Deep Q-learning with PCA and Prioritized Experience Replay for Trading in the Forex Market

João Manuel Guerreiro Barra Feijão

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. Rui Fuentecilla Maia Ferreira Neves

Examination Committee
Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Supervisor: Prof. Rui Fuentecilla Maia Ferreira Neves
Member of the Committee: Prof. Alexandra Sofia Martins de Carvalho

April 2018
“A lot of lessons will be learned and a lot money will be lost, before a lot of money can be made.”

Peter Denious
Acknowledgments

First of all, I would like to thank my family, specially my mom, dad and sister, for the support and encouragement. Had not been for them and I would not be where I am.

I would like to thank my supervisor, Prof. Rui Neves, for the orientation and total availability, and for the knowledge, experience, incentive and patience which guided me in the development of this work.

A kind word goes for all my colleagues and friends, who shared the last five years of this unique challenge, and made the hard work more enjoyable, and all the others from my academic path.
Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
Resumo

Esta tese apresenta uma abordagem que combina Análise Técnica e Análise de Componentes Principais (ACP) com uma Rede Q-profunda (RQP) que usa Dupla Q-aprendizagem e Repetição de Experiência Priorizada, para gerar decisões de investimento capazes de alcançar ganhos financeiros a longo prazo com baixo risco financeiro. A abordagem usa Análise Técnica para extrair características dos dados financeiros. O PCA transforma as características extraídas que têm uma dimensão muito alta em características com baixa dimensão mantendo a essência das características originais. O DQN usa as características de baixa dimensão para aprender padrões de mercado abstractos e robustos e aprende a prever as melhores decisões de investimento para os estados do mercado. A abordagem proposta foi testada com dados financeiros reais de cinco mercados Forex com diferentes características entre si e usando custos de transação. Dois diferentes tipos de funções de estado e três diferentes funções de recompensa foram propostas para o algoritmo DQN. Os resultados mais robustos são produzidos por uma combinação de uma função de estado usando as características de baixa dimensão produzidas pelo PCA e uma função de recompensa que mede a percentagem de lucro obtida pelo sinal de transação gerado num período de investimento. O sistema usa em todos os testes a mesma topologia da rede neuronal e os mesmos valores dos hiperparâmetros do sistema. Os resultados alcançados mostram que esta abordagem supera a estratégia de Buy and Hold (BnH) em quatro dos cinco mercados testados. No mercado do EUR-USD, o sistema atinge uma taxa de retorno de 19.21%, enquanto que a estratégia de BnH apenas atinge 3.92%. Além disso, concluiu-se que a técnica de PCA, a Repetição de Experiência Priorizada e a Dupla Q-aprendizagem são essenciais para o desempenho da abordagem proposta.

Palavras-chave: Mercados Forex, Análise Técnica, Análise de Componentes Principais (PCA), Aprendizagem Profunda, Rede Q-profunda (DQN), Dupla Q-aprendizagem, Repetição de Experiência Priorizada.
Abstract

This thesis presents an approach combining Technical Analysis and Principal Component Analysis (PCA) with a Deep Q-network (DQN) using Double Q-learning and Prioritized Experience Replay to generate trading decisions capable of achieving long-term financial gains with low financial risk. The approach uses Technical Analysis to extract features from raw financial data. PCA transforms the extracted high dimensional features into a low dimensional features while maintaining the essence of the original features. DQN uses the low dimensional features to learn abstract and robust market patterns and learn to predict the best trading actions for the market status. The proposed approach is tested with real data from five Forex markets with different characteristics from each other, using transaction costs. Two different state and three different reward functions are considered in the DQN algorithm. The most robust results are produced by a combination of a state representation using features produced by PCA and a reward function measuring the profit percentage obtained by the generated trading signal in a trading period. The system uses on every test the same network topology and system’s hyperparameters values. The results achieved show that this approach outperforms the Buy and Hold (BnH) strategy in four of the five tested markets. In the EUR-USD market, this system achieves a rate of return of 19.21% while the BnH strategy only achieves 3.92%. Furthermore, it is concluded that the PCA technique, Prioritized Experience Replay and Double Q-learning are essential to the performance of the proposed approach.

Keywords: Forex markets, Technical Analysis, Principal Component Analysis (PCA), Deep Learning, Deep Q-network (DQN), Double Q-learning, Prioritized Experience Replay.
# Contents

Acknowledgments ........................................................................ v
Declaration ............................................................................. vii
Resumo ..................................................................................... ix
Abstract ................................................................................... xi
List of Tables ........................................................................... xvii
List of Figures .......................................................................... xix
Nomenclature ........................................................................... xxi

1 Introduction ........................................................................ 1
   1.1 Background .................................................................... 1
   1.2 Motivation ..................................................................... 1
   1.3 Work Purpose .................................................................. 2
   1.4 Main Contributions ....................................................... 3
   1.5 Thesis Outline ............................................................... 3

2 Theoretical Background ....................................................... 4
   2.1 Financial Market ........................................................... 4
   2.2 Financial Trading .......................................................... 4
   2.3 Market Analysis ............................................................ 5
      2.3.1 Market Trends ......................................................... 5
      2.3.2 Fundamental Analysis ............................................ 6
      2.3.3 Technical Analysis ............................................... 6
   2.4 Reinforcement Learning ................................................ 16
      2.4.1 Definition of the Problem ..................................... 17
      2.4.2 State ...................................................................... 18
      2.4.3 Goals and Rewards .............................................. 19
      2.4.4 Returns ................................................................. 20
      2.4.5 Value Functions .................................................... 20
      2.4.6 Optimal Value Functions and Optimal Policies ....... 21
      2.4.7 Model ................................................................. 23
      2.4.8 Learning Methods ............................................... 23
4.5.3 Case Study II.c - Influence of Double Q-learning ........................................ 78
4.6 Case Study III - Performance Comparison ...................................................... 78

5 Conclusions ........................................................................................................ 81
5.1 Conclusions .................................................................................................... 81
5.2 Future Work .................................................................................................. 82

Bibliography ........................................................................................................... 83

