learning applications, but also because they represent an important stepping-stone in our quest for understanding how the human brain works and implicitly, how to understand certain neuronal behaviours, such as diseases or learning.

2.2 Mechanism of Spike Generation in Spiking Neurons

Esentially, action potentials or spikes from numerous presynaptic neurons reach a postsynaptic neuron and induce postsynaptic potentials (PSPs). The PSP represents the internal state of the postsynaptic neuron caused by the presynaptic spike and depends on synaptic properties such as travel time or delay through the synapse, strength of the synaptic connection, and other biological influences.
A spike train represents a sequence of such spikes. Each spike in the spike train generates a PSP at the moment it arrives at the postsynaptic neuron. In the long run, various presynaptic neurons, each with multiple spikes, produce multiple PSPs. These are temporally integrated to compute the internal state of the postsynaptic neuron as a function of time. The postsynaptic

neuron produces a spike when the integrated internal state exceeds a threshold.

Fig. 2.1 shows the internal state of a postsynaptic neuron after it has received a presynaptic spike. Nonetheless, if the postsynaptic neuron fires, its internal state does not continue to be the same as the PSP. In other words, the timing of the spike from the postsynaptic neuron affects the internal state if the neuron itself. Shorlty after an output spike is produced, the internl state declines sharply. This phase is named *repolarization*. After this phase, the internal state of the neuron will remain at a value smaller than the resting potential of the neuron (Fig. 2.1), and the neuron is in the *hyperpolarization* phase. Subsequently, it will be a challenge for the neuron to reach the threshold and fire again for a determined time period, known as *refractory period* (Fig. 2.1).
The three processes mentioned above are undoubtful evidence of the importance of the timing of the presynaptic spike train for encoding information [17].

2.3 Spiking Neurons

Whenever the membrane potential u crosses some threshold υ, a spike is generated. We consider $t^{(f)}$ to be the time moment when a spike occurs (the *firing time*). This phenomenon can be formally described by the following equation:

$$t^{(f)} : u(t^{t(f)}) = \upsilon \quad \text{and} \quad \left.\frac{du(t)}{dt}\right|_{t=t^{(f)}} > 0 \tag{2.1}$$

In the next following sections, some models of spiking neurons will be presented. We will begin by discussing the Leaky Integrate-and-Fire neuron.

2.3.1 Leaky Integrate-and-Fire Model

As shown in the figure 2.2, the basic circuit of the integrate-and-fire model (on the right side, inside the dashed circle) consists of a capacitor C in parallel with a resistor R. The RC circuit is charged by a current $I(t)$. The current $I(t)$ could be divided into two components: $I(t) = I_R + I_C$. The first component, I_R is the current which traverses the resistor R. Using the Ohm's law we get $I_R = u/R$, where u is the voltage between the two black dots, across the resistor. The second current (I_c) charges the capacitor C. If the electrical charge is denoted by q and u is the notation for the voltage, from the definition of the capacity we have $C = q/u$. The capacity current will then be $I_C = C\mathrm{d}u/\mathrm{d}t$.

Thus,

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \tag{2.2}$$

If we multiply Eq. (2.2) by R and denote $\tau_m = CR$ to be the time constant of the *leaky integrator*, it yields us the standard form:

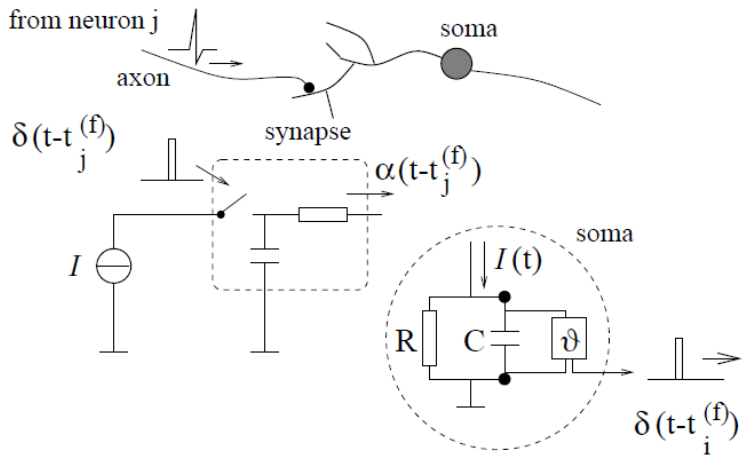

Figure 2.2: Schematic diagram of the integrate-and-fire model

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \tag{2.3}$$

We mention that u is the membrane potential and τ_m is the membrane time constant of the neuron. In integrate-and-fire models, spikes are formal events characterized by the firing time $t^{(f)}$, which is defined by a threshold criterion:

$$t^{(f)} : u(t^{t(f)}) = \upsilon \tag{2.4}$$

Immediately after $t^{(f)}$, the potential is reset to a new value $u_r < \upsilon$,

$$\lim_{t \to t^{(f)}; t > t^{(f)}} u(t) = u_r \tag{2.5}$$

For simplicity, u_r is usually considered to be 0 (the potential when the neuron is in a resting state has the value 0).

After defining the neuron, we want to see how it will behave under certain circumstances, that is, how it will react when presented to certain stimuli.

First, let's consider that our LIF neuron is stimulated by a **constant input current** $I(t) = I_0$. Under the assumption that $u_r = 0$, we can find the behaviour of the membrane potential by integrating the equation 2.3:

$$u(t) = RI_0 \left[1 - exp\Big(- \frac{t - t^{(1)}}{\tau_m} \Big) \right] \tag{2.6}$$

where $t^{(1)}$ is the time when a spike has occurred. When $t \to \infty$, the membrane potential (2.2) approaches the asymptotic value of RI_0. If $RI_0 < \upsilon$, no further spike could appear, as the threshold value is never reached. However, if $RI_0 > \upsilon$, the membrane potential reaches the

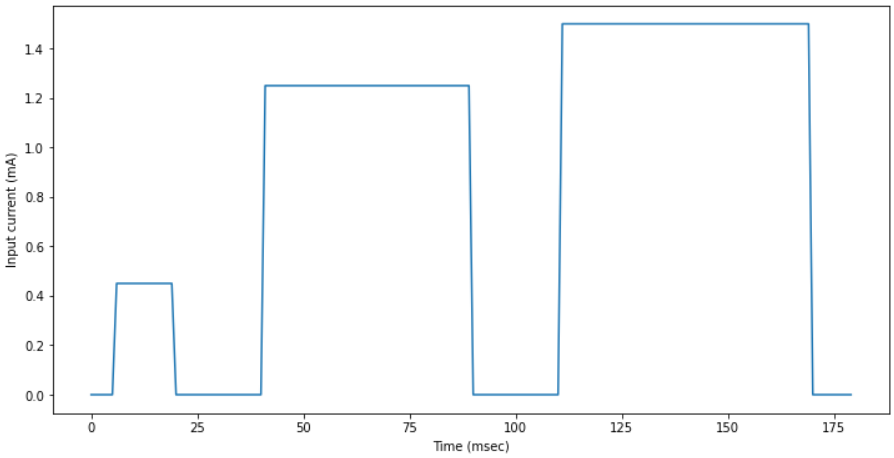

Figure 2.3: Three Constant Input Currents

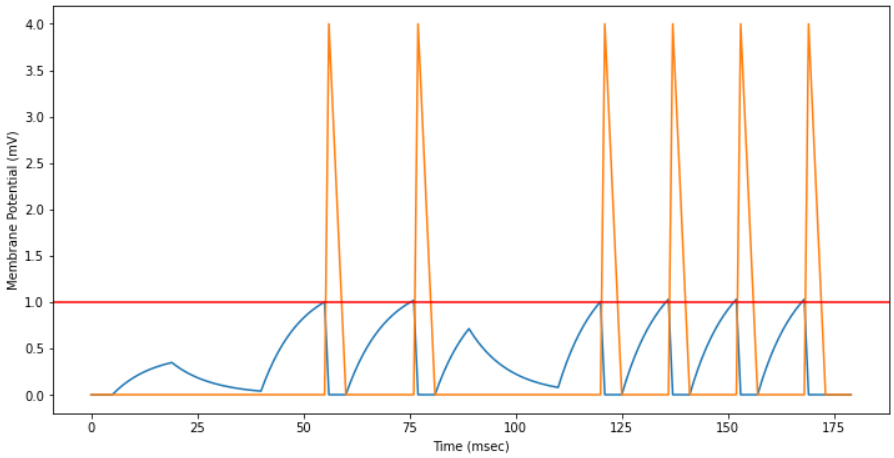

Figure 2.4: The LIF Neuron Response

threshold value υ at time $t = t^{(2)}$. The threshold value υ can be found from the condition $u(t^{(2)}) = \upsilon$ or

$$\upsilon = RI_0\left[1 - exp\Big(-\frac{t^{(2)} - t^{(1)}}{\tau_m}\Big)\right] \tag{2.7}$$

If we want to find the time between the occurrence of two consecutive spikes, we solve for $T = t^{(2)} - t^{(1)}$:

$$T = \tau_m ln\frac{RI_0}{RI_0 - \upsilon} \tag{2.8}$$

We conclude that for a constant input current I_0, the time between two consecutive spikes which are produced is constant, so the spiking period will be T (given by eq. 2.8).

To illustrate the aforementioned behaviour, a LIF neuron was created using the Python Tensorflow library. Three constant currents with the intensity of $0.45mA$, $1.25mA$ and $1.5mA$ respectively, were fed to our neuron (fig 2.3).

Fig. 2.4 shows the LIF neuron respone to the above input currents. The red line highlights the threshold value *upsilon*, which in our case was initialized with the value 1. The blue line shows the trajectory of the membrane potential, $u(t)$. At time $t = 5ms$, when first current is fed to the neuron, the membrane potential begins to increase. However, due to the low intensity of the current, the membrane potential has a slow growth and because of the short time if the input, the membrane potential does not reach the threshold during the first stimulus. At the time $t = 20ms$, when the first stimulus ceases to exist, the membrane potential begins to decrease until the momtent $t = 40ms$, when the second stimulus is applied. Then, it increases again until it reaches the threshold value (until it meets the red line). At that moment, a spike is produced (the orange line) and the membrane potential (the blue line) is set to resting value. The process repeats itself as long as the current has a positive value.

Now let's analyze what happens when the stimulus is a **time-dependent current**.

Suppose we have an input current $I(t)$, which changes over time and a spike has happened at the moment $t = \hat{t}$. Integrating the equation 2.2 we get the trajectory of the membrane potential, $u(t)$:

$$u(t) = u_r exp\left(- \frac{t - \hat{t}}{\tau_m} \right) + \frac{1}{C} \int_0^{t-\hat{t}} exp\Big(- \frac{s}{\tau_m} \Big) I(t - s) ds \tag{2.9}$$

This expression gives the membrane potential behaviour for $t > \hat{t}$ and is valid until the potential reaches the threshold value, when the $u(t)$ is reset to u_r and the integration restarts.

Figure 2.5 shows a varying current which corresponds to a normal distribution of 1.5 mA and standard deviation 1.0 mA. Now we stimulate our neuron with this current and its respons can be seen in figure 2.6. With the red line is represented the threshold value, which in our case is equal to 1.0 mA. When the current is fed to our neuron, the membrane potential(the blue line) value increases, but it shows a *random* or *varying* growth, until it reaches the threshold value. Then, a spike is emited (the orange line) and the membrane potential is set to the resting value, u_r. After that, the process repeats itself, as long as there is a stimulus (current). Each spike is separated from the next one by a resting period, defined when our neuron was initialized.

So far there was considered an isolated neuron that was stimulated by an external current $I(t)$. However, in a more realistic approach, there is a larger network where the integrate-and-fire model is part of and the input current $I(t)$ is generated by the **postsynaptic activity** of other, presynaptic neurons.

In the context of the integrate-and-fire model, each presynaptic spike generates a postsynaptic pulse. More precisely, if the presynaptic neuron j has emitted a spike at the moment $t_j^{(f)}$, the postsynaptic neuron i 'feels' the current with time course $\alpha(t - t_j^{(f)})$ (fig 2.2). The total input which enters the neuron i is the weighted sum over of current pulses, from all presynaptic neurons:

$$I_i(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)}) \tag{2.10}$$

The factor w_{ij} is a measure of the effectiveness of the synapse from neuron j to neuron i.

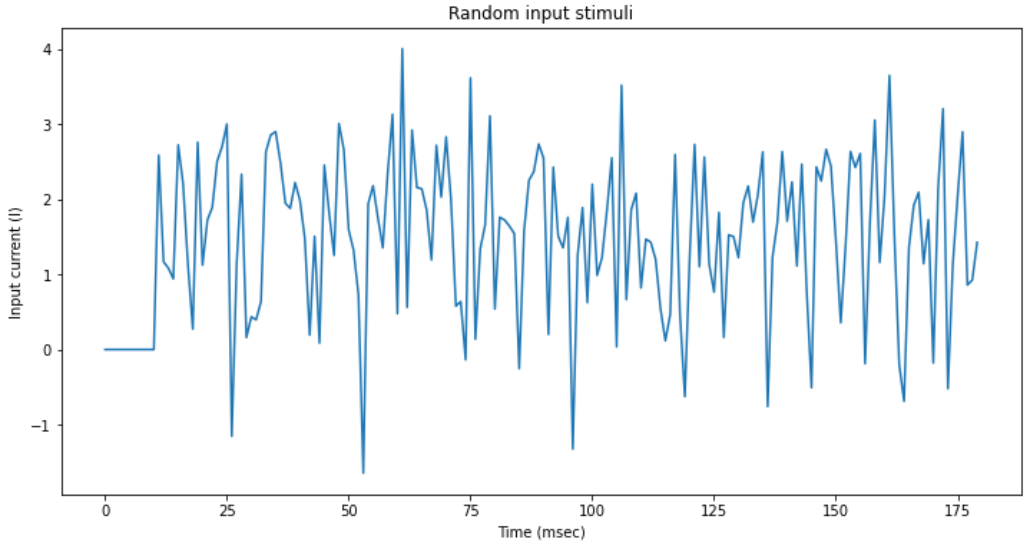

Figure 2.5: Time-dependent (random) Input Current

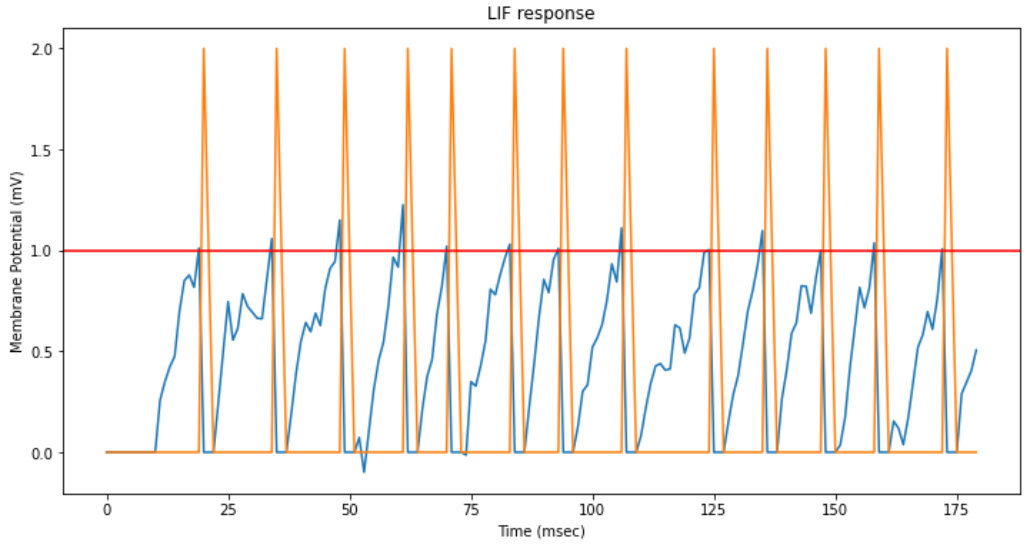

Figure 2.6: The LIF Neuron Response

The time course from eq. 2.10 could be defined in multiple ways, but the simple choice is a Dirac δ-pulse, $\alpha(t) = q\,\delta(t)$. Here, q is the total charge injected in a postsynaptic neuron via a synapse with efficacy w_{ij}=1. In a more realistic scenario, the postsynaptic current α should have a finite durations; for example, this is the case of an exponential decay with time constant τ_t,

$$\alpha(t) = \frac{q}{\tau_t} exp\Big(- \frac{s}{\tau_t}\Big)\,\Theta(t) \tag{2.11}$$

It is worth mentioning that the Θ from the equation 2.11 is the Heaviside step function with $\Theta(t)$=1 for t¿0 and $\Theta(t) = 0$ elsewhere.
We can now make sense of the rest of the figure 2.2: the output pulse (spike) $\delta(t-t_j^{(f)})$ generated fron the presynaptic neuron j enters the synapse (w_{ij}). After passing through the synapse where it is low-pass filtered, it generates and input current pulse $\alpha(t-t_j^{(f)}$, which is then multiplied by

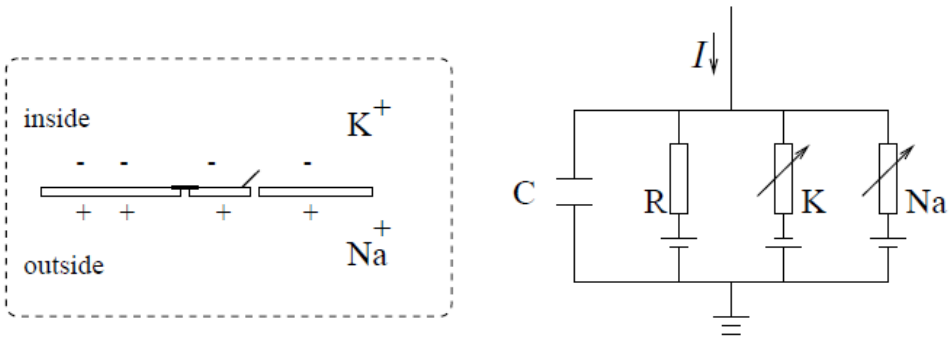

Figure 2.7: Schematic Diagram of the Hodgkin-Huxley Neuron Model

its weight. The resulting current will charge the RC circuit and will generate a spike $\delta(t - t_i^{(f)})$ *if* the membrane potential is equal to the threshold value: $u(t) = v$.

2.3.2 Hodgkin-Huxley Model

In 1952 Hodgkin and Huxley modelled the biochemical activity that occurs in the brain using differential equations. Several years later, they won the Nobel Prize for their revolutionary work. After doing experiments on the axon of the squid, they found three types of ion current: sodium (Na^+), potassium (K^+) and a leak current that is mainly made of Cl^- ions.