A Return Plots of Case Study I ........................................................................... 87
A.1 USD-JPY Exchange Market ........................................................................ 87
A.2 AUD-USD Exchange Market ....................................................................... 88
A.3 USD-CAD Exchange Market ...................................................................... 89

B Return Plots of Case Study II ......................................................................... 90
B.1 USD-JPY Exchange Market ........................................................................ 90
B.2 AUD-USD Exchange Market ....................................................................... 91
B.3 USD-CAD Exchange Market ...................................................................... 92

C Return Plots ....................................................................................................... 93
C.1 EUR-USD Exchange Market ......................................................................... 93
C.2 USD-JPY Exchange Market ......................................................................... 94
C.3 AUD-USD Exchange Market ....................................................................... 95
C.4 GBP-USD Exchange Market ....................................................................... 96
C.5 USD-CAD Exchange Market ...................................................................... 97
C.6 EUR-USD 2014 Exchange Market ................................................................. 98
C.7 EUR-USD 2015 Exchange Market ................................................................. 99
C.8 EUR-USD 2016 Exchange Market ................................................................. 100
List of Tables

2.1 Results of some studies relevant for this thesis. ................................................. 44
3.1 List of variables fed to the technical analysis module. ........................................ 48
3.2 List of the 26 computed technical indicators. .................................................. 49
3.3 Principal components of the 30 features obtained from the EUR-USD market data (period of 12/06/2003 to 05/12/2012). ................................................................. 52
3.4 List of hyperparameters and their values. ......................................................... 65
3.5 System's actions and their trading effect. .......................................................... 66
4.1 Number of experimented hidden layers and neurons in a layer. ......................... 70
4.2 Results of the Buy and Hold (BnH) strategy and the different state and reward functions combinations (average). ................................................................. 71
4.3 Results of the Buy and Hold strategy and the final system, and without PCA, without Prioritized Experience Replay, and without Double Q-learning (average). .... 76
4.4 Results of the Buy and Hold (BnH) strategy, the final proposed system and of Carapuço system when trading in the EUR-USD market in the different time periods (average). . . . 80
List of Figures

2.1 The 10-day SMA and 10-day EMA applied to the S&P 500 index close prices. . . . . . . . 8
2.2 MACD applied to the S&P 500 index close prices. . . . . . . . . . . . . . . . . . . . . . . 11
2.3 RSI applied to the S&P 500 index close prices. . . . . . . . . . . . . . . . . . . . . . . . . 12
2.4 OBV applied to the S&P 500 index. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.5 The agent-environment interaction in Reinforcement Learning. . . . . . . . . . . . . . . . 18
2.6 Simple grid-world environment with the immediate rewards. . . . . . . . . . . . . . . . . 26
2.7 One optimal policy $\pi^*$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
2.8 The $V^*(s)$ values. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.9 The $Q(s,a)$ values. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.10 Operation of the Q-learning algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.11 Simple fully-connected feedforward neural network. . . . . . . . . . . . . . . . . . . . . . 31
2.12 Mathematical model of a neuron. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
2.13 Activation functions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
2.14 Neural Network for functional dependencies example. . . . . . . . . . . . . . . . . . . . . 33
2.15 Neural Network for backpropagation example. . . . . . . . . . . . . . . . . . . . . . . . . 34
3.1 System's Architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.2 S&P500 historical data on .csv format file. . . . . . . . . . . . . . . . . . . . . . . . . . . 48
3.3 Example of the Min-Max normalization method. . . . . . . . . . . . . . . . . . . . . . . . 50
3.4 Example of the features state representation for a third trading time step. . . . . . . . . 54
3.5 Example of the historical state representation for the third trading time step. . . . . . . . 55
3.6 Prioritized Experience Replay sampling example. . . . . . . . . . . . . . . . . . . . . . . . 59
3.7 System learning and test architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
3.8 Different Paths. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
3.9 Trading execution example in EUR-USD market from 27/11/17 to 15/12/17. . . . . . . . . 66
4.1 Returns obtained by Features-period, Historical-profit, system with the worst performance (Features-cost) and the Buy and Hold strategy in the EUR-USD exchange market. . . . . . . 74
4.2 Returns obtained by Features-period, Historical-profit, system with the worst performance (Features-cost) and the Buy and Hold strategy in the GBP-USD exchange market. . . . . . . 74
4.3 Returns obtained by the BnH strategy and the average returns obtained by the final system, and without PCA, without Prioritized Experience Replay (PER) and without Double Q-learning in the EUR-USD exchange market. .................................................. 79

4.4 Returns obtained by the BnH strategy and the average returns obtained by the final system, and without PCA, without Prioritized Experience Replay (PER) and without Double Q-learning in the GBP-USD exchange market. .................................................. 79

A.1 Average returns obtained by Features-period, Historical-profit and Features-profit systems and Buy and Hold returns in the USD-JPY exchange market. .............................. 87

A.2 Average returns obtained by Features-period, Historical-profit, the system with the worst performance (Features-cost) and Buy and Hold returns in the AUD-USD exchange market. 88

A.3 Average returns obtained by Features-period, Historical-profit, the system with the worst performance (Historical-cost) and Buy and Hold returns in the USD-CAD exchange market. 89

B.1 Returns obtained by the BnH strategy, final system and the final system without the components in the USD-JPY exchange market. .................................................. 90

B.2 Returns obtained by the BnH strategy, final system and the final system without the components in the AUD-USD exchange market. .................................................. 91

B.3 Returns obtained by the BnH strategy, final system and the final system without the components in the USD-CAD exchange market. .................................................. 92

C.1 Best and average returns obtained by the system and Buy and Hold returns in the EUR-USD exchange market. .................................................. 93

C.2 Best and average returns obtained by the final system and Buy and Hold returns in the USD-JPY exchange market. .................................................. 94

C.3 Best and average returns obtained by the final system and Buy and Hold returns in the AUD-USD exchange market. .................................................. 95

C.4 Best and average returns obtained by the final system and Buy and Hold returns in the GBP-USD exchange market. .................................................. 96

C.5 Best and average returns obtained by the final system and Buy and Hold returns in the USD-CAD exchange market. .................................................. 97

C.6 Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market during 2014 used for performance comparison. .... 98

C.7 Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market during 2015 used for performance comparison. .... 99

C.8 Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market during 2016 used for performance comparison. .... 100
Nomenclature

AI     Artificial Intelligence
Forex Foreign Exchange
RL     Reinforcement Learning
NN     Neural Network
ANN    Artificial Neural Network
DNN    Deep Neural Network
PCA    Principal Component Analysis
DQN    Deep Q-network
MDP    Markov decision process
TD     Temporal-Difference
RRL    Recurrent Reinforcement Learning
NEAT   Neuro Evaluation of Augmenting Topologies
FFNN   Feedforward Neural Network
RNN    Recurrent Neural Network
ReLU   Rectifier Linear Unit
TanH   Hyperbolig Tangent
MSE    Mean Square Error
RMSProp Root Mean Square Propagation
CAD    Canadian Dollar
EUR    Euro
USD    United States Dollar
JPY    Japanese Yen
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUD</td>
<td>Australian Dollar</td>
</tr>
<tr>
<td>GBP</td>
<td>Great Britain Pound</td>
</tr>
<tr>
<td>BnH</td>
<td>Buy and Hold</td>
</tr>
<tr>
<td>SMA</td>
<td>Simple Moving Average</td>
</tr>
<tr>
<td>EMA</td>
<td>Exponential Moving Average</td>
</tr>
<tr>
<td>PSAR</td>
<td>Parabolic Stop and Reversal</td>
</tr>
<tr>
<td>ATR</td>
<td>Average True Range</td>
</tr>
<tr>
<td>MACD</td>
<td>Moving Average Convergence Divergence</td>
</tr>
<tr>
<td>PPO</td>
<td>Percentage Price Oscillator</td>
</tr>
<tr>
<td>RSI</td>
<td>Relative Strength Index</td>
</tr>
<tr>
<td>ADX</td>
<td>Average Directional Index</td>
</tr>
<tr>
<td>CCI</td>
<td>Commodity Channel Index</td>
</tr>
<tr>
<td>ROC</td>
<td>Rate of Change</td>
</tr>
<tr>
<td>OBV</td>
<td>On Balance Volume</td>
</tr>
<tr>
<td>MFI</td>
<td>Money Flow Index</td>
</tr>
<tr>
<td>TR</td>
<td>True Range</td>
</tr>
<tr>
<td>TP</td>
<td>Typical Price</td>
</tr>
<tr>
<td>DI</td>
<td>Directional Indicator</td>
</tr>
<tr>
<td>DM</td>
<td>Directional Movement</td>
</tr>
<tr>
<td>ROR</td>
<td>Rate of Return</td>
</tr>
<tr>
<td>RRR</td>
<td>Risk Return Ratio</td>
</tr>
<tr>
<td>MDD</td>
<td>Max Drawdown</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

From the beginning of mankind it has been man’s common goal to make life easier. The prevailing notion in society is that wealth brings comfort and luxury. Financial markets have always moved vast amounts of capital arousing interest to the possibilities within. As the years gone by the world suffered tremendous evolution and nowadays it is becoming easier for any person to trade in financial markets. Also, huge amounts of data are available in many fields of knowledge. Analyzing these large quantities of data and finding patterns within such data is in most cases a complex and not trivial problem. To face these challenges, Machine Learning emerged as a field of interest in automated methods capable of learning and make predictions based on data without being explicitly programmed.

Machine learning is a subfield of Computer Science, related to Artificial Intelligence (AI), and even has strong ties to Mathematical optimization and Statistics. In contrast to the traditional expert based approach, in which a computer is told exactly what to do in response to a given data, these type of approaches are not told how to behave in certain situations, they learn from the data and figure it out its own solution to the problem. Besides, given enough data and time, an appropriate machine learning method may even be able to discover complex nonlinear relationships through learning from the data [1]. Its applications range many fields from robot locomotion to speech and handwriting recognition.

1.2 Motivation

The basic motivation to perform financial trading is financial gain. Financial market trading is an application domain with large potential for machine learning. The existence of an enormous amount of historical data suggests that machine learning can provide a competitive advantage over human inspection of these data. So, it is not surprising at all that one of the biggest challenges in our current society is building an intelligent system that can outperform financial markets by carefully produce timely trading suggestions for financial investments. However, this is very difficult because financial markets are affected by too many highly interrelated factors, such as economic, political and investors psychological
factors. Also, financial markets have noisy, non-stationary and non-linear characteristics.

The Efficient Market Hypothesis, developed by Burton G. Malkiel [2], indicates that it is impossible to “beat the market”, because market efficiency causes existing prices to always incorporate and reflect all relevant information. All the changes in prices of financial markets are based on immediate economic events or news, with all future prices not following any trend or pattern. Therefore, outperforming the overall market throughout expert market timing is impossible, with the Buy and Hold (BnH) investing strategy, where an investor buys assets and holds them for a long time regardless of market fluctuations, being the best investment strategy. Despite this theory, it is also believed that the markets are inefficient enabling opportunities to obtain gains above the BnH strategy. This is mainly due to the markets not responding immediately to newly released information and due to the investors psychological factors affecting the markets.

For many years, market analysis such as fundamental and technical analysis were the most popular approaches used to predict and forecast market movements. However, in the last decades it was surpassed by the growth in the application of various machine learning methods and techniques to financial trading. Artificial Neural Networks, Support Vector Machines, Genetic Algorithms and Reinforcement Learning are among the most popular approaches.

With the breakthrough of Deep Learning, a field of machine learning, it is possible to extract high-dimensional features directly from raw inputs. A recent work published by Google DeepMind [3] described a system that combining deep learning methods and reinforcement learning, termed deep Q-network (DQN), created a system that can learn successful policies directly from high-dimensional sensory inputs. The system was tested on a large set of classic Atari 2600 games, learning to understand which moves are good and which are bad basing solely on the visual data. The system was able to master a number of different games, surpass the performance of all previous algorithms and play most of them better than a human player. The most famous example of application of the DQN system is AlphaGo, a system that defeated the highest ranking professional Go player of the world, Lee Sedol. Before this match, the game of Go was regarded as too hard for modern technology, because of the immense state-space. Since that work, some extensions to the DQN system were proposed by Google DeepMind, being the Double Q-learning [4] and the Prioritized Experience Replay [5] two of them.