Figure 2.7 helps us understand their model: the interior of the cell and the extracellular liquid are separated by a semipermeable cell membrane that acts as a capacitor. When an input current $I(t)$ is injected into the circuit it may add further charge on the capacitor, or flow through the ion channel into the cell membrane.
The above considerations could be described using mathematical equations as follows: We know from the conservation of the electric charge on the membrane, that the current $I(t)$ may divide into a capacitive current I_C which will charge the capacitor C and other component I_k which will penetrate through the ion channels.
Therefore, we can write

$$I(t) = I_C(t) + \sum_k I_k(t) \tag{2.12}$$

where k is the index of the ion channel and the sum takes into consideration all the ion channel of the cell membrane.
As we can see from the Figure 2.7, there are three types of channel: a sodium channel (Na), a potassium channel (K) and a leakage channel with resistance R. Because there is always a movement of the ions from the inside to the outside of the cell and vice-versa, there is a difference of the concentration. This difference of concentration from the interior of the cell to the extracellular liquid generates the Nernst potential and is represented by the batteries.
The definition of the capacity (C) states that if we have a charge Q and a voltage u across the capacitor, then $C = Q/u$. Thus, the current I_C which charges the capacitor is $I_C = C\ \mathrm{d}u/\mathrm{d}t$.
If we replace the above in 2.12 we get

$$C\frac{du}{dt} = -\sum_k I_k(t) + I(t) \tag{2.13}$$

x	E_x	g_x
Na	115 mV	120 mS/cm^2
K	-12 mV	36 mS/cm^2
L	10.6 mV	0.3 mS/cm^2

Table 2.1: The Empirical Values of the Parameters E and g in the Hodgkin-Huxley Equations

x	$\alpha_x(u/mV)$	$\beta_x(u/mV)$
n	(0.1-0.01u)/[exp(1-0.1u)-1]	0.125 exp(-u/80)
m	(2.5-0.1u)/[exp(2.5-0.1u)-1]	4 exp(-u/18)
h	0.07 exp(-u/20)	1/[exp(3-0.1u)+1]

Table 2.2: The Empirical Values of the Parameters α and β in the Hodgkin-Huxley Equations

In eq. 2.13 u represents the voltage across the membrane and $\sum_k I_k(t)$ is the sum of all ionic currents that pass through the cell membrane.
The three types of channel could be described by their resistance, but also by their conductance. The conductance of the leakage channel, g_L, is voltage-independent: $g_L = 1/R$. However, the other two channels have conductances that are dependent of time and voltage. Let g_{Na} and g_K be the maximum conductance of the sodium and the potassium channels, respectively. When all the channels are open, they transmit current with g_{Na} and g_K. In reality, only some of the channels are open, while some are blocked. The additional variables m, n and h describe the probability that a channel is open. The Na^+ gates are controlled by m and h, while n controls the K^+ channels.
Hodgkin and Huxley came up with the following mathematical formulation:

$$\sum_k I_k(t) = g_{Na} m^3 h(u - E_{Na}) + g_K n^4 (u - E_K) + g_L(u - E_L) \tag{2.14}$$

In eq. 2.14, E_{Na}, E_K and E_L are reversal potentials. Conductances and reversal potentials are empirical values. In Table 2.1 are given the original values discovered by Hodgkin and Huxley based on experimental behaviour of neurons. The values presented here are those found by considering a resting potential of 0 mV. However, today the resting potential accepted is -65 mV, so the scale has to be shifted by that value. For instance, the corrected value of E_K is -77 mV.
The variables m, n and h are called gating variables (because they give us information about the state of a gate). They evolve as function of u and their behaviour is given by the following differential equations:

$$\begin{aligned} \dot{m} &= \alpha_m(u)(1 - m) + \beta_m(u)m \\ \dot{n} &= \alpha_n(u)(1 - n) + \beta_n(u)n \\ \dot{h} &= \alpha_h(u)(1 - h) + \beta_h(u)h \end{aligned} \tag{2.15}$$

where $\dot{m}$ =dm/dt, $\dot{n}$ =dn/dt and $\dot{h}$ =dh/dt. In Table 2.2 are given the function α and β, that are empirical functions of u. They were found by Hodgkin and Huxley by trying to fit the data of the axon of the squid.
the Hodgkin-Huxley neuron model is defined by the equations 2.13, 2.14 and 2.15, together

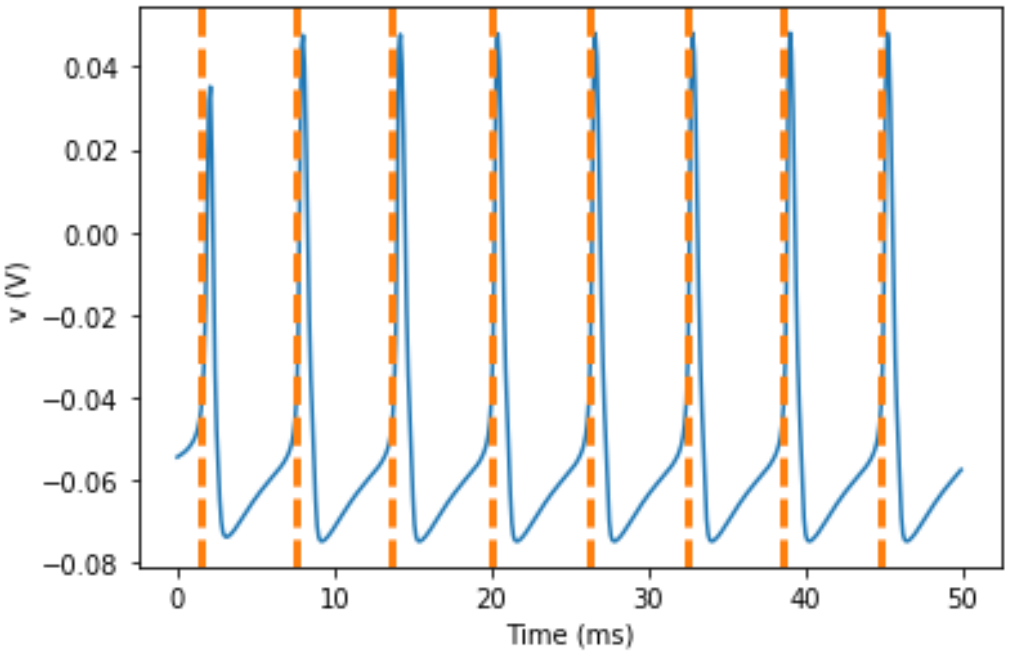

Figure 2.8: Hodgkin-Huxley Neuron Response to a Constant Input Current

with the values presented in Table 2.1 and Table 2.2. [18]

To illustrate the Hodgkin-Huxley neuron model behaviour, a simulation using the library Brian2 was performed. Figure 2.8 shows the response of the neuron to a constant input current of 0.5 nA. By giving the neuron a greater input current, it would have a greater rate of spikes (the spike period would have a smaller value). Similarly, if the input current was smaller, the neuron would have take longer to spike. The simulation time was set to 50 ms.
With the blue line is represented the voltage across the membrane and with orange are represented the spikes. As it can be seen, the voltage increases until the value of -40 mV, a spike is produced and the potential returns to its resting potential (-65 mV). -40 mV is the threshold value. As long as there is an input current, this process will repeat.

Despite the biochemical accuracy of the Hodgkin-Huxley neuron, this model is very computationally expensive and thus inefficient. In the next section we will explore a simpler neuron model.

2.3.3 Izhikevich Model

On the quest of finding a trade-off between the biologically realistic Hodgkin-Huxley neuron and the computationally efficient integrate-and-fire neuron, Eugene M. Izhikevich proposed a model that satisfies both requirements [1].

This model can describe the behaviour of a real neuron in an elegant way, by a system of two-dimensional differential equations as follows:

[1]Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com

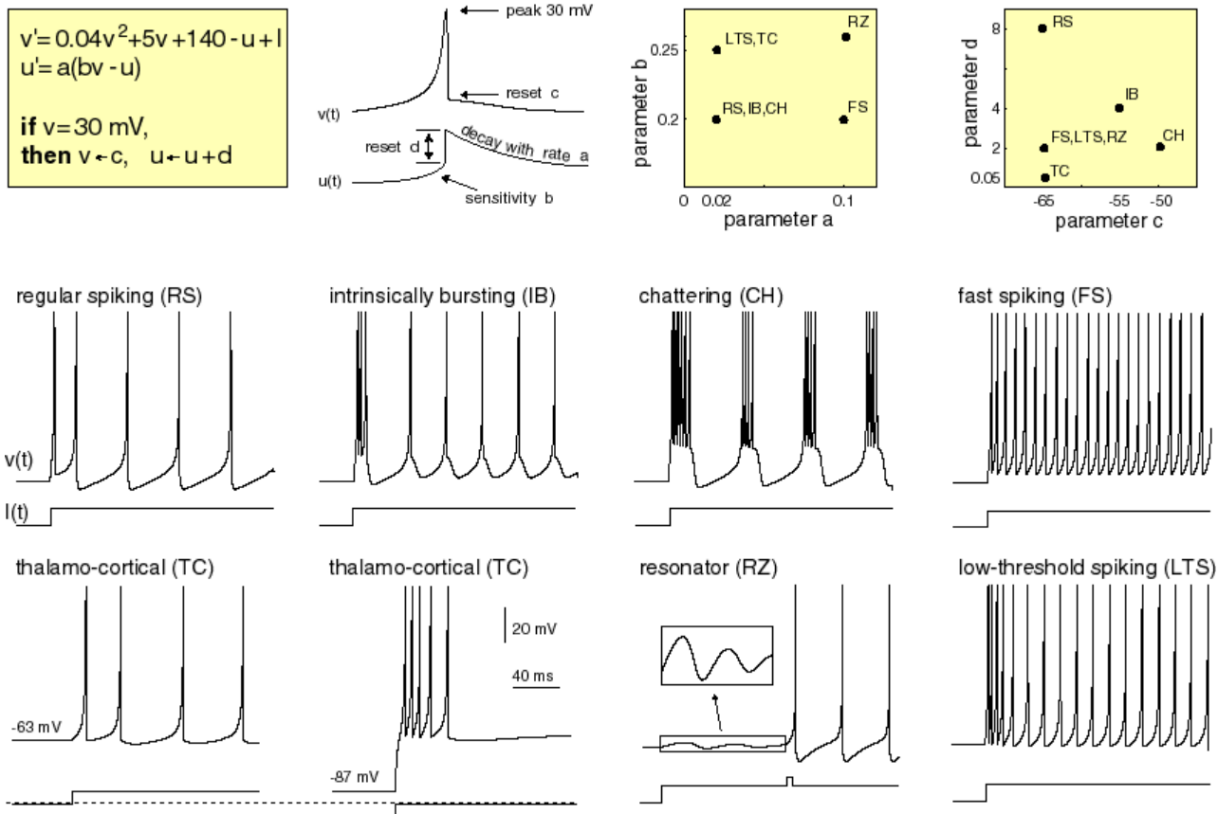

Figure 2.9: Izhikevich Model Applied to Known Types of Neurons

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.16}$$
$$u' = a(bv - u) \tag{2.17}$$

Also, there are the following after-spike resetting statements:

$$\text{if} \quad v \geq 30\,mV, \quad \text{then} \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.18}$$

The parameters a, b, c, d from the above equations are dimensionless constant, and the variables v and u are also dimensionless. v' is the derivative of the membrane potential of the neuron with respect to time. and u' is the derivative of the membrane recovery potential with respect to time. After the membrane potential reaches the threshold value (30 mV), the two variables are reset (eq. 2.18). The current I from eq. 2.16 could be an external current (injected), or coming from other neurons (synaptic current). There are various possibilities of choosing the paramenters of this model, and their significance is presented below:

- a is a constant that gives information about how long it will take for the membrane to recover. If we choose a smaller a, the recovery will be slower. Similarly, for large values of a there will be less time needed for this to happen. An usual value is $a = 0.02$.

- How sensitive the recovery variable is to the fluctuations that take place under the threshold of the membrane potential is described by the variable b. If b has a large value then u and v are strongly correlated and thus more subthreshold oscillations may happen. Greater values of b also mean that spikings occur at a lower threshold value,

resulting in a higher firing rate. A typical value is $b = 0.2$.

- The parameter c, as we could imagine from eq. 2.18, is the reset value which the membrane potential takes after a spike. Typically, this value is -65 mV.
- d gives information about the after-spike reset of the recovery variable u and usually $d = 2$.

However, the choice of the aforementioned parameters is made assuming that we want to model a specific type of neurons. In Fig. 2.9 can be seen that by choosing a specific combination of values for a and b, we obtain different types of neurons. For example, if we choose $a = 0.02$ and $b = 0.25$ we can model a low-threshold spiking (LTS) or a thalamo-cortical (TC) neuron. Similarly, for different compilation of parameters c and d we can model various neurons.
We can classify the neurons in the mammalian brain by their spiking and bursting patterns.

In the following paragraph will be presented the neuron types illustrated in Fig. 2.9. The **excitatory** cortical cells are split into the following categories:

- *RS (regular spiking)* neurons are the most common type in the cortex. When we apply a constant DC current as shown in Fig. 2.9, at the beginning there are a few spikes with a small period of repetition. After some time, The period becomes larger, so the firing rate decreases. This phenomenon is referred to as spike frequency adaptation. If a current with a greater amplitude is injected into the neuron, the interspike frequency will increase. RS neuron are modeled here with the following parameter values: $a = 0.02$, $b = 0.2$, $c = -65$ and $d = 8$.
- *IB (intrinsically bursting)* neurons present a burst of spike at the beginning of the action of the injected current. Then, their spiking patter becomes normal. The corresponding values on the model are $a = 0.02$, $b = 0.2$, $c = -55$(this means that after a spike the voltage reset is high) and $d = 4$.
- *CH (chattering)* neurons can fire bursts of spikes separated by some period of time . The inter-burst frequency can reach values up to 40 Hz. The parameter values for this type of neurons are $a = 0.02$, $b = 0.2$, $c = -50$ (this represents a very high reset voltage) and $d = 2$.

Besides excitatory cells, there are also **inhibitory** cortical cells. Two different kinds of inhibitory neurons exist:

- *FS (fast spiking)* neurons produce spikes with very high frequency, without slowing down, as it can be seen in Fig. 2.9. This neurons have the model parameters: $a = 0.1$ (this means they have a fast recovery), $b = 0.2$, $c = -65$ and $d = 2$.
- *LTS (low-thershold spiking)* neurons also emit spikes with high frequency, but unlike FS neurons, they do show a frequency adaption (The time period between the spikes increases after the initial spikes). The LTS neurons emit spikes at a low threshold. The corresponding model parameters are $a = 0.02$, $b = 0.25$, $c = -65$ and $d = 2$.

Apart from the aforementioned neocortical neurons, the Izhikevich model can also exhibit the behaviour of the neurons which deliver the most input to the cortex, namely the *thalamo-cortical (TC)* neurons Fig. 2.9. The model can reproduce other interesting dynamics, as *RZ (resonator)* neurons.
The function $v' = 0.04v^2 + 5v + 140 - u + I$ in eq. 2.12 was chosen to fit various types of neurons.

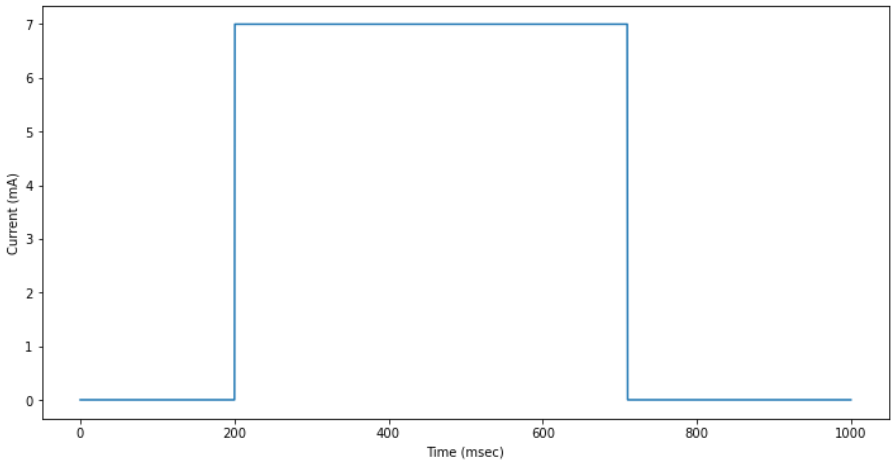

Figure 2.10: Step Input Current

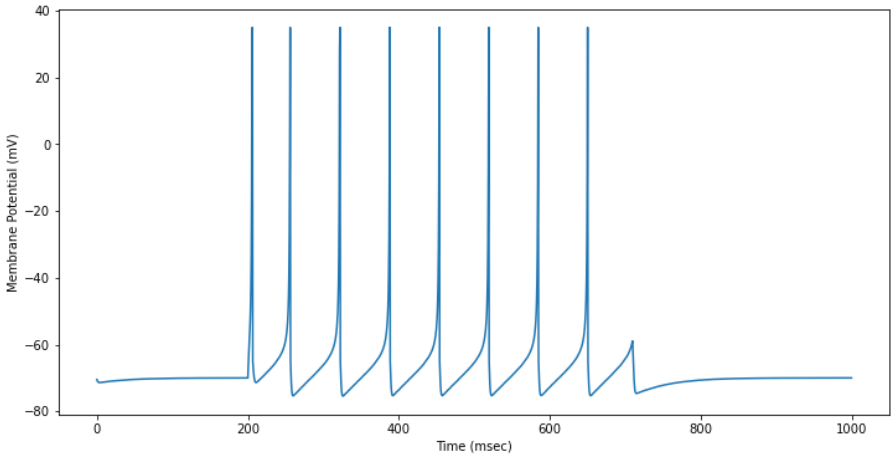

Figure 2.11: Izhikevich Neuron Model Response

As large networks are usually made by multiple kinds of neurons, a "one-fits-all" equations is needed. However, if we are interested to simulate the behaviour of only a specific neuron type, other equation and parameter values might suit better. [19], [20]

To exemplify the Izhikevich model, a simulation using Tensorflow was made. The model in our program had the following paramters: $a = 0.02$, $b = 0.2$, $c = -65$ and $d = 8$. From these values we would expect that our neuron will behave as a regular spiking (RS) neuron. The neuron was injected with a step input current with amplitude 7 A (Fig. 2.10). Indeed, the neuron produced spikes with a higher frequency at the beginning, followed by regular spikes with a lower firing rate. When the current was eliminated, the membrane potential stopped growing and did not reach its threshold. As a result, a spike was not produced and the membrane potential returned to its resting value.

2.4 Spiking Neural Networks

Straightforwardly, SNNs are netwotks of spiking neurons. The SNN architecture is generally identical to that of a traditional ANN.

Figure 2.12 shows an example of a SNN architecture. Contrary to traditional feedforward ANNs in which two neurons are connected by just one synapse, the connection between two SNN neurons is modeled by various synapses, as it can be seen in Fig. 2.12(b). A neuron from

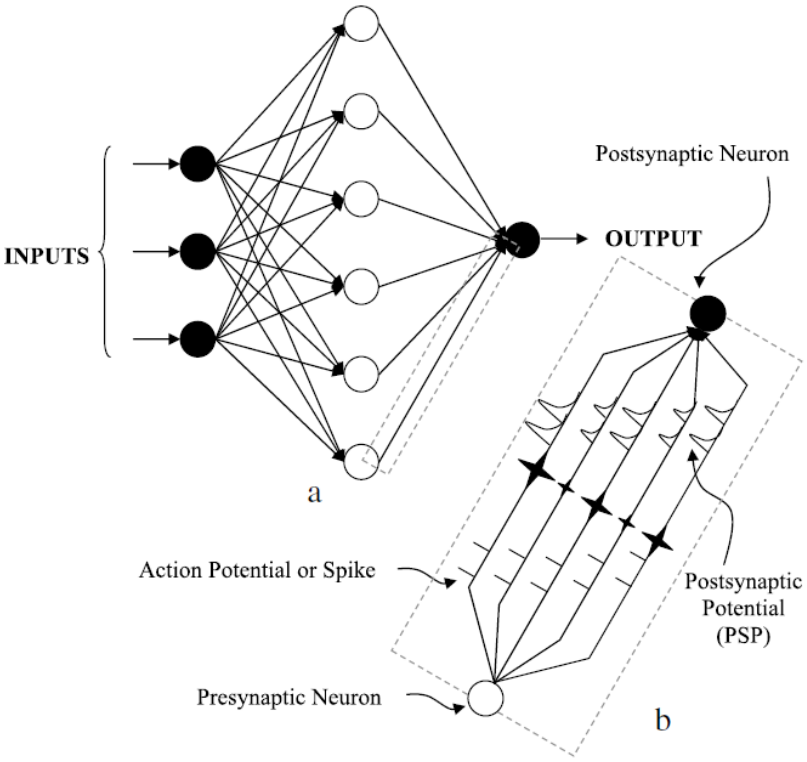

Figure 2.12: (a) Spiking Neural Network Architecture (b) Multiple Synapses Transmitting Multiple Spikes From A Presynaptic Neuron To A Postsynaptic Neuron
[21]

this network has the ability to assimilate multiple input spikes from presynaptic neurons and fire multiple output spikes as a response. Specifically, information conducted from one neuron to the next is encoded in the form of a spike train rather than a single spike. The enhanced connection which can be seen in Fig. 2.12(b) exhibits the temporal sequence of spikes (short vertical lines) from the presynaptic neuron, the synaptic wights (the weight are greater if the size of the star unit in the center is greater) and the resulting post-synaptic potential (proportional to the size of the PSP, as shown in figure). Therefore, a neuron $j \in \{1, 2, ..., N_l\}$ in layer l is postsynaptic to N_{l+1} presynaptic neurons, where N_l is the number of neurons in layer l. Each presynaptic neuron $i \in \{1, 2, ..., N_l\}$ is connected to the postsynaptic neuron j via K synapses. The number K is constant for any two neurons [17, 21].

2.5 Learning Rules in Neural Networks

2.5.1 Synapses

The synapse is where the axon of a presynaptic cell and the dendrite (or soma) of postsynaptic neuron are connected. Because most of the input currents in the brain come not from external injected stimuli but from other neurons, they have to somehow make contact [18].
In Fig. 2.13 two neurons, Neuron 0 and Neurons 1, are connected through a synapse. Only Neuron 0 is injected with an external current, I. If the two neurons were not connected with the synapse, the potential v of Neuron 1 would have remained 0 on the entire simulation period. The link between the two neurons is defined as follows: When Neuron 0 fires a spikes (every 10 ms), the potential v of Neuron 1 increases by 0.25. This happens for every spike of Neuron 0 until the value of the potential v of Neuron 1 reaches the threshold 1. At that moment, Neuron

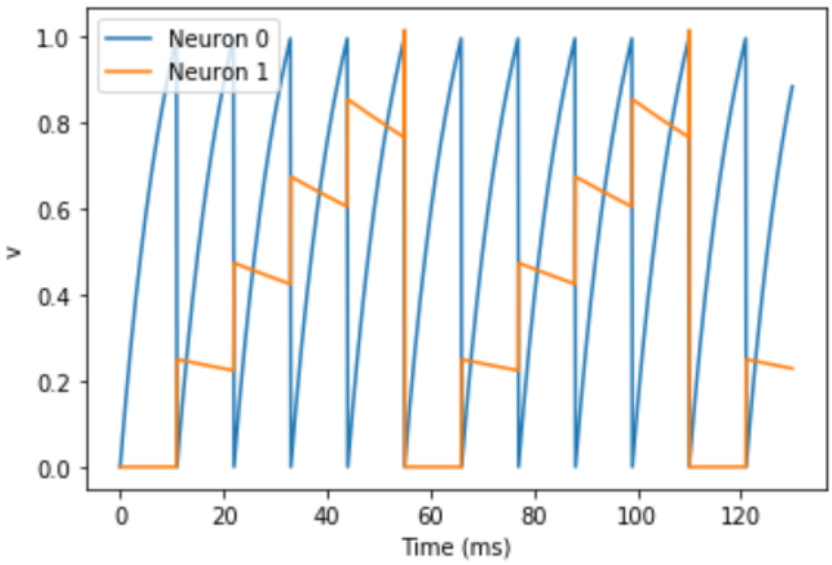

Figure 2.13: Example of Synaptic Behaviour

1 fires a spike and its potential v returns to 0, waiting for Neuron 1 to spike again.

In the above example, the synapse rule was specified explicitly. However, for large neural networks, as it is the case of most real applications and in biological networks, this is not usually possible [22]. For this reason, the next sections will present two of the most widely-known learning rules in neural networks.

2.5.2 Hebbian Learning

The formal theory of neural networks states the the weight w_{ij} of a link between neuron j and neuron i is a parameter that can modify its value over time in order to increase the performance of a network while performing a task.
Learning is defined as the process by which the parameters adapt their values, and the method by which the weights are adjusted is called a *learning rule*.

Hebb's postulate describes how the connection between a presynaptic neuron j and a postsynaptic neuron i should be altered: "When an axon of cell j is near enough to excite cell i or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that j's efficiency, as one of the cells firing i, is increased." In other words, the adjustment of the weight w_{ij} between neuron j and neuron i depends only of the states of the two neurons j and i, and the present value of w_{ij}, but not on the states of other neurons k. Hebb also wrote that if two cells are simultaneously active multiple times, they will move toward an association between them, so that an activity v_j in one of them promotes activity v_i in the other. Based on the above we can define a general mathematical formulation of Hebb's postulate:

$$\frac{d}{dt}w_{ij} = F(w_{ij}; v_i, v_j) \tag{2.19}$$

There are many other formulaic descriptions of Hebbian learning. For instance,

$$w_{ij} = x_i x_j \tag{2.20}$$

is also used to characterize the synaptic efficacy w_{ij} between neurons j and i. In eq. 2.20, x_i and x_j is the inputs of neurons i and j respectively. What this means for binary neurons (their states can be only 0 and 1) is that only when neuron i is active *and* neuron j is active, the weight w_{ij} would be set to 1.
Another formula used to describe Hebb's Rule is the following:

$$w_{ij} = \frac{1}{p} \sum_{k=1}^{p} x_i^k x_j^k \tag{2.21}$$

In eq. 2.21 p represents the number of training patterns, x_i^k is the k^{th} input for neuron i. This type of learning is called learning by epoch, that is when the weights are updated only *after all the training examples* are disclosed.

Hebb's Learning Rule is regularly generalised as follows:

$$w_{ij}[n+1] = w_{ij}[n] + \eta x_i[n] x_j[n] \tag{2.22}$$

What this means is that the efficacy of a synapse from neuron j to neuron i at the $(n+1)^{th}$ step, $w_{ij}[n+1]$ is equal to the efficacy of the synapse at the n^{th} step plus the product between the output of neuron j at the n^{th} step and the input of neuron i at the n^{th} step, multiplied by a learning rate, η.

To summarise what Hebb concluded about the firings of the neurons, we will say that if a synapse participates repeatedly in firing a postsynaptic neuron, the efficacy of that synapse will rise. However, we often over-simplify the rule by saying that *neurons that wire together, fire together*. This simplification is not accurate because if two neuron fire simultaneously (at the exact same tame), then we cannot have a causality relationship between the two neurons. To have this cause-effect relationship (a presynaptic neuron to contribute in the firing of a postsynaptic neuron), the presynaptic neuron has to spike just before the postsynaptic neuron fires. This temporal dependency led to a new theory, which explores how the timing of the spikes influences the weights adjustment and how the changes of the weights affect the timing of the spikes [18, 23, 24].

2.5.3 Spike-Time Dependent Plasticity

Spike-Time Dependent Plasticity (STDP) is a form of Hebbian Learning that takes into account the temporal relationships between the spikes produced by the presynaptic and postsynaptic neurons. STDP is also believed to underlie the learning process, the storage of information and the ontogenesis of neural paths in the human brain. In this theory, presynaptic spikes that arrive many times to a postsynaptic neuron just a few milliseconds before it fires a spike, lead to Long-Term Potentiation (LTP) of the synapses which carried those spikes. On the other hand, if a synapse repeatedly transmits spikes to a postsynaptic neuron just after it fired a spike, the synapse will suffer Long-Term Depression (LTD).
The STDP function is defined as the modification of the efficacy of a synapse plotted as a function of the time difference between the presynaptic and postsynaptic action potentials . The STDP function may be different for particular synapse types.

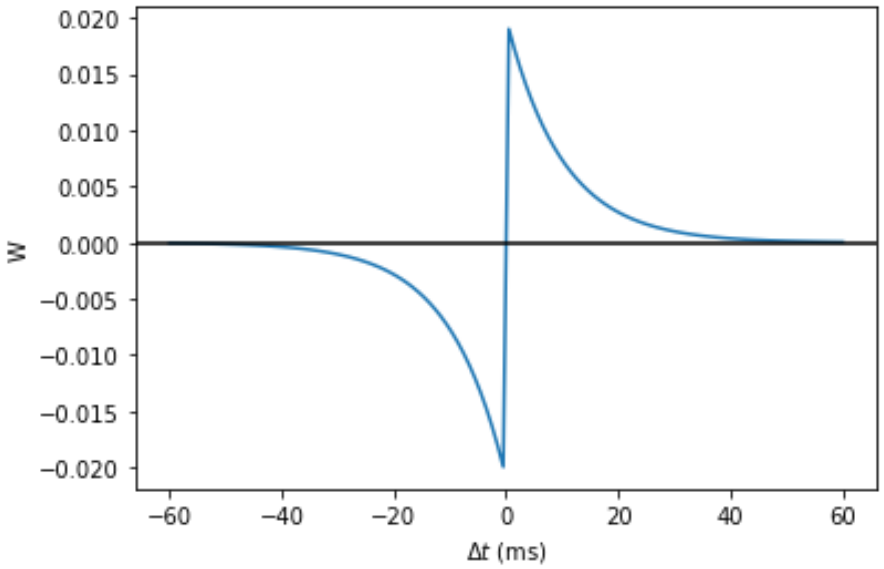

Figure 2.14: STDP function

Let us write mathematically the definition of STDP. If we denote the change of weight of particular synapse from a presynaptic neuron j by Δw_j, then Δw_j will depend on the relative timing between the spike that arrives from the presynaptic neuron j and the firing of postsynaptic neuron i. If t_j^f are the time moments when the spikes f, with $f = 1, 2, ..., N$ arrive at the synapse j and t_i^n are time moments when the postsynaptic neuron i fires the spike n, with $n = 1, 2, ..., N$, then we can conclude that

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f) \tag{2.23}$$

where $W(x)$ stands for one the various STDP functions (they are also named *learning windows*). A favored STDP function $W(x)$ is the following:

$$\begin{aligned} W(x) &= A_+ exp(-\frac{x}{\tau_+}) \qquad for\ x > 0 \\ W(x) &= -A_- exp(\frac{x}{\tau_-}) \qquad for\ x < 0 \end{aligned} \tag{2.24}$$

The eq. 2.24 was used in fits to experimental datasets and models. The time constants τ_+ and τ_- are of order of 10 ms. The parameters A_+ and A_- may vary depending on the actual value of the efficacy, w_j.

Figure 2.14 illustrates the SDTP function $W(x)$ defined in eq. 2.24, with τ_-=τ_+=10 ms, A_+=0.02 and A_-=0.021.

Simulating STDP using directly these equations would be highly ineffective, since the sum goes over all the spike pairs. Besides, this is not physiologically conceivable because a neuron cannot recall all its preceding spike times. For that reason, we will present a more productive and physiologically plausible method for achieving the same results.

We can use the following assumptions: each presynaptic spike t_j^f (Fig. 2.15) leaves behind a trace $x_j(t)$ which decays exponentially if there no spikes and is updated by an amount $a_+(x_j)$,

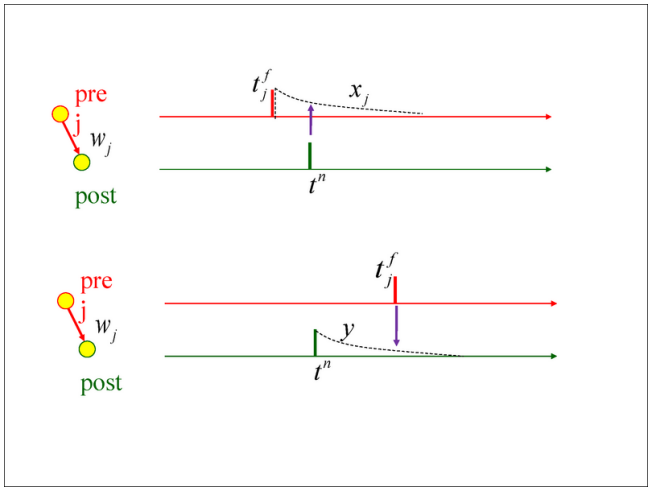

Figure 2.15: STDP with Local Variables

as follows:

$$\tau_+ \frac{dx_j}{dt} = -x_j + a_+(x_j) \sum_f \delta(t - t_j^f) \tag{2.25}$$

Similarly, each spike fired by a postsynaptic neuron (a postsynaptic spike), leaves a trace y (Fig. 2.15), which is increased by an amount $a_-(y)$ when the neuron fired and falls exponentially in the absence of postsynaptic spikes:

$$\tau_- \frac{dy}{dt} = -y + a_-(y) \sum_n \delta(t - t^n) \tag{2.26}$$

Consequently, the weight change will have the following formula:

$$\frac{dw_j}{dt} = A_+(w_j)x(t) \sum_n \delta(t - t^n) - A_-(w_j)y(t) \sum_f \delta(t - t_j^f) \tag{2.27}$$

Hence, the synaptic efficacy is heightened when there is a postsynaptic firing by an amount that varies according to the value of the trace x left by the presynaptic spike. Then as well, the weight suffers a depression with a magnitude proportional to the the trace left by the trace y of the postsynaptic spike. If we integrate eq. 2.27 we obtain eq. 2.24. [18],[22],[25]

To illustrate how the weight w changes as a function of the local variables introduced, x_j and y, a simulation using the library Brian2 was done. Below is presented how the code by which x_j is updated as a function of y and y is modified as a function of x_j works.
For the first 10 ms, there was no spike produced and x_j and y are both set to zero (Fig. 2.16). At the moment $t = 10$ ms a presynaptic spike occurs and the variable x_j is updated by the rule: $x_j \leftarrow x_j + A_+$, where $A_+ = 0.01$ in our simulation [26]. When a presynaptic spike arrives the weight w is updated according to the following rule: $w \leftarrow w + y$, but as there was no postsynaptic firing y is still zero, so the weight w will remain unchanged. After the presynaptic spike, x_j starts to fall exponentially. At the time $t = 20$ ms the postsynaptic neuron fires and the variable y is changed: $y \leftarrow y + A_-$, with $A_- = -0.0105$. At the moment of a postsynaptic

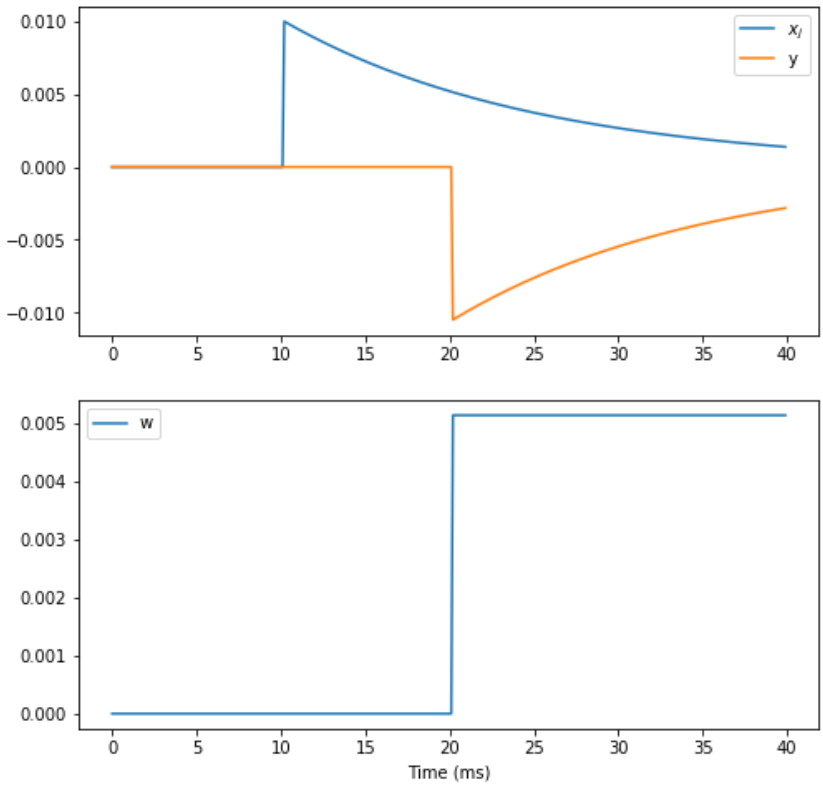

Figure 2.16: The Weight Change as a Function of Local Variables

spike, w is changed: $w \leftarrow w + x_j$. At $t = 20$ ms, x_j has become 0.005, as it can be seen in Fig. 2.16, so the weight is now $w = 0.005$, and it will remain so until another spike is produced [26].

Chapter 3
Design, Implementation and Experimental Results

3.1 Foreign Exchange Rate

3.1.1 Foreign Exchange Market

The foreign exchange market, also known as forex, FX or the currency market, is a global marketplace that regulates the exchange rate for currencies around the world. Those who participate in the FX market are allowed to buy, sell, exchange and speculate on currencies [27]. Foreign Exchange rate is one of the most significant channels through which we determine a country's relative level of economic health. A country's foreign exchange rate yields a window to its economic stability, so it is understandable the interest for it being persistently watched and analyzed. For instance, we want to know the foreign exchange rate when we have to send or receive money internationally.
Although, there are undoubtedly a myriad of factors that influence the shifts in exchange rates, the ones we will present are the following:

- *Inflation rates*: currency exchange rates are dependent on the market inflation. If a country has a lower inflation rate than another country, the first will experience an appreciation in the value of its currency. When the inflation is small the prices of goods and services rise at a slower rate.
- *Interest rates*: Fluctuations in interest rate disturb currency value and exchange rates. FX rates, interest rates and inflation are correlated. When a country has consistent increases in interest rate, its currency will appreciate due to the rates to lenders. As a consequence, it will attract more foreign capital, which will result in increased exchange rates.
- *Government Debt*: the public debt or national debt owned by the central government. If a country has a relatively high government debt, it will less likely acquire foreign capital, which will lead to inflation.
- *Political Stability and Performance*: A country which is less predisposed to political turbulence is more attractive to foreign investors, resulting in an appreciation in the value of its domestic currency [28].

From the aforementioned arguments we can conclude that the foreign exchange market is of great interest to the majority of the population, and at the same time a market influenced by many aspects which are difficult to predict.

3.1.2 Database

The database used in this simulation was obtained from the National Bank of Romania (BNR) website. It contains the exchange rates between Euro and RON (Romania's national currency) – EUR/RON, from 2010 until 2020. The samples were collected from the 4^{th} of January 2010 until the 3^{rd} of June 2020, on a daily basis, resulting in a total of 2629 samples.