Since then, DQN has been successfully applied in many fields. Recently, Carapuço [6] applied DQN to Forex trading and concluded that the system was suitable to trade in the EUR-USD market. This approach solely relied on feature extraction to summarize the market environment.

1.3 Work Purpose

The main goal of this thesis is to develop a DQN system framework, using Double Q-learning and Prioritized Experience Replay extensions, capable of maximizing financial gains by timely selecting the best trading actions while trading in Forex markets. The framework will use the same network topology, system’s hyperparameters values and raw financial data parameters in every market.

To achieve this goal, technical analysis will be used to extract features from the raw financial data
in order to mitigate data noise and uncertainty enhancing the learning of market patterns and trends. However, instead of solely relying on feature extraction, a feature learning method, Principal Component Analysis (PCA), will be used to learn features automatically from the raw financial data and from the features extracted by the technical analysis. It will also transform the high dimensional extracted features to a lower dimensional features that are going to be the DQN algorithm inputs. The purpose of the DQN algorithm is to use deep learning to learn abstract and robust market patterns from the input features and use Q-learning algorithm to predict the best trading actions for the market status. Different state and reward functions of the DQN algorithm will be experimented to find the most suitable combination for maximizing returns while avoiding transaction costs and minimizing financial risk. To prove the robustness of the system's framework the approach will be severely tested in five different Forex markets using real market conditions and observing if it is capable of outperforming the BnH strategy.

1.4 Main Contributions

The main contributions of this thesis are:

- A DQN system applied to financial markets relying on feature learning. Features are learned automatically from the raw financial data and from features extracted from the raw financial data using technical analysis.

- The application of a DQN system using Double Q-learning and Prioritized Experience Replay to trade in financial markets.

- A DQN framework capable of maximizing financial gains in many Forex markets using the same network topology, system’s hyperparameters values and raw financial data parameters, supported by the positive results in five currency pairs over a large time frame.

1.5 Thesis Outline

This thesis is structured as follows:

- Chapter 2 addresses the theoretical background of the developed work, namely the concepts of financial trading, market analysis, reinforcement learning, neural networks, deep learning, principal component analysis and deep Q-networks.

- Chapter 3 presents the architecture and describes each of the components that constitute the implemented solution.

- Chapter 4 describes the metrics used to evaluate the developed solution, the topology study and results obtained.

- Chapter 5 presents the conclusions of this thesis and suggestions for future work.
Chapter 2

Theoretical Background

This chapter presents some useful concepts and theoretical foundations necessary to understand the developed work. First, the domain relative to financial markets and fundamentals about financial trading and market analysis will be addressed. Next, the machine learning algorithms that are the base for the trading system and the importance of deep learning will be explained. A feature learning procedure that performs a linear mapping of the data to a lower-dimensional space will also be introduced. Finally, the related studies using these algorithms will be analyzed.

2.1 Financial Market

Normally, market means the aggregate of possible buyers and sellers of a certain good or service and the transactions between them.

A financial market is a market in which people trade financial securities, commodities, currencies and other fungible items of value at low transaction costs and at prices that reflect supply and demand. Securities include stocks and bonds, and commodities include precious metals or agricultural products.

The foreign exchange (Forex) market is a global decentralized market for the trading of currencies. This includes all aspects of buying, selling and exchanging currencies at current or determined prices. The currencies are always traded in pairs with the market determining a relative value by setting the market price of one currency if paid with another. For example, in the EUR-USD market, EUR is the base currency and 1 EUR is worth X USD.

2.2 Financial Trading

The basic motivation to perform financial trading is financial gain. In order to accomplish this goal many types of approaches and techniques can be used. Two of the most known techniques are Speculation and Investment. The distinction between Speculation and Investment has been disappearing. However, the amount of risk undertaken in a transaction can be classified as the difference between them. Graham [7] defined "Investment operation is one which, upon thorough analysis promises safety
of principal and an adequate return. Operations not meeting these requirements are speculative”. Speculation can be defined as the purchase of an asset with the hope that it will become more valuable and profit from the short term fluctuations rather than attempting to profit from the underlying financial attributed embodied in the financial instruments such as capital gains, dividends, or interest for stocks - safety of principal.

The work here developed can be classified as a speculator because the system cares only about maximizing profits from short-term fluctuations without any safety of principal. However, this work proposes to use speculation with some type of prediction capability.

In financial trading an investor can assume three positions: long, short or neutral. In Forex trading a long position means the first currency is bought, hoping that the market price will rise. A long position is expressed in terms of the base currency. In contrast, a short position occurs when the first currency is sold while the second currency is bought, hoping for a decline in the market price. A short position is also expressed in terms of the base currency. For example, a trader going short in the EUR-USD would be selling Euro’s and buying USD. If, however, the trader went long the currency pair it would be buying Euro’s and selling USD. A neutral position is taken when the investor expects the market price to neither increase nor decrease in the near future opting to stay out of the market. In practice, it can be done by buying the same exact value of the first and second currencies of the exchange pair. Changing the trading position has a transaction cost that depends on the corresponding financial market and broker.

2.3 Market Analysis

Market analysis is responsible for the study and evaluation of a particular instrument, an investment sector or the market as a whole in an attempt to determine what will happen in the future and profit from these predictions. This is a big challenge because financial markets are an example of a signal which is dynamic, noisy, non-stationary, and non-linear [8]. Most of the investors analyze the market using handcrafted financial features in order to mitigate the noise and being capable of summarizing the market conditions.

Mainly there are two types of market analysis, the fundamental analysis and the technical analysis. Both of these approaches to market forecasting attempt to solve the same problem, that is, to determine the direction prices are likely to move, but approach the problem from a different way. The fundamentalist studies the cause of market movement while the technician studies its effect.