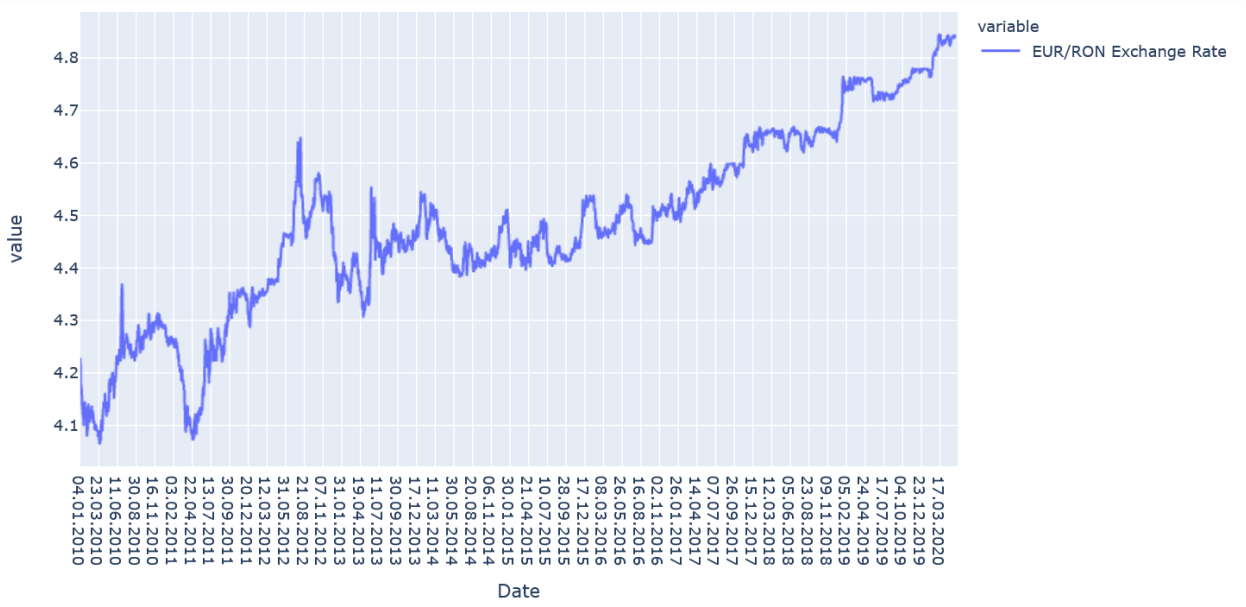

Figure 3.1: Foreign Exchange Rate EUR/RON, 2010-2020

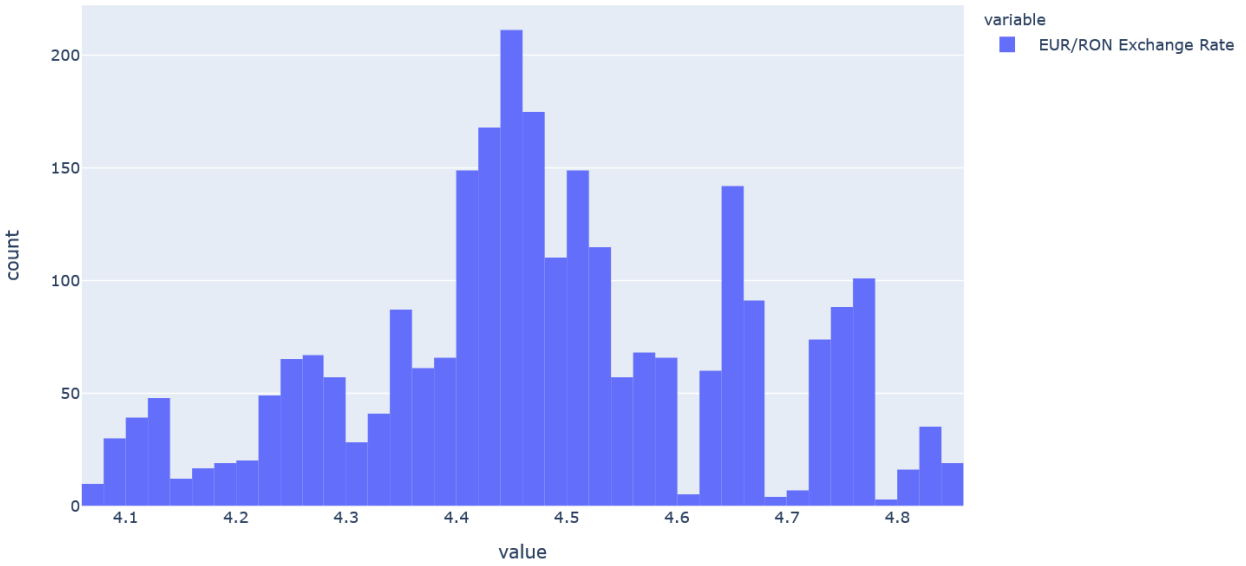

Figure 3.2: Histogram of the Foreign Exchange Rate EUR/RON, 2010-2020

Figure 3.1 shows the evolution of the previously mentioned variable over a 10-year period. For example, on the 6^{th} of August 2014, 1 Euro was exchanged for 4.4455 RON. Despite the fluctuations, there is clearly an upward trend in the exchange rate – RON sees a depreciation compared to EUR.

Statistic	*Value*
mean	4.479480
std	0.175376
min	4.065300
25%	4.383700
50%	4.466400
75%	4.598500
max	4.844800

Table 3.1: Statistical Parameters for the Foreign Exchange Database

The histogram of our dataset can be seen in fig. 3.2. The most frequent exchange rates are between 4.4 and 4.5. However, the distribution is not a perfect Gaussian, as we can find a low frequency for rates around 4.2 and 4.8, but also for rates around 4.3, 4.6 and 4.7.

Let us now proceed by looking at some statistics for the database.

Table 3.1 shows different statistical tools which give us better insight into the data. The *mean* has a value of 4.479480, meaning that the expected value of our data over all data points sits around this value. The *standard deviation* shows how dispersed from the mean are the values from our dataset, in this case the standard deviation being equal to 0.175376. A smaller value gives us a more uniform distribution, while a greater value would mean a more volatile distribution. The *minimum* value from our dataset is 4.065300, and the *maximum* value is 4.844800, which means that we cannot find any value out of this range. Another useful insight would be to know what percent of the data we can find below a certain value. From the Table 3.1 we can see that 25% (or one quarter) of the data lies below the value 4.383700, 50% (or half of the data) can be found below the value 4.466400 and 75% (or three quarters) of the data lies below the value 4.598500.

3.1.3 ARMA Based Forecasting

3.1.3.1 ACF and PACF of the Foreign Exchange Database

In order to apply a stochastic model, it is useful to know how the data is correlated. Figure 3.3 illustrates the autocorrelation of our dataset, for a number of $n = 600$ lags. We can see that for $n = 0$ we have an autocorrelation of 1 (as it should be, since a value is perfectly correlated with itself).
As n increases, meaning that the distance in time between two data point increases, the autocorrelation decreases, and eventually tends to zero. An autocorrelation of zero means that there is no relationship between two data points. This shows us that the data does not exhibits any seasonality trends, and the more distanced in time are any two data points, the less they are correlated. It can also be observed that for the first $n = 600$ lags, the data points are positively correlated.

The partial autocorrelation function (PACF) is slightly different from the autocorrelation function (ACF). The partial autocorrelation function seeks to remove the autocorrelations from previous lags. For this reason, in figure 3.4 we can see that only the first lag has the value 1, and then the rest are very close to zero. This is because there is little variation in the autocorrelation between two consecutive lags.

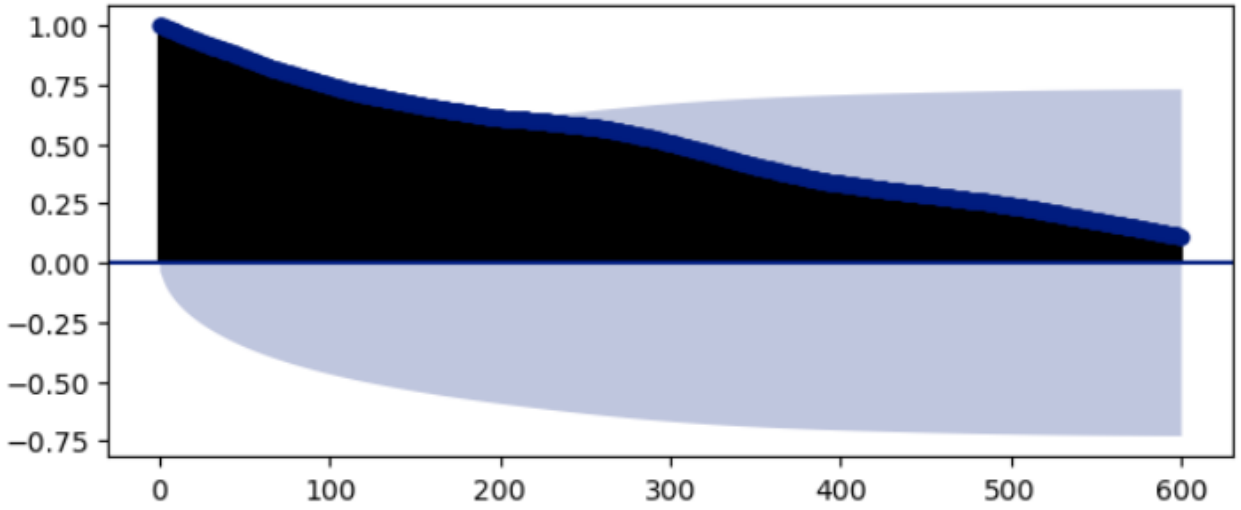

Figure 3.3: Autocorrelation of the Foreign Exchange Data

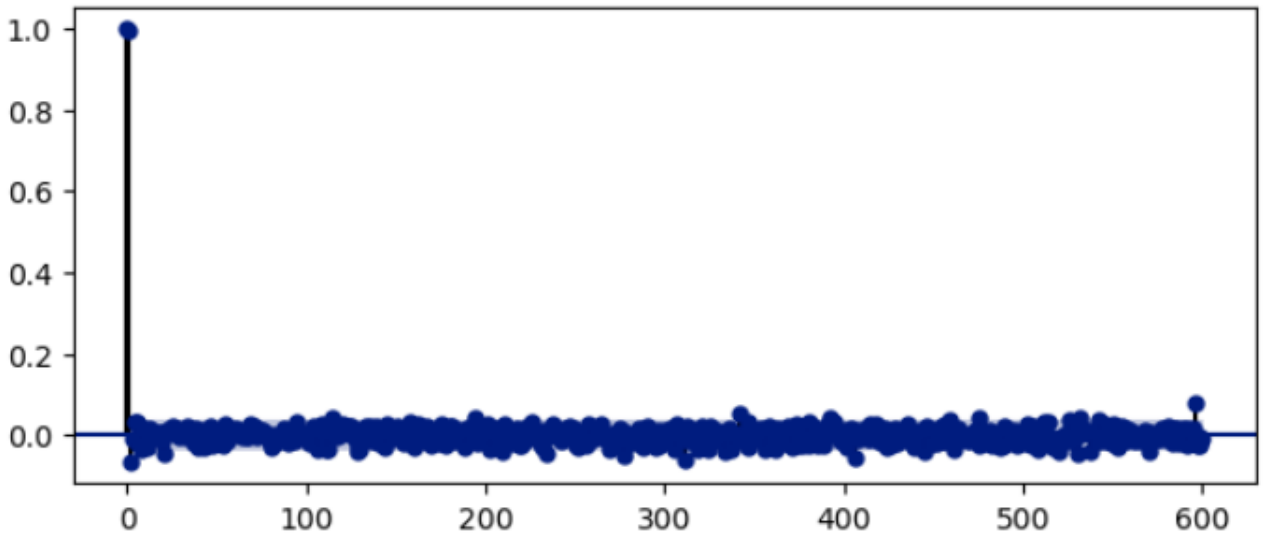

Figure 3.4: Partial Autocorrelation of the Foreign Exchange Data

3.1.3.2 Dickey–Fuller Test for Stationarity

$Dickey - FullerParameter$	$Value$
ADF Statistic	-1.208443
p-value	0.669953

Table 3.2: Dickey-Fuller Test Results

Before we can proceed forecasting with any ARMA-based model, we need to know if our data is stationary or not. If it is, then we can directly apply an ARMA model. However, if not, then we need to transform our data in order to make it stationary.
The Dickey-Fuller Test is one of the tools we have at our disposal to test the stationarity of the data set. What is of great interest are two parameters: the ADF Statistic and the p-value. If the ADF statistic is very negative, it means we have a stationary dataset. However, as it gets closer to zero, it is more likely that we have to deal with non-stationary data. A p-value below

a threshold (such as 5% (or 1%)) suggests that the data is stationary. On the other hand, a p-value above the threshold suggests our data is non-stationary.[11]

Table 3.2 gives us the results of the Dickey-Fuller test, when applied to the Foreign Exchange Rate database. The p-value exceeds the threshold of 0.05 (actually, the p-value is very large – 0.669953, so we can say with enough confidence the data is non-stationary). Also, the ADF Statistic is not very negative, so the data is non-stationary.

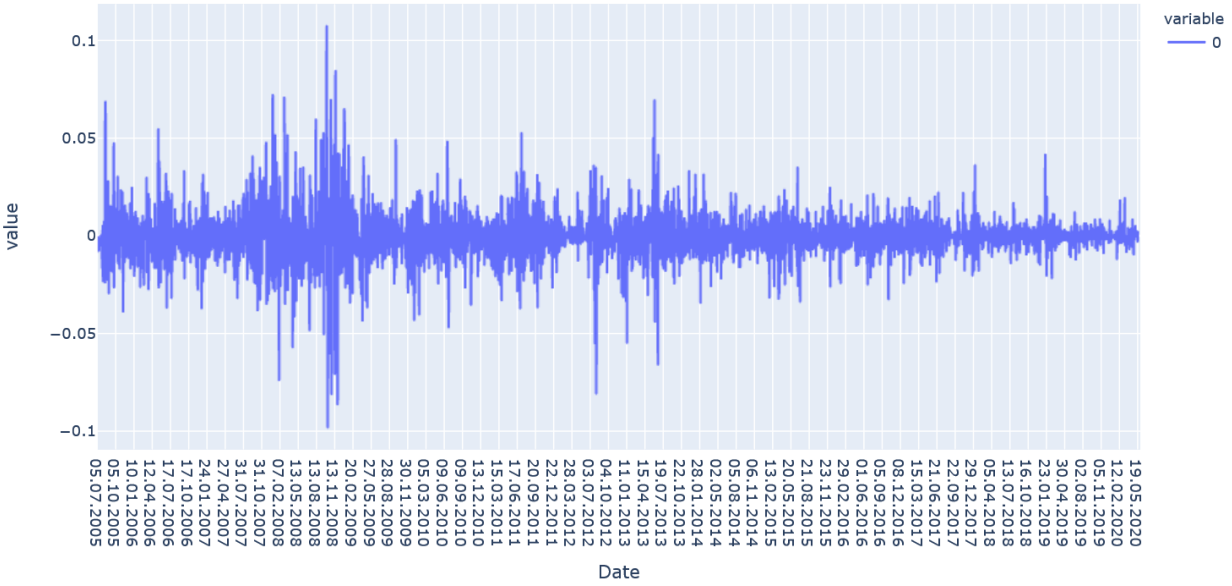

Figure 3.5: Residuals After First-Order Differencing

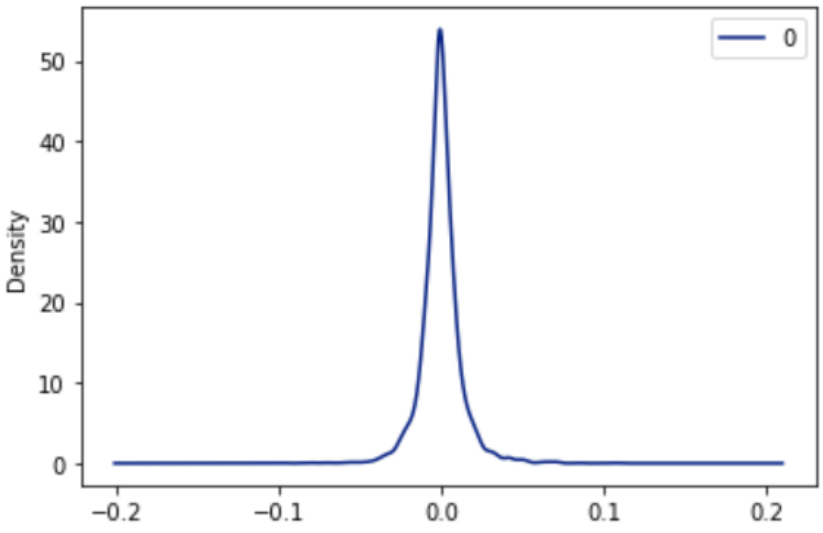

Figure 3.6: Distribution of the Residuals After First-Order Differencing

The next step is to transform the data in order to make it stationary. To do that, we will apply a first differencing to the data (I=1). The residuals from our first-order differencing can be seen in figure 3.5. It is clear that the data no longer poses any trends and it looks like a white noise, with the mean centered at zero.

However, to be really sure there is no need to further differentiate the data, the distribution of the residuals is shown in figure 3.6. There is no doubt that the mean of the residuals is zero, so the data resulted is stationary (there is no relationship between any two data points, no matter how close to each other they are in time).

3.1.3.3 ARMA Forecasting Results

After we have seen that there is no need for us to do a second-order differencing, we can now proceed by forecasting the data using an ARIMA model. Because we found that $I = 0$ (first-order differencing), we apply ARIMA (p,1,q), where p is the order of the AR model, and q is the order of the MA model.

We will compare the results from different ARMA-based model by using the mean squared error (MSE) metric (or, alternatively, the root mean squared error (RMSE) metric). As we would expect, we want these two metrics to be as close to zero as possible, so it will give us the most accurate results.

Model	*MSE*	*RMSE*
AR(1)	23.55×10^{-6}	48.528×10^{-4}
AR(2)	23.58×10^{-6}	48.563×10^{-4}
AR(3)	23.68×10^{-6}	48.671×10^{-4}
AR(4)	23.71×10^{-6}	48.696×10^{-4}
AR(5)	23.71×10^{-6}	48.697×10^{-4}

Table 3.3: Autoregressive Model Forecasting Results

Table 3.3 shows the results from our simulation for different values of the parameter p, while the parameter q is set to zero, so we get an autorregresive AR(p) model.

Model	*MSE*	*RMSE*
MA(1)	23.54×10^{-6}	48.526×10^{-4}
MA(2)	23.57×10^{-6}	48.552×10^{-4}
MA(3)	23.68×10^{-6}	48.661×10^{-4}
MA(4)	23.75×10^{-6}	48.734×10^{-4}
MA(5)	23.74×10^{-6}	48.727×10^{-4}

Table 3.4: Moving Average Model Forecasting Results

Table 3.4 shows the results from our simulation for different values of the parameter q, with the parameter p set to zero, so we get a moving average MA(q) model.

Table 3.5 shows the results from our simulation for different values of the parameter p ($p = 1, 2, 3, 4$), and different values for the parameter q ($q = 1, 2, 3, 4$).

From the above results we can conclude that the most accurate predictions are obtained for lower values of the parameters p and q. Also, simulating greater MA and AR models with greater values for p and q would be computationally expensive and thus inefficient. Fortunately, there is no need for us to do so, as the best results require smaller values of the aforementioned parameters.

Model	*MSE*	*RMSE*
ARMA(1,1)	23.67×10^{-6}	48.6×10^{-4}
ARMA(1,2)	23.58×10^{-6}	48.532×10^{-4}
ARMA(1,3)	23.72×10^{-6}	48.756×10^{-4}
ARMA(1,4)	23.82×10^{-6}	48.854×10^{-4}
ARMA(2,1)	23.71×10^{-6}	48.697×10^{-4}
ARMA(2,2)	24.06×10^{-6}	49.055×10^{-4}
ARMA(2,3)	24.11×10^{-6}	49.262×10^{-4}
ARMA(2,4)	24.30×10^{-6}	49.421×10^{-4}
ARMA(3,1)	24.79×10^{-6}	49.897×10^{-4}
ARMA(3,2)	25.02×10^{-6}	50.076×10^{-4}
ARMA(3,3)	25.36×10^{-6}	50.358×10^{-4}
ARMA(3,4)	25.89×10^{-6}	50.882×10^{-4}
ARMA(4,1)	26.25×10^{-6}	51.234×10^{-4}
ARMA(4,2)	26.64×10^{-6}	51.613×10^{-4}
ARMA(4,3)	26.98×10^{-6}	51.942×10^{-4}
ARMA(4,4)	27.23×10^{-6}	52.182×10^{-4}

Table 3.5: Autoregressive Moving Average Model Forecasting Results

The lowest MSE value was obtained for the MA(1) model (MSE=0.000023548), but overall there was little difference in the performance of the ARMA-based model forecasts. To get an idea about how our network performed, table 3.6 shows the actual vs. predicted values for the first 10 forecasted values, using the MA(1) model.

Date	*Actual*	*Predicted*
27.04.2018	4.649000	4.647277
30.04.2018	4.661800	4.649338
02.05.2018	4.658900	4.663002
03.05.2018	4.663000	4.658778
04.05.2018	4.661800	4.663543
07.05.2018	4.656900	4.661867
08.05.2018	4.654200	4.656708
09.05.2018	4.649600	4.654202
10.05.2018	4.645400	4.649433
11.05.2018	4.639700	4.645276

Table 3.6: Actual vs. Predicted Values for MA(1)

Now that we have seen how the stochastic models perform on the Foreign Exchange database, let us proceed to the results obtained by using artificial neural networks. The next section will analyze two different network architectures, and for each there will be given the two metrics (MSE and RMSE).

3.1.4 Neural Networks Forecasting

In this section, we can find the results obtained by simulating artificial neural networks using the Python Keras library.