2.3.1 Market Trends

Before explaining the fundamental and technical analysis concepts in more detail, it is necessary to introduce the concept of market trends. In a general sense, the trend is simply the direction of the market, which way it’s moving. However, markets don’t generally move in a straight line in any direction, their moves are characterized by a series of zigzags. These zigzags resemble a series of successive waves with fairly obvious peaks and troughs. It is the direction of those peaks and troughs that constitutes
market trend.

There are three types of trends: upward trend, downward trend and sideways trend. An upward trend is a series of successively higher peaks and troughs. The opposite situation, with successively lower peaks and troughs, defines a downward trend. Finally, horizontal peaks and troughs would identify a sideways trend that reflects a period of equilibrium in the price level where the forces of supply and demand are in a state of relative balance.

It is also common to distinguish bull market, when the prices are rising or are expected to rise, and bear market, when the prices are falling or are expected to fall.

### 2.3.2 Fundamental Analysis

Fundamental analysis is a method of evaluating a financial instrument in an attempt to measure its intrinsic value, by studying related economic, financial and other qualitative and quantitative factors. Studies anything that can affect the instrument's value, including microeconomic factors such as the instrument's management, financial conditions and macroeconomic factors such as the overall economy and industry conditions.

The end goal of fundamental analysis is to produce a quantitative value that an investor can compare with the instrument's current price. If the current instrument price is lower than the quantitative value, then the instrument is undervalued, and otherwise if the current instrument price is higher than the quantitative value the instrument is overvalued. This type of analysis takes in credit that the market instrument's are mispriced in the short-term but that the instrument price will correct itself over the long term appreciating or depreciating accordingly, making this type of analysis not suitable for the purposed system that attempts to profit from short-term fluctuations. To take advantage of the market misprice the investor should buy an instrument when it is undervalued and sell it when it is overvalued, similar to the very known “Buy low, sell high” strategy. The investment based on this analysis was made famous by Graham [7] and has W. Buffett has his most notable follower [9, 10].

In Forex markets, fundamental analysis looks at the effect of the economic indicators which can eventually determine the exchange rate.

### 2.3.3 Technical Analysis

Technical analysis consists in the study of market action, primarily through the use of charts, for the purpose of forecasting future price trends. The two main sources of information are the price and the volume. The technical approach is based on the philosophy that anything that can possibly affect the price is actually reflected in the price of that market, including political, psychological and fundamental factors. It claims that price actions reflects shifts in supply and demand. If demand exceeds supply, prices should rise. If supply exceeds demand, prices should fall. The charts do not in themselves cause markets to move up or down, they simply reflect the bullish or bearish psychology of the marketplace that has the underlying forces of supply and demand and also the economic fundamentals of that market.

There are other two essential concepts of the technical approach. The first assumes that the prices
move in trends. In fact, most of the techniques used in this approach are trend-followers, meaning that their intent is to identify and follow existing trends. The other concept is that history tends to repeats itself.

In terms of criticism, the last concept is the main focus. The Random Walk Theory [2], based on the efficient market hypothesis, claims that price movement is random and unpredictable and that price history is not a reliable indicator of future price direction.

**Technical Indicators** A technical indicator is a metric whose value is calculated from the price or the volume of an asset. Technical indicators are a fundamental part of technical analysis as they try to predict the future price or future trend based on their value. Contrary to fundamental indicators, technical indicators are easily accessible to anyone as they are deducted from price or volume and these data are available in huge quantities. The technical indicators that are used in this work are presented next. A more detailed description of technical analysis and technical indicators can be found in [11].

**Moving Averages** Are indicators that help to smooth out price series by filtering out the "noise" from random price fluctuations, making it much easier to view the underlying trend. The term moving is used because only the latest defined time period’s prices are used in the calculation. Therefore, the body of data to be averaged moves forward with each new trading time period. A moving average never anticipates, only reacts. Its purpose is to identify or signal that a new trend has begun or that an old trend has ended or reversed.

Different periods of time can be used to calculate a moving average, with shorter moving averages much more sensitive to the price action and usually used for short-term investments. Longer range averages are less sensitive and usually used for long-term investments. The price most commonly used in simple moving average construction is the closing price.

The Simple Moving Average (SMA) represents the mean value of the closing prices over a certain amount of time, usually days. Equation (2.1) presents the formula of the SMA indicator where \( n \) refers to the number of time periods, \( t \) to the current time and \( \text{Close} \) to the Closing price:

\[
SMA_n = \frac{\sum_{i=t-n}^{t} \text{Close}_i}{n}.
\]  

(2.1)

Exponential Moving Average (EMA) also presents the mean value of the closing prices over a certain amount of time but assigns a greater weight to the more recent data, reacting faster to recent price changes than a SMA. Equation (2.2) presents the formula of the EMA indicator where \( n \) refers to the number of time periods, \( t \) to the current time and \( \rho \) to the smoothing factor:

\[
EMA_n = \begin{cases} 
\text{Close}_t & \text{if } t = 1 \\
\text{Close}_t \times \rho + EMA_{n-1} \times (1 - \rho) & \text{if } t > 1
\end{cases} \quad \text{with } \rho = \frac{2}{n + 1}.
\]  

(2.2)

Figure 2.1 presents the close price and its 10-day SMA and 10-day EMA chart of S&P 500 index from
04/01/2016 to 30/04/2017. It can be observed that the moving averages help filter out noise but have a lagged response. Also, EMA follows the price more closely due to the increased weight it imposes in the more recent prices.

![Figure 2.1: 10-day SMA and 10-day EMA applied to the S&P 500 index close prices.](image)

**Bollinger Bands** Two trading bands are placed two standard deviations above and below a middle band consisted of a moving average of usually 20 time periods. Standard deviation ($\sigma$) is a statistical concept that describes how prices are dispersed around an average value. Two standard deviations ensures 95% of the price data will fall between the two trading bands. Prices are considered to be overbought when they touch the upper band providing a good selling opportunity. Contrarily, they are oversold when they touch the lower band providing a good buying opportunity. Equation (2.3) presents the calculations of the three bands of the Bollinger Bands indicator:

\[
\text{MiddleBand} = \text{SMA}_{20},
\]

\[
\text{UpperBand} = \text{SMA}_{20} + 2 \times \sigma_{20},
\]

\[
\text{LowerBand} = \text{SMA}_{20} - 2 \times \sigma_{20}.
\]

**Parabolic Stop and Reversal (PSAR)** Is a trend-following indicator that aims to find potential reversals in the market price direction, points in time when the price has a higher-than-normal probability of switching directions.
PSAR is placed above or below an asset’s price on a chart and is used to generate transaction signals depending on where PSAR is placed relative to the price itself. If PSAR is placed below the price, is deemed to be a bullish signal, expecting the prices to remain in the upward direction. Contrarily, if PSAR is placed above the price, is deemed to be a bearish signal, expecting the prices to remain in the downward direction. Once a downward trend reverses and starts to go up, PSAR is placed below the price and will rise, first slowly and then picking up speed and accelerating with the trend but never decreasing. When the price stops rising and reverses below PSAR, a reversal happened and a downward trend starts with PSAR now placed above the price. PSAR will fall, never rising, as it follows the price till it stops to fall and crosses the PSAR happening a reversal. The rising PSAR and falling PSAR are independent and are calculated in different ways shown in the equations 2.4:

\[
\text{RisingPSAR}_t = \text{RisingPSAR}_{t-1} + AF_{t-1} \ast (EP_{t-1} + \text{RisingPSAR}_{t-1}), \quad (2.4a)
\]

\[
\text{FallingPSAR}_t = \text{FallingPSAR}_{t-1} - AF_{t-1} \ast (\text{FallingPSAR}_{t-1} - EP_{t-1}). \quad (2.4b)
\]

\(EP\) refers to the extreme point, highest value reached by the price during the current upward trend, or the lowest value during a downward trend. During each period, if a new maximum (or minimum) is observed, the \(EP\) is updated with that value. The \(AF\) value represents the acceleration factor, usually set initially to a value of 0.02 and increased by 0.02 each time a new \(EP\) is recorded.

**Average True Range (ATR)**  This indicator provides an indication of the degree of price volatility. Volatility is the degree of variation of a trading price series over time. If the price series fluctuates wildly and moves unpredictably it is considered highly volatile and inherently riskier, while a price series relatively stable has low volatility and inherently lower risk. ATR is a 14 time period moving average of the true range values. True range (TR) is the largest of the most recent period’s high minus the most recent period’s low, the absolute value of the most recent period’s high minus the previous close price and the absolute value of the most recent period’s low minus the previous close price. Equation (2.5) presents the formula of the TR:

\[
TR = \max [(\text{high} - \text{low}), \abs{\text{high} - \text{close}_{\text{prev}}}, \abs{\text{low} - \text{close}_{\text{prev}}}] . \quad (2.5)
\]

The first ATR value is the average period TR values values for the last 14 periods. At the other moments in time, the ATR indicator is calculated by:

\[
\text{ATR}_t = \frac{TR_{t-1} \ast 13 + TR_t}{14} . \quad (2.6)
\]

The ATR value can be used to measure the enthusiasm of a market movement. If its value is big its because there is a strong movement (up or down).

**Moving Average Convergence Divergence (MACD)**  This indicator is a momentum oscillator that can alert the trader of market extreme conditions, as overbought or oversold conditions. It is a combina-
tion of some of the oscillator principles with a dual exponential moving average crossover approach. The MACD line is the difference between two EMA’s of closing prices, usually 12 and 26 time periods. The Signal line is usually a 9-time period EMA of the MACD line. Finally, the difference between the former lines allows us to obtain a MACD histogram that can be easily analyzed and offer perspectives on price evolution. Equation (2.7) presents the equations of the MACD line, Signal line and MACD histogram are given by:

\[
MACD \text{ Line} = EMA_{12} - EMA_{26}, \tag{2.7a}
\]

\[
Signal \text{ Line} = EMA_{9}(MACD \text{ Line}). \tag{2.7b}
\]

\[
MACD \text{ Histogram} = MACD \text{ Line} - Signal \text{ Line}. \tag{2.7c}
\]

The application rules of the MACD indicator are:

1. The actual buy and sell signals are given when the lines cross and MACD histogram is zero. When the histogram is positive and starts to fall crossing the zero line, a sell signal is generated. Contrarily, if the histogram is negative and starts to move upward crossing the zero line, a buy signal is generated.

2. In the sense of the first rule, the MACD indicator resembles a dual crossover method. However, the MACD values also fluctuate above and below a zero line. That’s where it begins to resemble an oscillator. An overbought condition is present when the lines are too far above the zero line. An oversold condition is present when the lines are too far below the zero line.

3. Divergences appear between the trend of the MACD lines and the price line, suggesting that the current trend is ending. A negative divergence exists when the MACD lines are on overbought conditions (well above the zero line) and start to weaken while prices continue to trend higher, reaching most of the times a market top. A positive divergence exists when the MACD lines are on oversold conditions (well below the zero line) and start to move up ahead of the price line, reaching most of the times a market low.

Figure 2.2 presents the close price and MACD’s three components chart of S&P 500 index from 01/12/2016 to 30/11/2017.

**Percentage Price Oscillator (PPO)** Is a momentum oscillator that measures the difference between a long-term and a short-term moving average as a percentage of the long-term moving average. PPO is positive when the shorter moving average is above the longer moving average reflecting an upside momentum and negative when the shorter moving average is below the longer moving average reflecting a downside momentum. Equation (2.8) presents the formula of the PPO indicator:

\[
PPO = \frac{EMA_{12} - EMA_{26}}{EMA_{26}} \times 100. \tag{2.8}
\]

**Relative Strength Index (RSI)** Is a momentum indicator that compares the magnitude of recent gains and losses over a specified time period to measure speed and change of price movements. Aims
Figure 2.2: MACD applied to the S&P 500 index close prices.

to identify overbought and oversold conditions in the market. Equation (2.9) presents the RSI indicator formula:

\[ RSI = 100 - \frac{100}{1 + RS} \]  \hspace{1cm} (2.9a)

\[ RS = \frac{Average \ gain \ of \ up \ periods \ during \ x \ time \ periods}{Average \ loss \ of \ down \ periods \ during \ x \ time \ periods} \]  \hspace{1cm} (2.9b)

The values of the indicator range between 0 and 100 and usually it is calculated for an interval of 14 time periods. Movements above 70 identifies an overbought condition and therefore a good selling opportunity. An oversold condition is identified by a move under 30 and therefore a good buying opportunity.