3.1.4.1 Proposed Architecture – 1

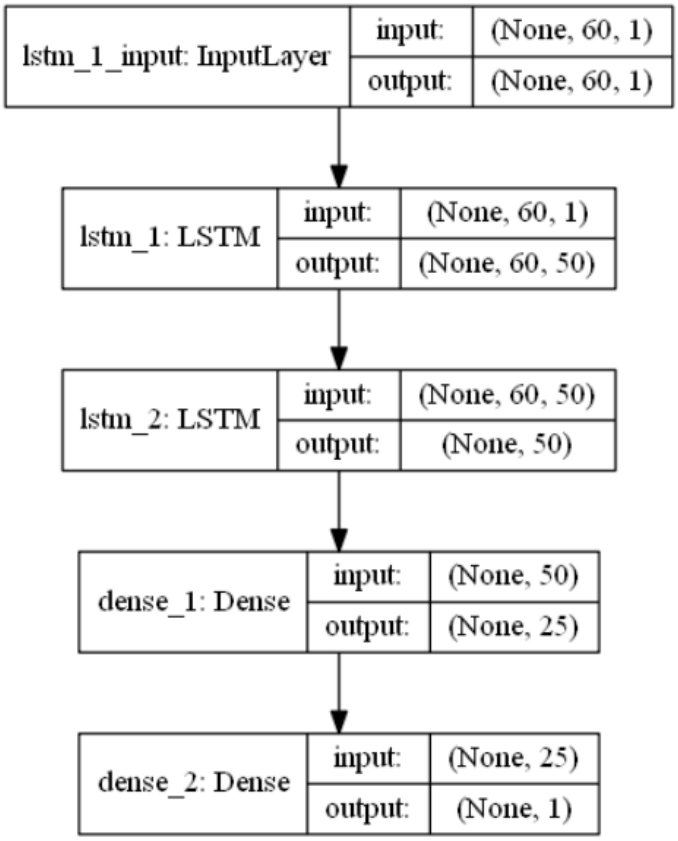

Figure 3.7: First Neural Network Architecture

The first neuronal architecture used to forecast the foreign exchange data is shown in fig. 3.7. This neural architecture was proposed on the Kaggle website to forecast financial time series.[29] The architecture uses a sequential model, and contains four layers plus the input layer. The input layer has the shape of the training data. The *first* and the *second* layers have 50 long short-term memory (LSTM) neurons each. The *third* layer is a *dense* layer, that is, a regular layer of neurons in a neural network, where each neuron receives input from all the neurons in the previous layer (thus densely connected). This third layer has 25 neurons, so its output will be 25, as it can be seen in fig. 3.7. The output from the third layer will be the input for the *fourth* layer, which is also a dense layer, but with only one neuron (output=1, fig. 3.7).
In table 3.7 we can find the MSE and RMSE scores for a batch size (the number of training samples to work through before the model's internal parameters are updated) of 1, but different values for the epochs (complete passes through the training dataset).

Number of Epochs	*MSE*	*RMSE*
1	26.35×10^{-5}	16.232×10^{-3}
2	37.39×10^{-6}	58.175×10^{-4}
3	30.27×10^{-5}	17.4×10^{-3}
4	33.84×10^{-6}	61.635×10^{-4}
5	63.59×10^{-6}	79.746×10^{-4}
10	37.39×10^{-5}	19.33×10^{-3}
25	85.43×10^{-6}	92.431×10^{-4}

Table 3.7: First Architecture Results, Batch Size = 1

In table 3.8 we can see the results obtained with the first architecture, by varying the batch size, keeping the number of epochs constant at the value of 25. It is clear that the best performance of this architecture was obtained for a batch size of 20 and the number of epochs of 25.

Batch Size	*MSE*	*RMSE*
10	43.76×10^{-6}	66.15×10^{-4}
20	28×10^{-6}	52.918×10^{-4}
30	10.08×10^{-5}	10.042×10^{-3}
40	19.39×10^{-5}	13.927×10^{-4}
50	30.06×10^{-6}	54.833×10^{-4}
100	80.94×10^{-6}	89.97×10^{-4}

Table 3.8: First Architecture Results, Number of Epochs=25

Figures 3.8, 3.9, 3.10, 3.11, 3.12 show the results obtained for different values of the number of epochs and the batch size. With the blue colour are represented the *training* data, in our case, the first 80% of the values. With green are represented the data used for *validation*, that is, the last 20% of the values from our dataset. The values are used to compare our *predicted* values, which are illustrated with the colour red.

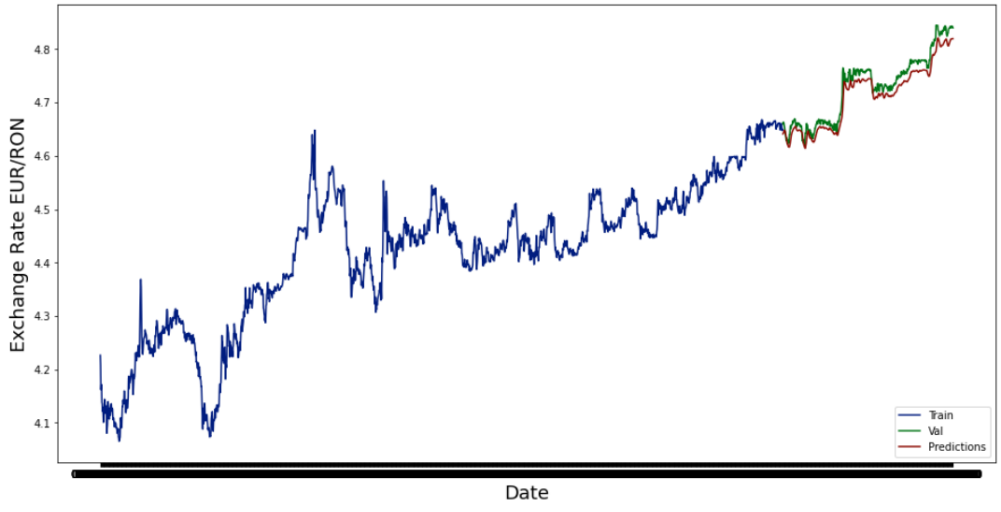

Figure 3.8: First Neural Network Results - Batch Size=1, Epochs=1

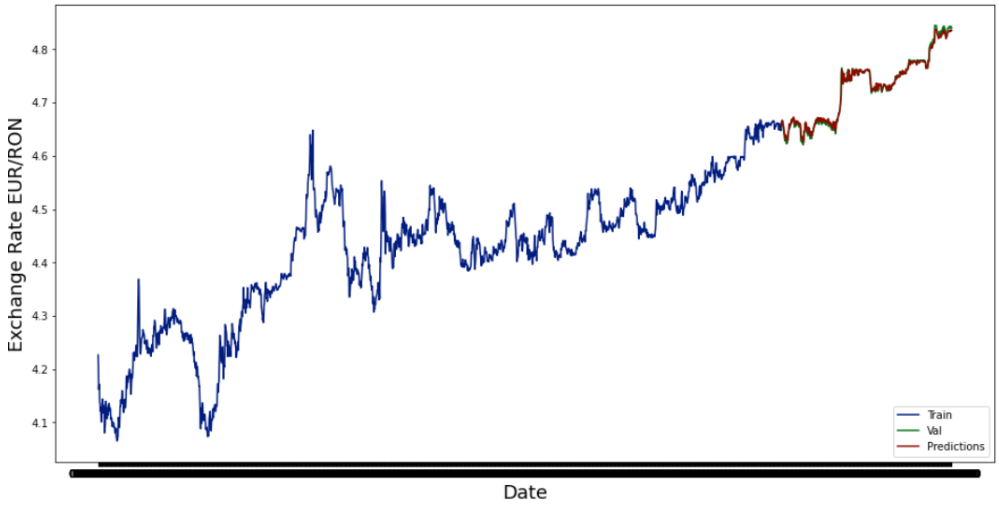

Figure 3.9: First Neural Network Results - Batch Size=1, Epochs=2

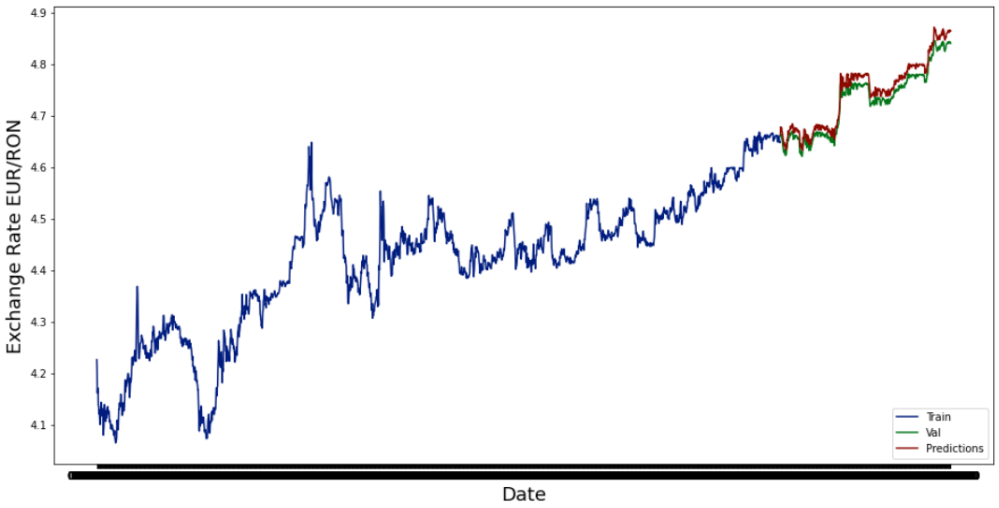

Figure 3.10: First Neural Network Results - Batch Size=1, Epochs=3

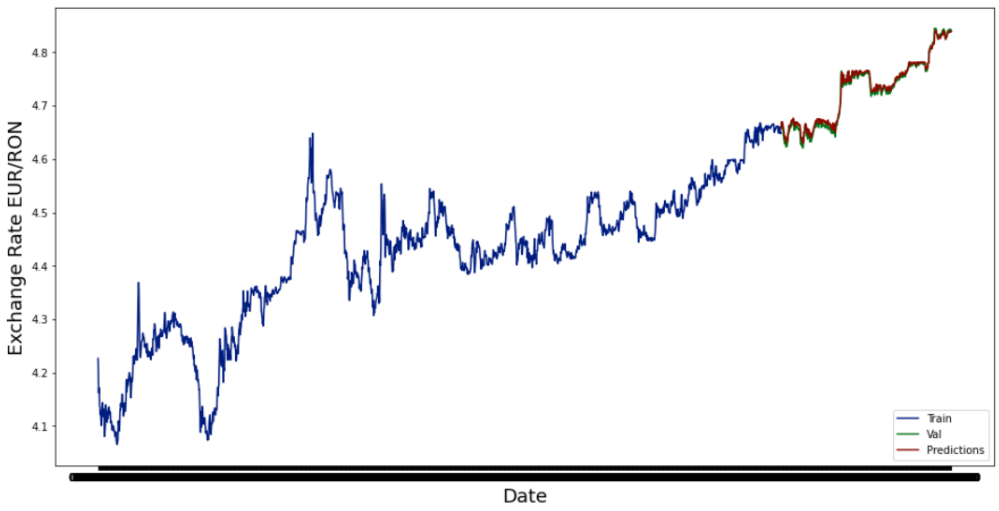

Figure 3.11: First Neural Network Results - Batch Size=1, Epochs=25

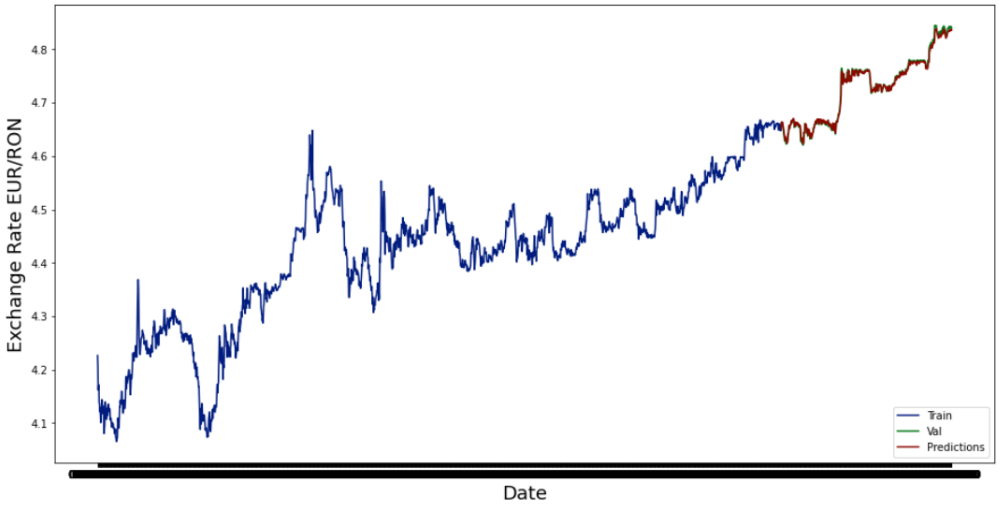

Figure 3.12: First Neural Network Results - Batch Size=50, Epochs=25

Similar to our assumptions made just by looking at the MSE, the predictions obtained using a batch size of 1 and two epochs are better than the predictions made by using a batch size of 1 and only one epoch (in the first case the green and the red line are almost one and the same; in the second case they are not quite overlapping).

In fig. 3.12 we can see the best performing model from the first architecture (batch size=50, epochs=25). It is obvious the two lines are almost the same, so the performance of this model is quite remarkable.

3.1.4.2 Proposed Architecture – 2

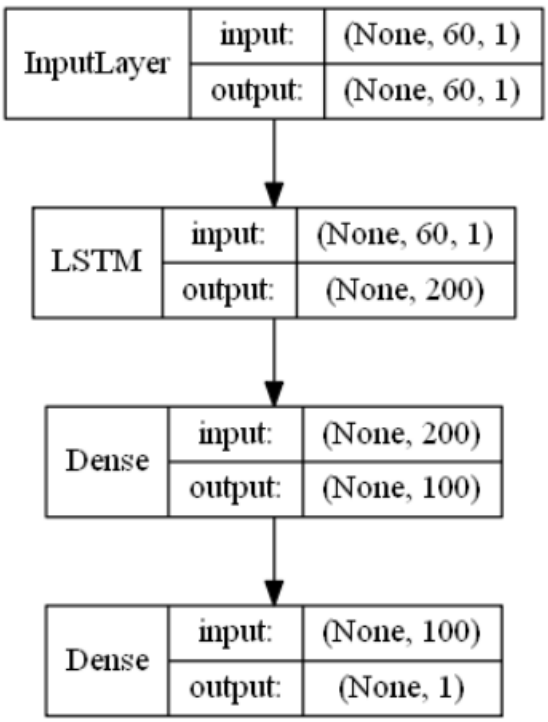

Figure 3.13: Second Neural Network Architecture

The second neural network architecture (fig. 3.13) is a neural network developed by me for the scope of this thesis. Although it has a similar sequential model,the number of layers and

neurons differ. The input layer has, as in the previous case, the shape of the training data. The *first* layer has 200 LSTM neuron cells. The *second* and *third* layers are dense and have 100 neurons and one neuron, respectively.

Number of Epochs	*MSE*	*RMSE*
1	27.23×10^{-6}	52.182×10^{-4}
2	26.14×10^{-6}	51.12×10^{-4}
3	39.76×10^{-6}	63.055×10^{-4}
4	35.97×10^{-6}	59.974×10^{-4}
5	31.65×10^{-6}	52.258×10^{-4}
10	25.28×10^{-6}	50.52×10^{-4}
25	36.73×10^{-6}	60.605×10^{-4}

Table 3.9: Second Architecture Results, Batch Size = 1

In table 3.9 we can find the MSE and RMSE scores for a batch size of 1, but different values for the epochs, for the second neural network architecture.

Batch Size	*MSE*	*RMSE*
10	25.89×10^{-6}	50.882×10^{-4}
20	37.23×10^{-6}	61.016×10^{-4}
30	32.52×10^{-6}	57.026×10^{-4}
40	24×10^{-6}	48.989×10^{-4}
50	35.62×10^{-6}	59.682×10^{-4}
100	30.83×10^{-6}	55.524×10^{-4}

Table 3.10: Second Architecture Results, Number of Epochs=25

In table 3.10 we can see the results obtained with the second architecture, by varying the batch size, keeping the number of epochs constant at the value of 25. It is clear that the best performance of this architecture was obtained for a batch size of 40 and the number of epochs of 25, with a MSE score of 24×10^{-6}.

From tables 3.7, 3.8, 3.9 and 3.10, it is clear that the second neural network architecture outperformed the first one, giving more accurate predictions. Hence, the MSE and RMSE scores are lower, especially for a lower number of epochs (therefore more time effective).

Figures 3.14 3.15, 3.16, 3.17, 3.18 show the results obtained for different values of the number of epochs and the batch size. The training data is represented in blue (first 80% of the data), while the data used for validation is coloured in green (last 20% of the values) . The values are compared to our predicted values, which are illustrated with the colour red. This architecture gives a very accurate prediction for a batch size of 40 and 25 epochs, as it can be seen in fig. 3.18, where the green and red line are very hard to distinguish.