Figure 2.3 presents the close price and RSI line chart of S&P 500 index from 04/01/2016 to 30/11/2017. It is also plotted two dotted lines representing overbought and oversold lines.

**Average Directional Index (ADX)** Is designed to quantify trend strength by measuring the amount of price movement in a single direction.

First, it must be determined the positive and negative directional movement, \( +DM \) and \( -DM \) respectively. Their values are found by calculating the “Upmove”, or the difference between the current high and the previous high, and the “Downmove”, or the difference between the previous low minus the
Equation (2.10) presents the calculations of the directional movements:

\[
\text{Upmove} = \text{High} - \text{PreviousHigh}, \quad \text{(2.10a)}
\]

\[
\text{Downmove} = \text{PreviousLow} - \text{Low}, \quad \text{(2.10b)}
\]

\[
+DM = \begin{cases} 
\text{Upmove} & \text{if Upmove} > \text{Downmove} \text{ and Upmove} > 0 \\
0 & \text{otherwise}
\end{cases}, \quad \text{(2.10c)}
\]

\[
-DM = \begin{cases} 
\text{Downmove} & \text{if Downmove} > \text{Upmove} \text{ and Downmove} > 0 \\
0 & \text{otherwise}
\end{cases}. \quad \text{(2.10d)}
\]

The market direction is determined by the levels of the positive directional indicator, \(+DI\), and the negative directional indicator, \(-DI\), during a \(n\) time period. If \(+DI\) is the higher number comparing to \(-DI\), market direction is up, otherwise, if \(-DI\) is the higher number, market direction is down. Equation (2.11) presents the calculations of the directional indicators with ATR being the indicator:

\[
+DI = 100 \times EMA_n(+DM/ATR), \quad \text{(2.11a)}
\]
Finally, the ADX indicator ranges in value from 0 to 100. If the value is over 20 it indicates the existence of a trend and if it is over 40 indicates a strong trend. Equation (2.12) presents the formula of the ADX indicator:

\[
ADX = 100 \times EMA_n \left( \frac{\left| (+DI) - (-DI) \right|}{(+DI) + (-DI)} \right).
\]  

Commodity Channel Index (CCI) This indicator is used to identify price reversals, price extremes and trend strength. It is calculated as the difference between the actual price and a moving average of the price over a given period, usually 20 time periods, divided by the mean deviation. Equation (2.13) presents the CCI indicator formula where \( TP \) is the typical price and \( \sigma \) is the mean deviation of the prices over the last 20 time periods:

\[
Typical\ Price\ (TP) = \frac{High + Low + Close}{3},
\]

\[
CCI = \frac{1}{0.015} \frac{TP - SMA_{20}(TP)}{\sigma_{TP}}.
\]

The CCI typically oscillates above and below a zero line. Normal oscillations will occur within the range of +100 and -100. Readings above +100 imply an overbought condition, while readings below -100 imply an oversold condition.

Momentum Measures the velocity of price changes. It is the difference between the current time period closing price and the close price \( n \) periods of time ago. The momentum indicator formula is given by the equation (2.14), where \( Close \) is the current time period close price and \( Close_n \) is the close price \( n \) periods of time ago:

\[
Momentum = Close - Close_n -
\]

Rate of Change (ROC) To measure the rate of change, the ROC indicator is a ratio constructed of the most recent closing price to a price a certain number of time periods in the past. The ROC indicator formula is given by equation (2.15) where \( Close \) is the current time period close price and \( Close_n \) is the close price \( n \) periods of time ago:

\[
ROC = \frac{Close - Close_n}{Close_n} \times 100 -
\]

Stochastics (K%D) Is based on the observation that as prices increase, closing prices tend to be closer to the upper end of the price range. Conversely, in downtrends, the closing price tend to be near the lower end of the range. The intent is to determine where the most recent closing price is in relation to the price range of a chosen time period. Two lines are used, the %K line and the %D line. Equation
(2.16) presents the calculations of the Stochastics $\%D$ and $\%K$ indicators:

$$\%K = \frac{Close - LowestLow}{HighestHigh - LowestLow} \times 100, \quad (2.16a)$$

$$\%D = SMA_3(\%K), \quad (2.16b)$$

$\%K$ line measures, on a percentage basis of 0 to 100, where the closing price is in relation to the total price range for a selected time period, based on the lowest low and highest high during that period. The $\%D$ line is a 3 time period moving average of the $\%K$ line. If the $\%D$ line value is below 20 it indicates that price is near its low for the given time period. When the value is above 80 it indicates that price is near its high for the given time period.

**William’s $\%R$**  This indicator is also an oscillator with the main factor being the presence of divergences in overbought or oversold conditions but can also signal a market reversal. It is given by the current time period close price minus the highest price of the range for a given number of time periods and that difference is divided by the total range for the same period. It ranges from 0 to -100 and indicates overbought conditions from 0 to -20 and oversold conditions from -80 to -100. Equation 2.17 presents the William’s $\%R$ indicator formula over a period of time:

$$\%R = \frac{Close - HighestHigh}{HighestHigh - LowestLow} \times 100. \quad (2.17)$$

**On-Balance Volume (OBV)**  Is a momentum indicator that uses volume flow to predict changes in close price. Act as a confirmation tool for price trends, and when OBV and close price are moving in opposite directions, it indicates that a price trend reversal could happen soon. Equation (2.18) presents the calculations of the OBV indicator where $Volume_t$ is the volume value at time $t$:

$$OBV_t = \begin{cases} 
OBV_{t-1} + Volume_t & \text{if } Close_t > Close_{t-1} \\
OBV_{t-1} - Volume_t & \text{if } Close_t < Close_{t-1} \\
OBV_{t-1} & \text{if } Close_t = Close_{t-1} 
\end{cases} \quad (2.18)$$

Different characteristics can be observed from this indicator. If the OBV and the prices are in an upward trend, it can be seen as a confirmation of the trend. The same can be concluded if OBV and the prices are in a downward trend. If the OBV is increasing, the prices are expected to increase, even if they are sideway or going down at the time. If OBV is decreasing, prices are expected to decrease even if the prices are not falling at the time.