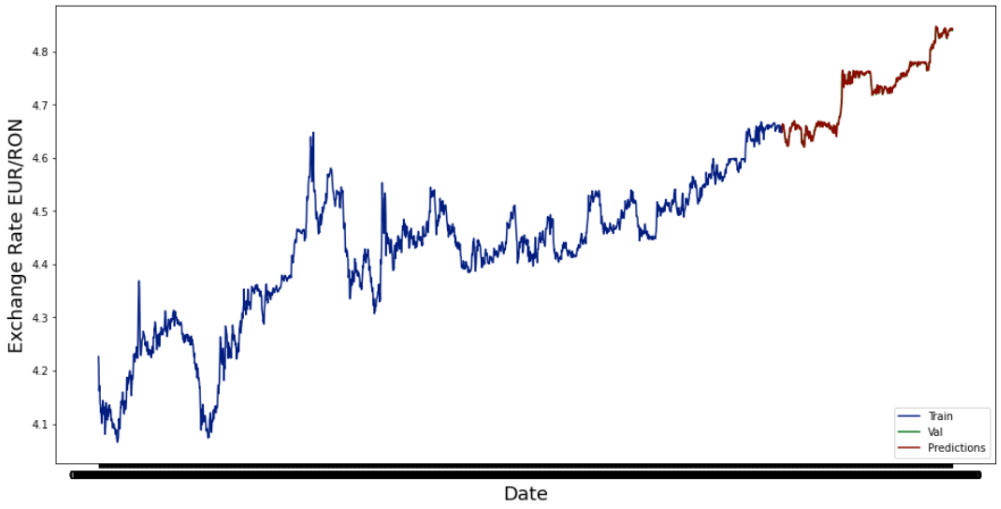

Figure 3.14: Second Neural Network Results - Batch Size=1, Epochs=1

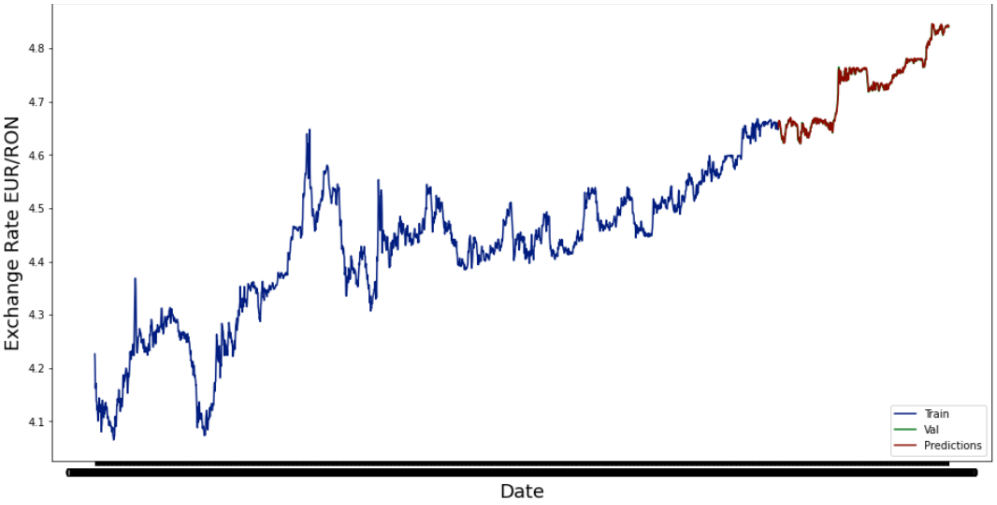

Figure 3.15: Second Neural Network Results - Batch Size=1, Epochs=2

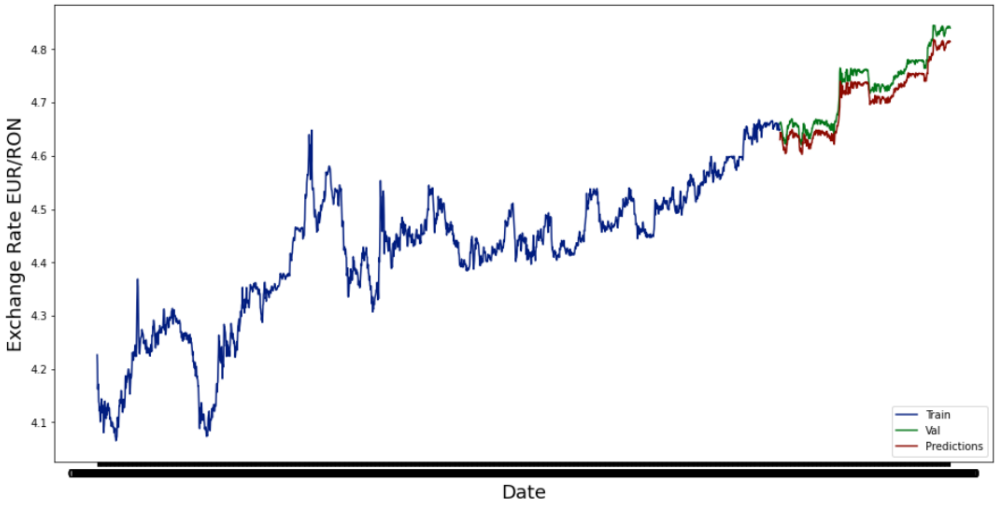

Figure 3.16: Second Neural Network Results - Batch Size=1, Epochs=3

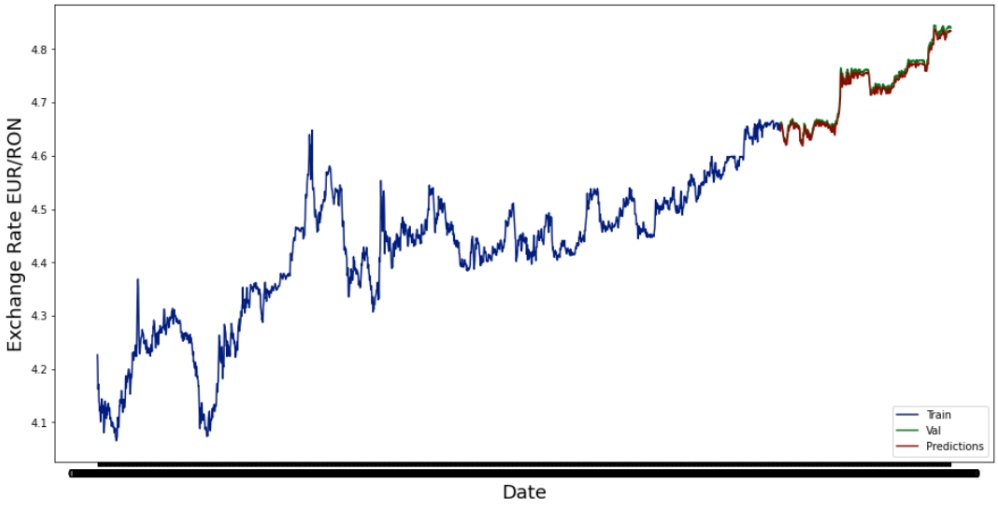

Figure 3.17: Second Neural Network Results - Batch Size=1, Epochs=5

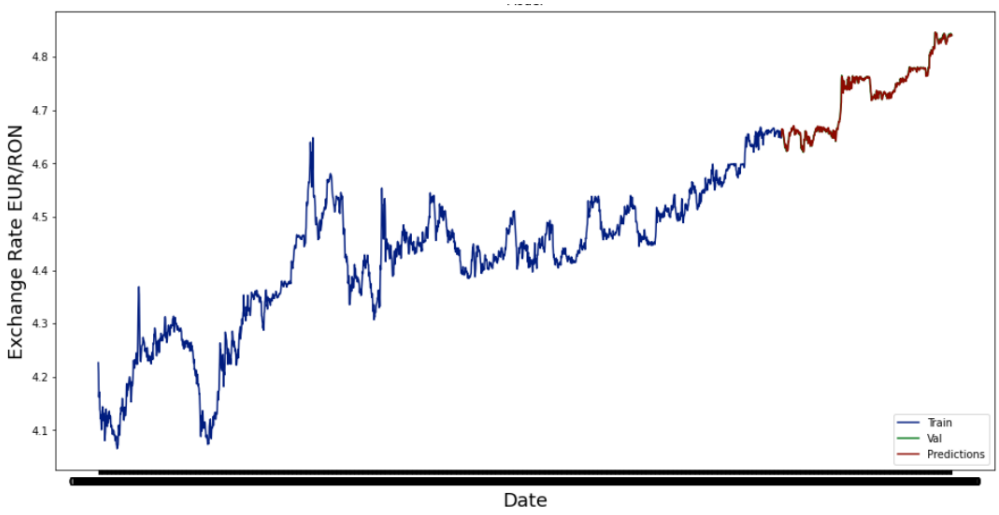

Figure 3.18: Second Neural Network Results - Batch Size=40, Epochs=25

3.2 Crude Oil Stock Price

3.2.1 Market Overview

Crude oil, sometimes referred to as "black gold", is one of the world's most precious commodities. Its changes in price can affect the economic ecosystem at every level, from family budgets to corporate earnings or to the nations' GDP. Indeed, abrupt price drops or unexpected spikes can send global financial markets into animosity. Crude oil prices change rapidly in response to news cycles, policy changes, and fluctuations in the world's other markets. Since 2014, oil prices have experienced a downward trend, falling from highs of around $105 per barrel. In February and March of 2020, crude prices accelerated their decline in reaction to the coronavirus pandemic and an expected extreme drop in demand. In addition, major oil producers failed to come to an agreement on production cuts, intensifying the problem.

The major factors which drive the crude oil price are enumerated below:

- ***Supply***
  For several decades, the Organization of Petroleum Exporting Countries (OPEC) has

been a monumental player on the world's trading floors, with its oil-producing member nations working together to determine prices by raising or reducing crude oil production. While OPEC's grip on the market has loosened in the past years, its decisions continue to have a dominant role. Every action made by OPEC is watched closely by governments, oil companies, speculators, hedgers, investors, traders, policymakers, and consumers. OPEC's policies are affected, in turn, by geopolitical developments. The crude oil supply is also affected by external factors, which might include exploration and production (E&P) costs, weather patterns, investments and innovations.

- ***Demand***
  Intense economic growth and industrial production tend to heighten the demand for oil — as reflected by the changes in demand patterns by the non-OECD (Organization for Economic Cooperation and Development) nations, which have grown quickly in recent years. Other critical factors that influence the demand for oil include transportation (both commercial and personal), population growth, and seasonal changes. For instance, oil use increases during the busy summer travel seasons and in the winters, when more heating fuel is consumed.

- ***Derivatives and Reports*** There is an increasing trend of market participants buying and selling crude oil, not in its physical form, but in the form of contracts. For instance, airlines and oil producers use derivatives, like futures and options, to hedge against fluctuations in the oil price, while speculators drive those prices upwards or downwards when there are waves of buying or selling over incoming news.[30]

It is clear that predicting oil prices is much more difficult than predicting other stock prices, due to the great forces which shape the oil market. Thus, it is a huge challenge to forecast or even speculate how the oil market is going to evolve.

3.2.2 Database

The database used in this simulation was collected from the Yahoo Finance website. It contains the crude oil prices for a period of two years: from the 18^{th} of June 2018 until the 17^{th} of June 2020. The samples were collected on a daily basis, resulting in a total of 609 samples.

Figure 3.19 shows the evolution of the previously mentioned variable over a period of two years. In the first image (fig. 3.19 first row, left) it is presented the evolution of the *open* price, that is, the price the crude oil stock was traded at the beginning of the trading day. The first image (fig. 3.19 first row, right) shows the evolution of the *closing* price, that is, the price the crude oil stock was traded at the end of the trading day. The second row presents the evolution of the *high* and *low* prices, which are the maximum and minimum prices of the oil stock on a given day. The fifth image from fig. 3.19 (third row, left) shows the *volume* traded on a specific day, that is, the number of crude oil stocks traded on a given day. The *adjusted close* price is the closing price adjusted for splits and dividend distributions.
It is clear that the oil price presented some fluctuations, but it had a dramatic decline around February 2020 (most likely because of the coronavirus crisis). It is also then when the volume of stocks being traded daily skyrocketed from around 500.000 to approximately 500 millions.
Because the price of a stock is usually referred to as its adjusted closing price, we will further analyze this variable in greater detail.
Fig. 3.20 shows the evolution of the adjusted closing price of the crude oil over a two years period. What is worth noting is that on the 20^{th} of April 2020 the price fell even under zero,

an unprecedented event for the oil market. This proves once again the difficulty of predicting such noisy datasets as financial time series.

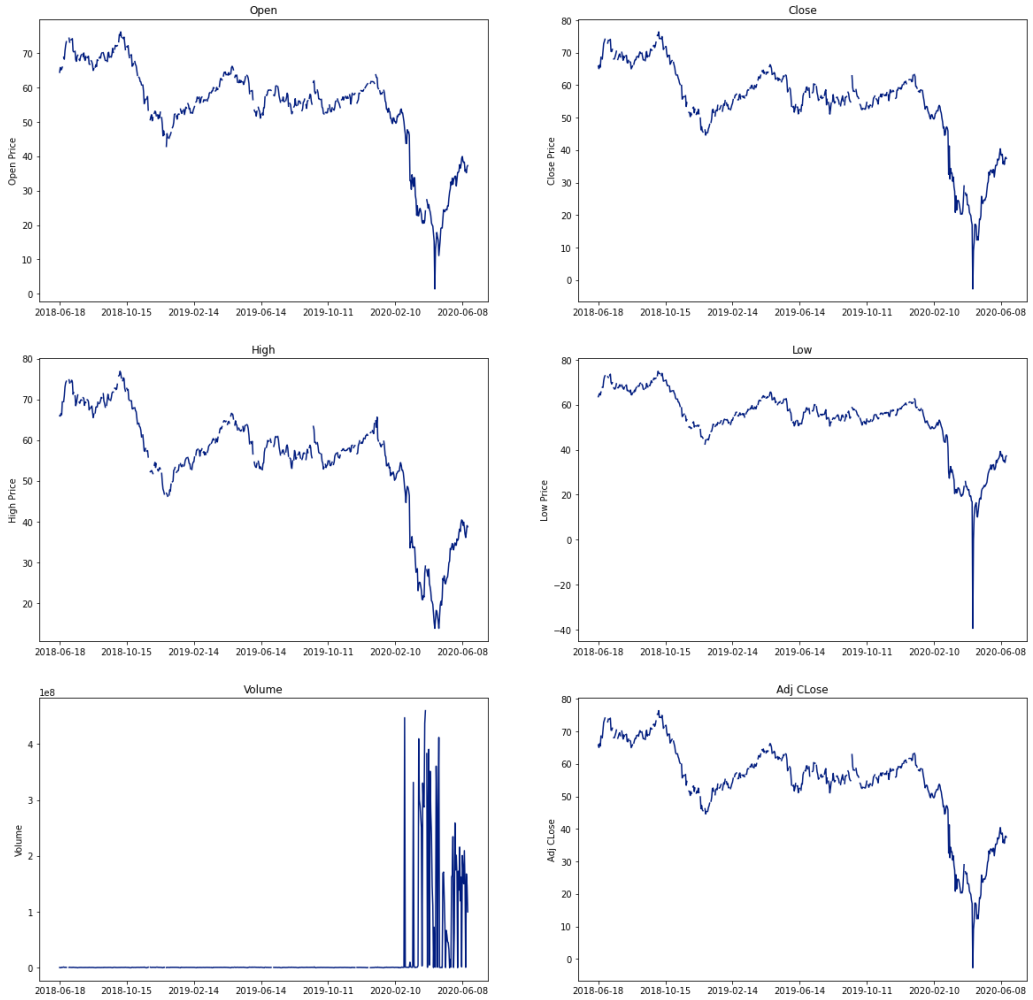

Figure 3.19: Crude Oil Price - All Columns

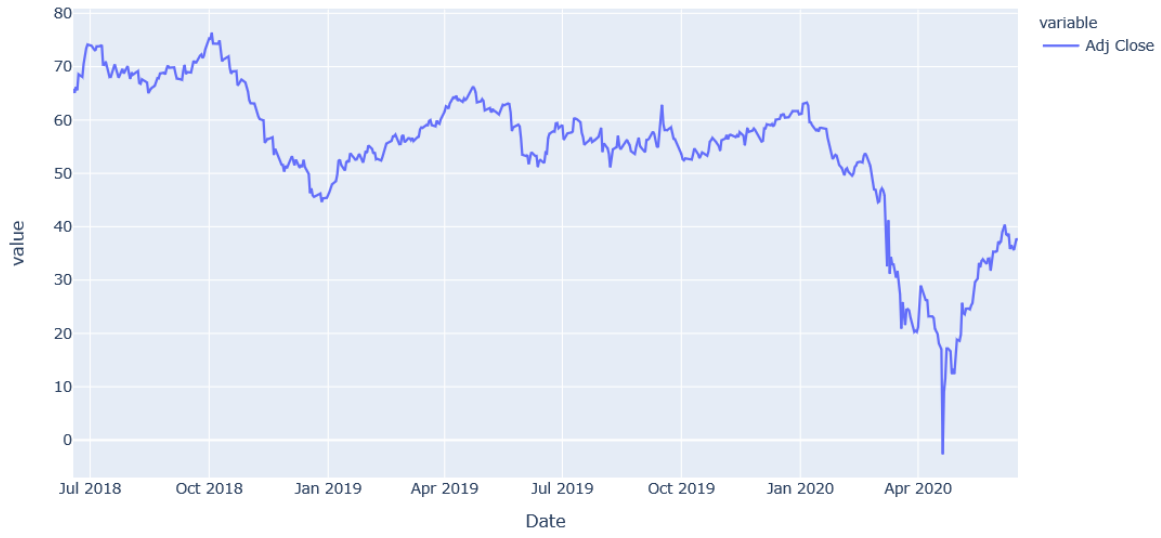

Figure 3.20: Crude Oil - Adjusted Close Price

Fig. 3.21 represents the histogram of the adjusted closing price. As expected, most values of the dataset are between \$50 and \$70, with significantly less values outside this range.

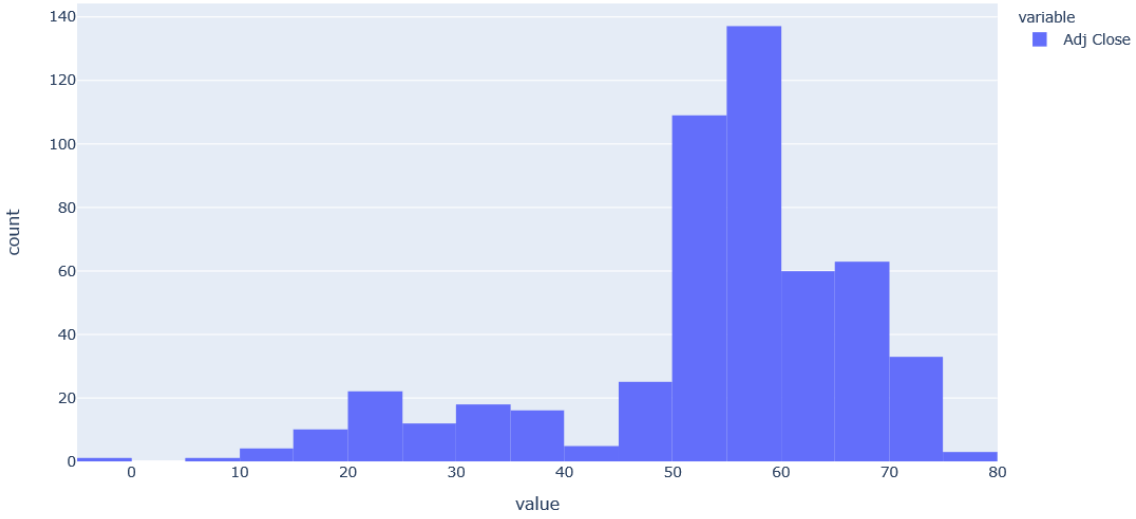

Figure 3.21: Histogram - Crude Oil, Adjusted Close Price

Let us now proceed by looking at some statistics of the adjusted close price.

Table 3.11 shows different statistical tools which give us better insight into the data. The *mean* has a value of 53.696994, meaning that the expected value of our data over all data points sits around this value. The *standard deviation* shows how dispersed from the mean are the values from our dataset, in this case the standard deviation is equal to 13.947700. A smaller value gives us a more uniform distribution, while a greater value would mean a more volatile distribution. The *minimum* value from our dataset is -2.72, and the *maximum* value is 76.410004, which means that we cannot find any value out of this range. Another useful insight would be to know what percent of the data we can be found below a certain value. From the Table 3.1 we can see that 25% (or one quarter) of the data lies below the value 51.44, 50% (or half of the data) can be found below the value 56.25 and 75% (or three quarters) of the data lies below the value 62.059999.

Statistic	*Value*
mean	53.696994
std	13.947700
min	-2.720000
25%	51.440000
50%	56.250000
75%	62.059999
max	76.410004

Table 3.11: Statistical Parameters for the Crude Oil Adjusted Close Price

3.2.3 ARMA Based Forecasting

3.2.3.1 ACF and PACF of the Crude Oil Adjusted Close Price

As mentioned in the previous section, in order to apply a stochastic model, it is useful to know how the data is correlated. Figure 3.22 illustrates the autocorrelation of our dataset, for a number $n = 500$ lags. We can see that for $n = 0$ we have an autocorrelation of 1 (as it should be, since a value is perfectly correlated with itself).

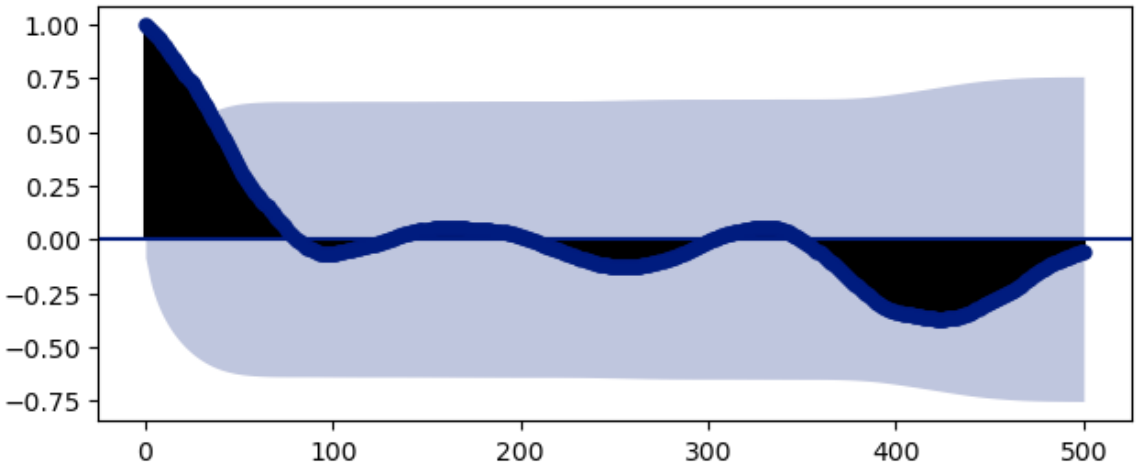

Figure 3.22: Autocorrelation of the Crude Oil Adjusted Close Price

As n increases, meaning that the distance in time between two data point increases, the autocorrelation decreases, and eventually tends to zero after nearly 100 lags. An autocorrelation of zero means that there is no relationship between two data points. What is also important is that at about $n = 400$ lags the data is very negatively correlated, meaning that one value increases, the other (after 400 moments) decreases.

The partial autocorrelation function (PACF) is slightly different from the autocorrelation function (ACF). The partial autocorrelation function seeks to remove the autocorrelations from previous lags. For this reason, in figure 3.23 we can see only a few spikes in the plot, and they happen when the correlation between two consecutive lags tends to be very different.

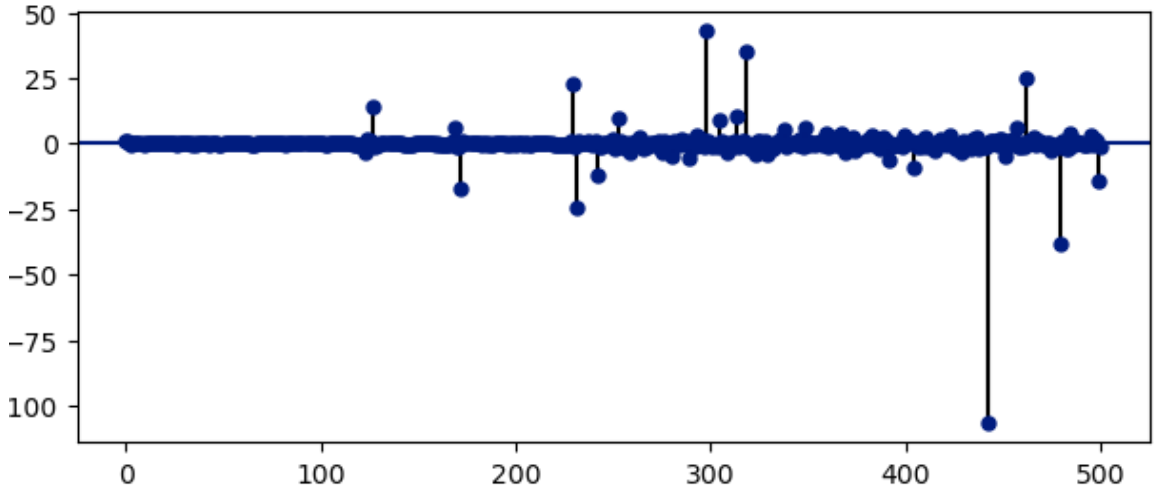

Figure 3.23: Partial Autocorrelation of the Crude Oil Adjusted Close Price

3.2.3.2 Dickey-Fuller Test for Stationarity

Before we can proceed forecasting with any ARMA-based model, we need to know if our data is stationary or not. If it is, then we can directly apply an ARMA model. However, if it is not, we need to transform our data in order to make it stationary.

We use once again the Dickey-Fuller Test to test the stationarity of the data set. What is of great interest are two parameters: the ADF Statistic and the p-value. If the ADF statistic is very negative, it means we have a stationary dataset. However, as it gets closer to zero, it is more likely that we have to deal with non-stationary data. A p-value below a threshold (such as 5% (or 1%)) suggests that the data is stationary. On the other hand, a p-value above the threshold suggests our data is non-stationary.[11]

$Dickey-FullerParameter$	$Value$
ADF Statistic	-1.651211
p-value	0.456409

Table 3.12: Dickey-Fuller Test Results

Table 3.12 gives us the results of the Dickey-Fuller test, when applied to the adjusted price of the crude oil database. The p-value exceeds the threshold of 0.05 (actually, the p-value is large enough – 0.456409, so we can say with enough confidence the data is non-stationary). Also, the ADF Statistic is not very negative, so the data is non-stationary.

The next step is to transform the data in order to make it stationary. To do that, we will apply a first differencing to the data (I=1). The residuals from our first-order differencing can be seen in figure 3.24. It is clear that the data no longer poses any trends and it looks like a white noise, with the mean centred in zero.

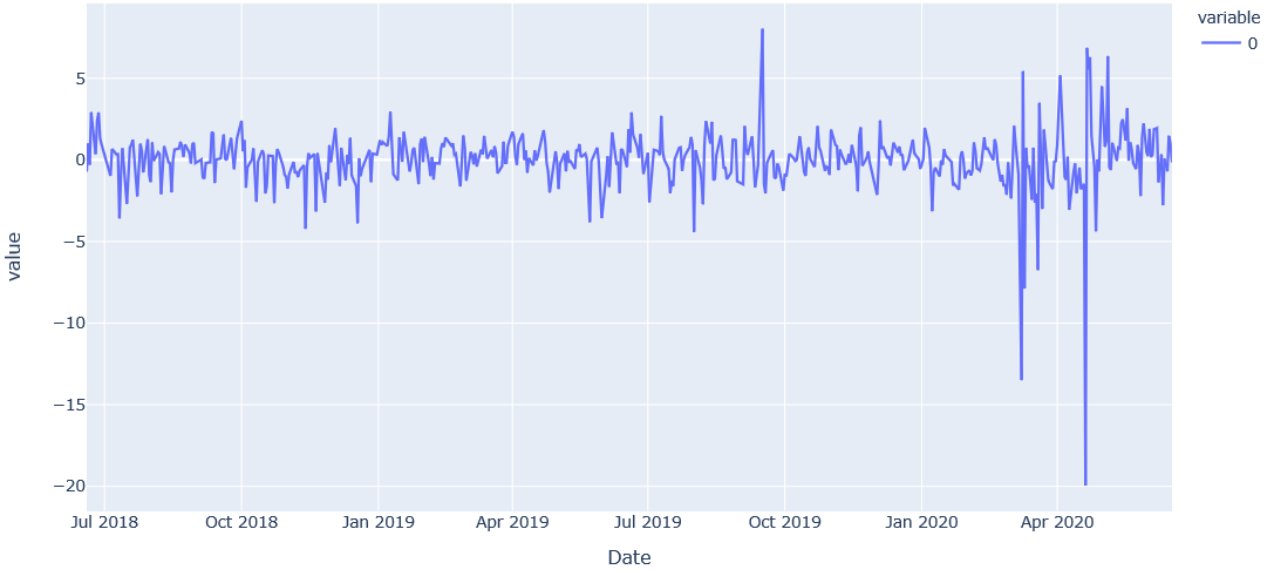

Figure 3.24: Residuals After First-Order Differencing

However, to be really sure there is no need to further differentiate the data, the distribution of the residuals is shown in figure 3.25. There is no doubt that the mean of the residuals is

zero, so the data resulted is stationary (there is no relationship between any two data points, no matter how close to each other they are in time).

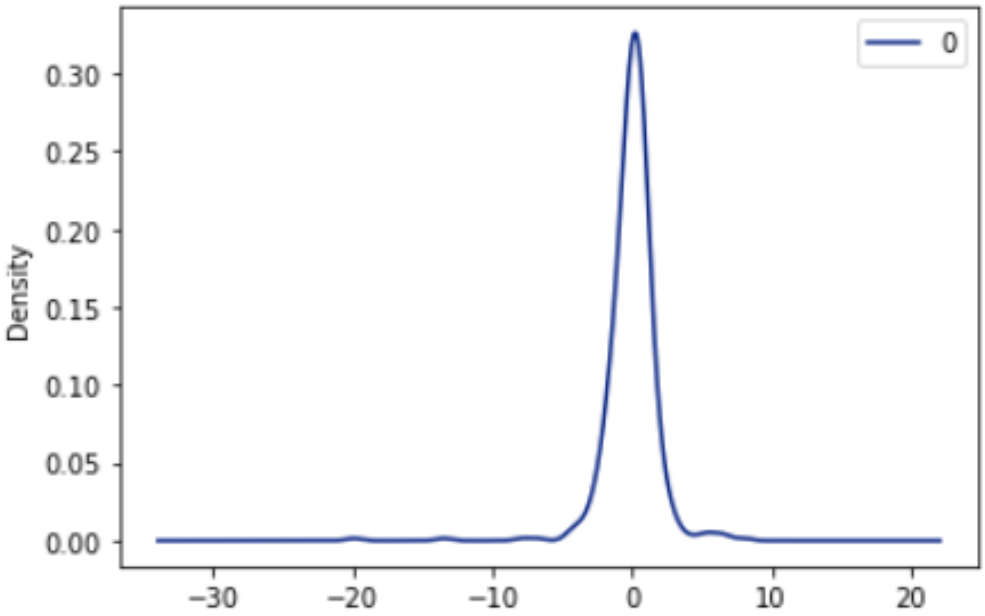

Figure 3.25: Distribution of the Residuals After First-Order Differencing

3.2.3.3 ARMA Forecasting Results

After we have seen that there is no need for us to do a second-order differencing, we can now proceed by forecasting the data using an ARIMA model. Because we found that $I = 0$ (first-order differencing), we apply ARIMA (p,1,q), where p is the order of the AR model, and q is the order of the MA model.
We will compare the results from different ARMA-based models by using the mean squared error (MSE) metric (or, alternatively, the root mean squared error (RMSE) metric). Needless to say, we want these two metrics to be as close to zero as possible, so it will give us the most accurate results.

Table 3.13 shows the results from our simulation for different values of the parameter p, while the parameter q is set to zero, so we get an autorregresive AR(p) model.

Model	*MSE*	*RMSE*
AR(1)	11.5976	3.4055
AR(2)	11.8932	3.4486
AR(3)	11.9299	3.4539
AR(4)	11.9706	3.4598
AR(5)	12.1210	3.4815
AR(10)	12.0932	3.4775

Table 3.13: Autoregressive Model Forecasting Results

Table 3.14 shows the results from our simulation for different values of the parameter q, with the parameter p set to zero, so we get a moving average MA(q) model.

Model	*MSE*	*RMSE*
MA(1)	11.7236	3.4240
MA(2)	11.9284	3.4537
MA(3)	11.8675	3.4449
MA(4)	12.0188	3.4668
MA(5)	12.1008	3.4786
MA(10)	12.2522	3.5003

Table 3.14: Moving Average Model Forecasting Results

Table 3.15 shows the results from our simulation for different values of the parameter p ($p = 1, 2, 3, 4$), and different values for the parameter q ($q = 1, 2, 3, 4$).

Model	*MSE*	*RMSE*
ARMA(1,1)	11.8385	3.4407
ARMA(1,2)	12.0934	3.4775
ARMA(1,3)	12.5421	3.5414
ARMA(1,4)	13.0076	3.6066
ARMA(2,1)	11.8251	3.4388
ARMA(2,2)	12.0521	3.4716
ARMA(2,3)	12.4786	3.5325
ARMA(2,4)	12.7659	3.5729
ARMA(3,1)	12.1244	3.4820
ARMA(3,2)	12.1693	3.4884
ARMA(3,3)	12.2342	3.4977
ARMA(3,4)	12.2854	3.5050
ARMA(4,1)	12.0845	3.4763
ARMA(4,2)	12.1457	3.4851
ARMA(4,3)	12.5827	3.5472
ARMA(4,4)	13.0769	3.6161

Table 3.15: Autoregressive Moving Average Model Forecasting Results

From the above results we can conclude that the most accurate predictions are obtained for lower values of the parameters p and q. Also, simulating greater MA and AR models with greater values for p and q would be computationally expensive and thus inefficient. Fortunately, there is no need for us to do so, as the best results require smaller values of the aforementioned parameters.

The lowest MSE value was obtained for the AR(1) model (MSE=11.5976), but overall there was little difference in the performance of the ARMA-based model forecasts. To get an idea about how our network performed, table 3.16 shows the actual vs. predicted values for the first 10 forecasted values, using the AR(1) model. As we can see, its performance is very good, the values predicted being very close to the expected values.

Date	*Actual*	*Predicted*
14.02.2020	52.049999	51.369145
16.02.2020	52.180000	51.979570
18.02.2020	52.049999	52.138122
19.02.2020	53.290001	52.022433
20.02.2020	53.779999	53.188660
21.02.2020	53.380001	53.722763
23.02.2020	52.080002	53.370878
24.02.2020	51.430000	52.116446
25.02.2020	49.900002	51.428605
26.02.2020	48.730000	49.939284

Table 3.16: Actual vs. Predicted Values for AR(1)

3.2.4 Neural Networks Forecasting

3.2.4.1 Proposed Architecture – 1

The neural network architecture used in the following simulations is the same as the network used in section 3.1.4.1 and illustrated in fig. 3.7.

Number of Epochs	*MSE*	*RMSE*
1	35.8193	5.9849
2	40.3747	6.3541
3	35.9031	5.9919
4	27.1093	5.2066
5	37.1407	6.0943
10	39.0285	6.2472
25	31.7712	5.6366

Table 3.17: First Architecture Results, Batch Size = 1

In table 3.17 are given the MSE and RMSE scores for a batch size (the number of training samples to work through before the model's internal parameters are updated) of 1, but different values for the epochs (complete passes through the training dataset).

Batch Size	*MSE*	*RMSE*
10	43.2143	6.5737
20	46.7505	6.8374
30	34.3024	5.8568
40	33.2613	5.7672
50	27.2719	5.2222
100	31.5979	5.6212

Table 3.18: First Architecture Results, Number of Epochs=25

In table 3.18 we can see the results obtained with the first architecture, by varying the batch size, keeping the number of epochs constant at the value of 25. It is clear that the best performance of this architecture was obtained for a batch size of 1 and the number of epochs of 4.

In fig. 3.26, 3.27, 3.28, 3.29, 3.30 are shown the results obtained for different values of the number of epochs and the batch size. With the blue colour are represented the *training* data, in our case, the first 80% of the values. With green are represented the data used for *validation*, that is, the last 20% of the values from our dataset. The values are used to compare our *predicted* values, which are illustrated with the colour red.

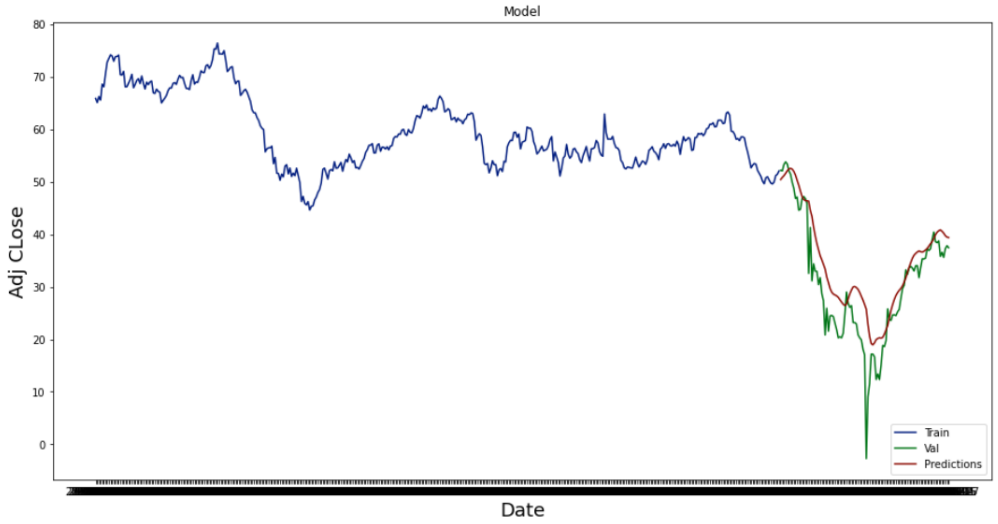

Figure 3.26: First Neural Network Results - Batch Size=1, Epochs=1

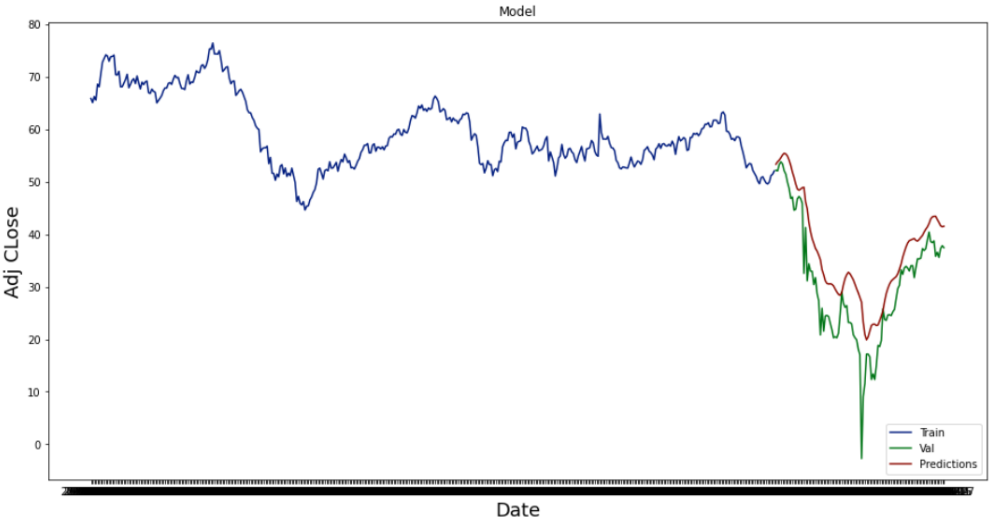

Figure 3.27: First Neural Network Results - Batch Size=1, Epochs=2

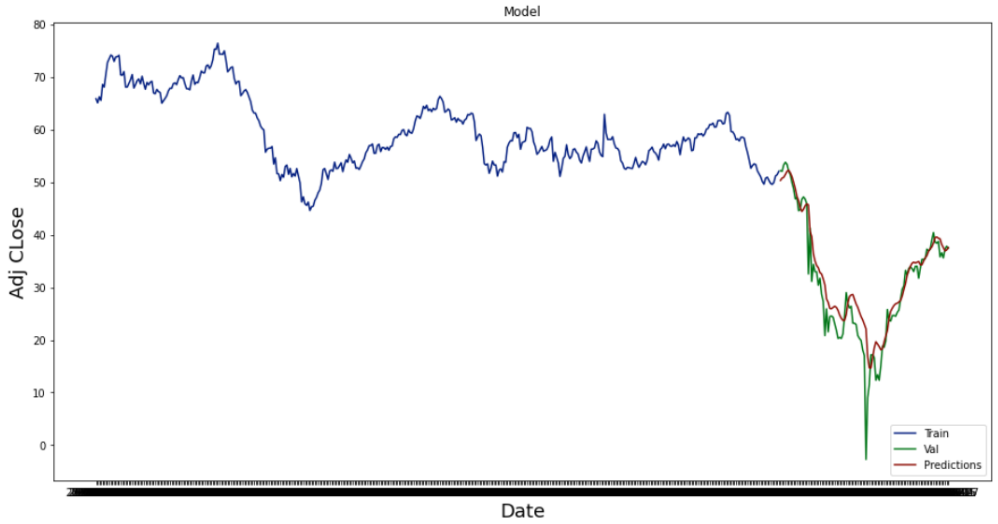

Figure 3.28: First Neural Network Results - Batch Size=1, Epochs=25

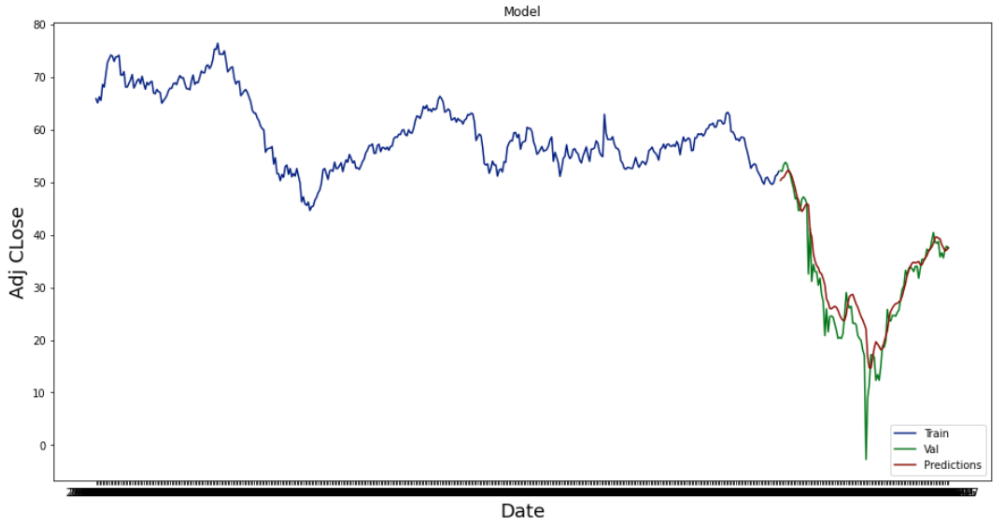

Figure 3.29: First Neural Network Results - Batch Size=1, Epochs=4

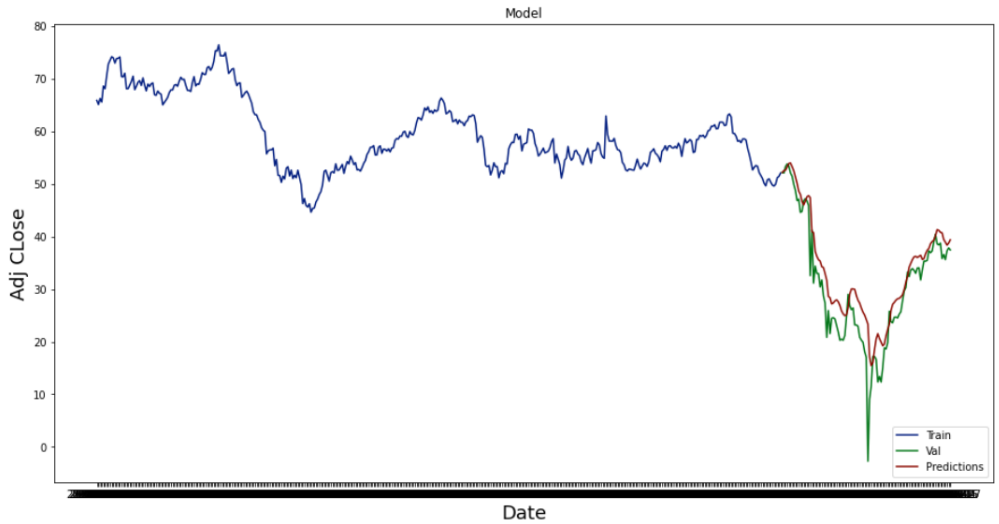

Figure 3.30: First Neural Network Results - Batch Size=50, Epochs=50

As we would have supposed by looking at the MSE and RMSE scores, the predictions made by using a batch size of 1 and 4 epochs, a batch size of 1 and 25 epochs and a batch size of 50 and 50 epochs are the best (in this case the green and the red line are very close to each other; in the rest of the cases they are not quite overlapping).

3.2.4.2 Proposed Architecture – 2

The neural network architecture used in the following simulations is the same as the network used in section 3.1.4.2 and illustrated in fig. 3.13.

Number of Epochs	*MSE*	*RMSE*
1	15.8012	3.9750
2	13.3535	3.6542
3	26.2934	5.1277
4	20.0975	4.4830
5	12.5851	3.5475
10	15.2495	3.9050
25	12.8710	3.5876

Table 3.19: Second Architecture Results, Batch Size = 1

In table 3.19 are given the MSE and RMSE scores for a batch size of 1, but different values for the epochs, for the second neural network architecture.

Batch Size	*MSE*	*RMSE*
10	18.6608	4.3198
20	16.9491	4.1169
30	17.3486	4.1651
40	19.9724	4.4690
50	15.4323	3.9284
100	16.9039	4.1114

Table 3.20: Second Architecture Results, Number of Epochs=25

From tables 3.17, 3.18, 3.19 and 3.20, it is clear that the second neural network architecture outperformed the first one, giving more accurate predictions, thus lower MSE and RMSE scores.

In fig. 3.31 3.32, 3.33, 3.34, 3.35 are shown the results obtained for different values of the number of epochs and the batch size. With the colour blue are represented the training data (first 80% of the data) With green are represented the data used for validation(last 20% of the values). The values are used to compare our predicted values, which are illustrated with the colour red. This architecture gives a very accurate prediction for a batch size of 1 and 5 epochs and a batch size of 1 and 25 epochs, as it can be seen in fig. 3.33, and 3.34 where the green and red line are very hard to be distinguished.

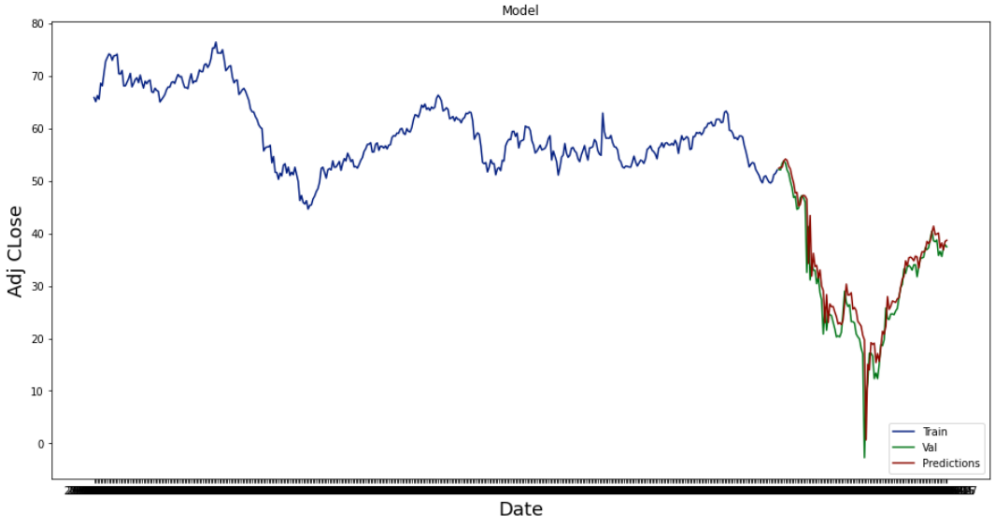

Figure 3.31: Second Neural Network Results - Batch Size=1, Epochs=1

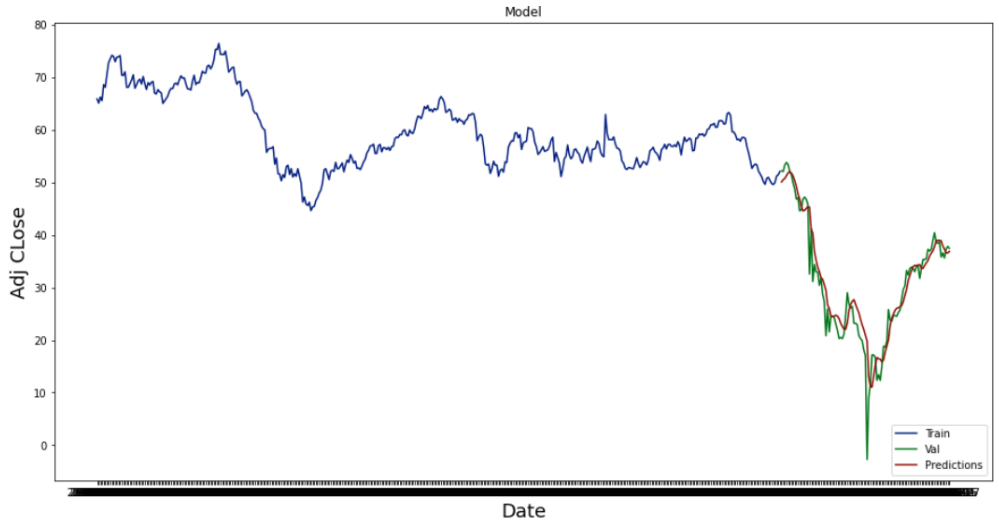

Figure 3.32: Second Neural Network Results - Batch Size=1, Epochs=2

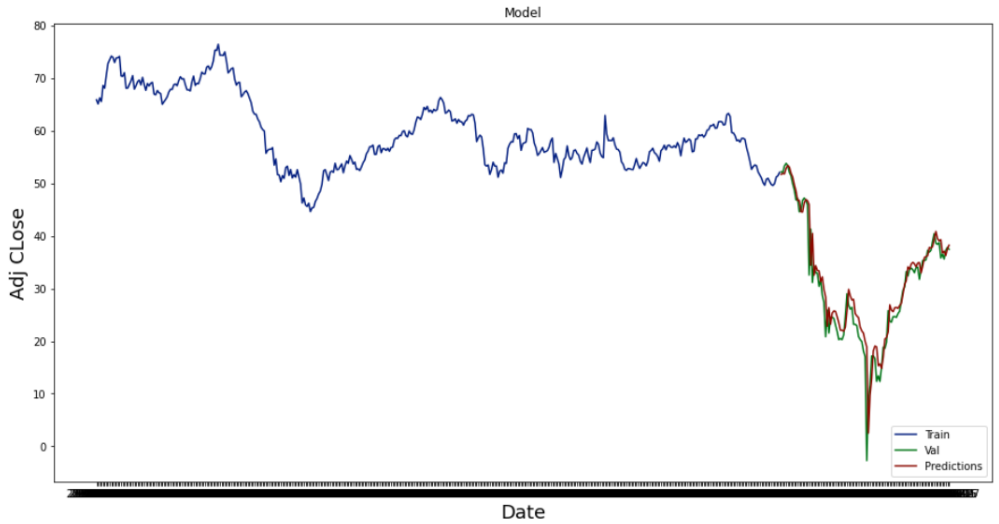

Figure 3.33: Second Neural Network Results - Batch Size=1, Epochs=5

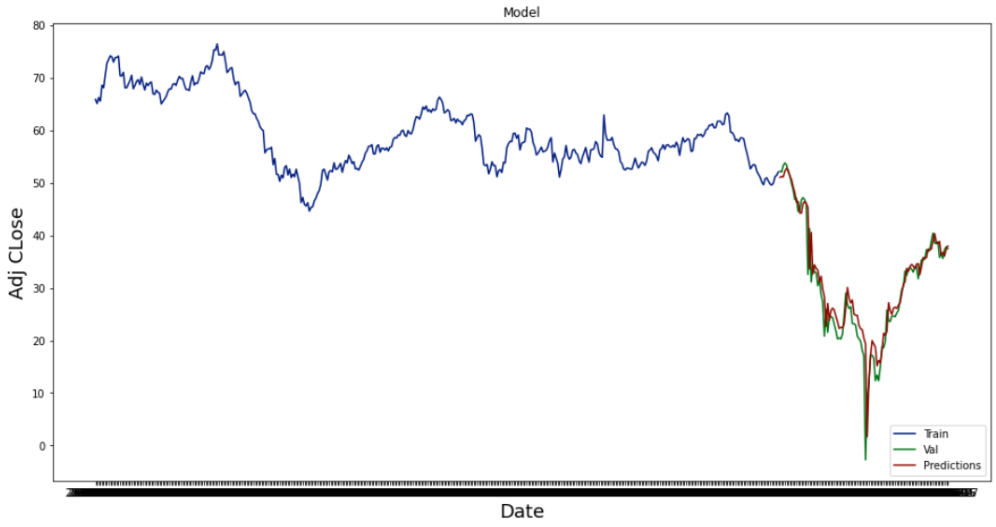

Figure 3.34: Second Neural Network Results - Batch Size=1, Epochs=25

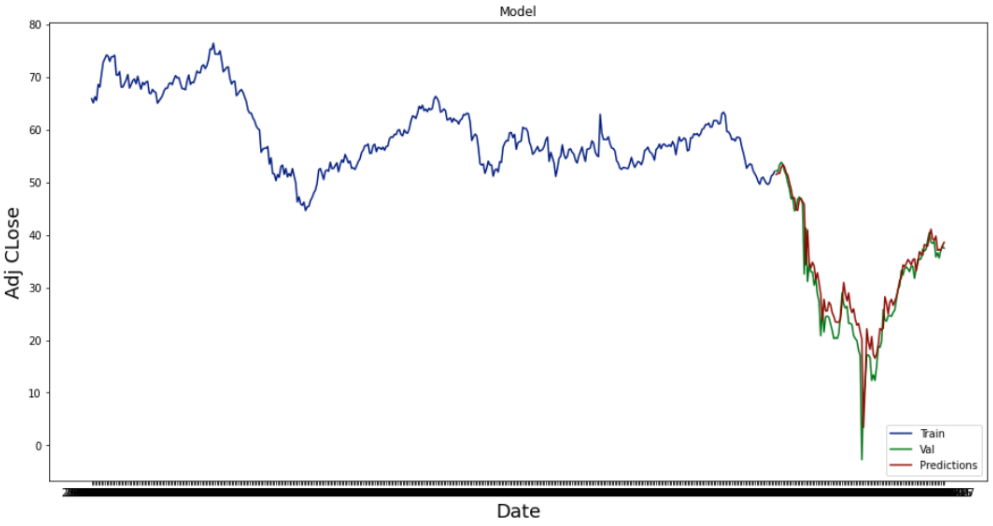

Figure 3.35: Second Neural Network Results - Batch Size=50, Epochs=25

Overall, the second neural network proved to be more effective and more accurate than the first neural network, giving remarkable results on both datasets.

Chapter 4
Conclusions and future steps

4.1 General Conclusions

The main motivation of this thesis was to perform an analysis on financial time series, which are one of the most difficult values to predict. Due to their complicated nature and the great financial rewards associated with them, their prediction is one of the favourite task of many professionals and not only.

Stochastic methods like Moving Average, Autorregresive and Autorregresive Moving Average proved to be very accurate, but also required more statistical analysis and data pre-processing. For higher parameter values and large datasets, they were also very time-consuming.

To solve this problem, Artificial Neural Networks were proposed as an alternative to the stochastic methods. They did not require a statistic analysis of the data to give outstanding performance, but the selection of the architecture was crucial.

Two architectures were analyzed, one already applied to financial time series and one developed by me. They both gave good predictions, however, the last architecture outperformed the first one.

The selection of the Crude Oil Price database was for the purpose of testing the ANNs performance when the series display a chaotic and completely unpredictable behaviour. The crash of the oil price (even below zero), an event completely unexpected some months ago, was successfully modeled by the second neural network, with the right selection of its parameters.

To summarize, ANNs are powerful substitutes for stochastic methods when analyzing financial time series, even though their full potential has not yet been reached and need to be further researched.

4.2 Personal Contributions

For the scope of this thesis, my contributions are the following:

- I did an extensive research on the statistical models used for time series forecasting (AR, MA, ARIMA based models).
- I also searched for databases and simulated in Python examples for seasonality, stationarity and non-stationarity (fig. 1.2, 1.3, 1.4).
- I reviewed the existing literature up to date on spiking neural networks, and various spiking neuron models.

- I simulated using various libraries examples of neurons and their response to multiple types of inputs for the Leaky Integrate-and-Fire model (fig. 2.3, 2.4, 2.5, 2.6), the Hodgkin-Huxley model (fig. 2.8) and the Izhikevich model (fig. 2.10, 2.11).
- I simulated a simple synaptic behaviour between two spiking neurons (fig. 2.13).
- I implemented and simulated the STDP function (fig. 2.14) and the weight change as a function of local variables for the online STDP function (fig. 2.16).
- I designed the method of analyzing the two databases (histograms, statistical parameters, autocorrelation and partial autocorelation functions).
- I analyzed the stationarity of the datasets by implementing the Dickey-Fuller Test and interpreting its results.
- I simulated multiple AR, MA and AMRA models by varying their parameters and calculated the accuracy of their prediction.
- I applied an existing neural network architecture for financial time series to the two databases and did many simulations for different batch size and number of epochs.
- I developed a new neural network architecture for financial time series (with a better performance than the existing one) and applied it to the two databases.

4.3 Future Work

As with every emerging technology, there is still a great amount of work to be done regarding artificial neural networks. In thei context of financial time series analysis, I would like to introduce a financial analysis of the data before training a neural network. Market events and the economy also needs to be taken into account when forecasting time series.

I would also like to find out the correlation between different time series and trying to make a forecast based on how dependent on each other they are. In addition to these, I would implement an algorithm which would buy or sell a stock based on the predictions made by the neural networks.

Likewise, further exploring new architectures and training spiking neural networks on more powerful CPUs and GPUs is also a step to be done in the future.

References

[1] *Current COVID-19 situation - COVID 19 graph —& data*. Available from: https://covidgraph.com/ [Accessed 15th May 2020].

[2] P. L. Bernstein. *Against the Gods: The Remarkable Story of Risk*. (John Wiley & Sons. Inc., United States of America, 1998).

[3] *History of The Stock Market. How to Model Volatility with ARCH and GARCH for Time Series Forecasting in Python*. Available from: https://bebusinessed.com/history/history-of-the-stock-market/ [Accessed 31st May 2020].

[4] D. C. Montgomery, C. L. Jennings, M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. (John Wiley Sons. Inc., Hoboken, New Jersey, 2008).

[5] USGS. *ComCat Documentation - Event Terms*. Available from: https://earthquake.usgs.gov/data/comcat/data-eventterms.php#time [Accessed 15th May 2020].

[6] NIST/SEMATECH. *e-Handbook of Statistical Methods*. Available from: http://www.itl.nist.gov/div898/handbook/ [Accessed 15th May 2020].

[7] Ed Cohen and Andrew Walden. *Lecture Notes for Course MATH96053 Time Series Analysis*, Autumn term 2019.

[8] *Daily-Min-Temperatues.csv*. Available from: https://github.com/selva86/datasets/blob/master/daily-min-temperatures.csv [Accessed 15th May 2020].

[9] Shay Palachy. *Stationarity in time series analysis*. Available from: https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322 [Accessed 16th May 2020].

[10] Tavish Srivastava. *A Complete Tutorial on Time Series Modeling in R*. Available from: https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/ [Accessed 16th May 2020].

[11] Jason Brownlee. *How to Check if Time Series Data is Stationary with Python*. Available from: https://machinelearningmastery.com/time-series-data-stationary-python/ [Accessed 16th May 2020].

[12] *Gaussian White Noise*. Available from: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780471679370.app2 [Accessed 16th May 2020].

[13] *Air Passengers*. Available from: https://www.kaggle.com/abhishekmamidi/air-passengers [Accessed 16th May 2020].

[14] R. Yaffee, M. McGee. *Introduction to Time Series Analysis and Forecasting with Applications of SAS and SPSS*. (ACADEMIC PRESS, INC., San Diego London Boston New York Sydney Tokyo Toronto, 1999).

[15] R. AdhikariR. K. Agrawal. *An Introductory Study on Time Series Modeling and Forecasting*.

[16] Jason Brownlee. *How to Model Volatility with ARCH and GARCH for Time Series Forecasting in Python*. Available from: https://machinelearningmastery.com/develop-arch-and-garch-models-for-time-series-forecasting-in-python/ [Accessed 30th May 2020].

[17] H. Adeli S. Ghosh-Dastidar. SPIKING NEURAL NETWORKS. *International Journal of Neural Systems*, **19**(4):295–308, (2009).

[18] W. Gerstner, W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. (Cambridge University Press, 2002).

[19] E. M. Izhikevich. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, **14**(6):1569–1572, (2003).

[20] Chinmay Chiplunkar. *An Introduction to Spiking Neural Networks (Part 1)*. Available from: https://medium.com/analytics-vidhya/an-introduction-to-spiking-neural-networks-part-1-the-neurons-5261dd9358cd [Accessed 22th May 2020].

[21] H. Adeli S. Ghosh-Dastidar. A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks*, (22):1419–1431, (2009).

[22] Brian authors. *Introduction to Brian part 2: Synapses*. Available from: https://brian2.readthedocs.io/en/stable/resources/tutorials/2-intro-to-brian-synapses.html [Accessed 25th May 2020].

[23] *Hebbian theory*. Available from: https://en.wikipedia.org/wiki/Hebbian_theory [Accessed 25th May 2020].

[24] *Spike-timing-dependent plasticity*. Available from: https://en.wikipedia.org/wiki/Spike-timing-dependent_plasticity [Accessed 25th May 2020].

[25] J. Sjöström and W. Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2):1362, 2010. revision #184913.

[26] D. Waitzman. Standard for the transmission of ip datagrams on avian carriers, 1990.

[27] *AKHILESH GANTI. Foreign Exchange Market*. Available from: https://www.investopedia.com/terms/forex/f/foreign-exchange-markets.asp [Accessed 7th June 2020].

[28] *CompareRemit. 8 Key Factors that Affect Foreign Exchange Rates*. Available from: https://www.compareremit.com/money-transfer-guide/key-factors-affecting-currency-exchange-rates/ [Accessed 7th June 2020].

[29] *Fares Sayah. Stock Market Analysis + Prediction using LSTM*. Available from: https://www.kaggle.com/faressayah/stock-market-analysis-prediction-using-lstm [Accessed 8th June 2020].

[30] *PRABLEEN BAJPAI. Top Factors That Affect the Price of Oil*. Available from: https://www.investopedia.com/articles/investing/072515/top-factors-reports-affect-price-oil.asp [Accessed 18th June 2020].