Figure 2.4 presents the close price, volume and OBV line chart of S&P 500 index from 04/01/2016 to 30/11/2017.

**Money Flow Index (MFI)**  Is a momentum indicator that measures the inflow and outflow of money over a specific period of time. It uses the close price and volume to measure trading pressure. MFI
Figure 2.4: OBV applied to the S&P 500 index.

ranges between 0 and 100 and when the value is above 80 suggests a overbought situation, while a value lower than 20 suggests a oversold situation. Equation (2.19) present the calculations of the MFI indicator:

\[
\text{Typical Price} = \frac{\text{High} + \text{Low} + \text{Close}}{3},
\]

\[
\text{Raw Money Flow} = \text{Typical Price} \times \text{Volume},
\]

\[
\text{Money Flow Ratio} = \frac{14 - \text{time period Positive Money Flow}}{14 - \text{time period Negative Money Flow}},
\]

\[
\text{MFI} = 100 - \frac{100}{1 + \text{Money Flow Ratio}}.
\]

The positive money flow for the previous 14 time periods is the sum of all the raw money flow when the typical price is higher than the previous typical price. The negative money flow for the previous 14 time periods applies the same logic but sums all the raw money flow when the typical price is lower than the previous typical price. When the typical price rises there is a buying pressure and a positive money flow. When the typical price falls there is a selling pressure and a negative money flow.
Chaikin Oscillator is an oscillator that measures the accumulation/distribution line (ADL) using the formula for the MACD. Generally, when this indicator is positive there is a strong buying pressure and when the indicator is negative there is a stronger selling pressure. Equations (2.20) present the calculations of the Chaikin Oscillator:

\[
\text{Money Flow Multiplier} = \frac{(\text{Close} - \text{Low}) - (\text{High} - \text{Close})}{\text{High} - \text{Low}},
\]

\[
\text{Money Flow Volume} = \text{Money Flow Multiplier} \times \text{Volume},
\]

\[
\text{ADL}_t = \text{ADL}_{t-1} + \text{Money Flow Volume},
\]

\[
\text{Chaikin Oscillator} = \text{EMA}_3(\text{ADL}_t) - \text{EMA}_{10}(\text{ADL}_t).
\]

2.4 Reinforcement Learning

Applying machine learning to financial markets is about learning from data and making predictions and/or decisions. It was previously mentioned that financial markets are very noisy and nonlinear but in spite of these characteristics, financial data are among the best application domains for intelligent processing because data have been recorded very accurately for very long periods of time, scales and different markets providing a very rich environment for analysis and experimentation using advanced processing techniques [12]. Machine learning algorithms are capable of processing a big amount of past financial data suggesting that it can provide a competitive advantage over human inspection of the data, detecting patterns and predicting future outcomes of the market.

Such as human investors, machine learning methods for financial trading can also use technical and fundamental analysis. In fact most of these methods are used to extract features from the raw financial data because they are not as noisy and may enhance the learning of market patterns and trends, that can facilitate the prediction of future outcomes. Features are the properties on which the analysis or prediction is to be done.

The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience. Machine Learning can be roughly split into three major subfields considering the different types of experiences that form the basis to learn a task. They are Supervised Learning, Unsupervised Learning and Reinforcement Learning (RL). In Supervised Learning the experiences contain both the input of the task, as well as the desired output. In some occasions it's not practical to give a desired output, for this reason only the input is given in Unsupervised Learning that aims to discover a hidden structure or patterns inside the input. Reinforcement Learning places between those two fields, since its experiences contain an input and rewards which form an indirect way for acknowledging what the desired outputs are.

In RL approach, the learner and decision-maker is called the agent. The agent has to perform a task, without being told what to do. It interacts with the environment in the form of actions and in each
interaction will cause the environment to change from a current state into a new state. For each of these transitions the agent will receive a reward which indicate the quality of the transition. So, the transitions and reward functions are outside of the agents control, all it can influence is its actions. Its task is to learn a policy, that defines the learning agent’s way of behaving at a given time, for choosing actions that achieve its goals. The agent must perform a sequence of actions, observe their consequences, and learn the policy. The desired policy is one that, from any initial state, chooses actions that maximize the long-term cumulative reward. At the beginning the agent does not know which action to make regarding a current state, but has to explore by trial-and-error in order to understand which actions provides the highest reward over time.

In practical applications, the successes of RL have been extensively demonstrated in a number of tasks, including robotics [13], helicopter control [14] and many other [15].

### 2.4.1 Definition of the Problem

A RL task that satisfies the Markov property is called a Markov decision process (MDP). The Markov property will be explained afterwards in Section 2.4.2 but in order to introduce the RL problem it will be assumed that all RL problems can be formulated as MDPs. A MDP is characterized by the following sets and functions:

- a set of states $S$,
- a set of actions $A(s)$, that represents the actions available to the agent given the state $s \in S$,
- a transition model where $P(s' \mid s, a) = P(S_{t+1} = s' \mid S_t = s, A_t = a)$ is the transition probability between two states $s, s' \in S$ given an action $a \in A(s)$,
- a reward function where $r(s, a, s') = E[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$ is the reward that the agent expects to receive for performing an action $a$ in the state $s$ that leads to the state $s'$.

At each discrete time step $t = 0, 1, 2, 3, \ldots$, the agent receives information about the environment in the form of a state $s_t \in S$. Given this state and its previous experience, the agent selects an action $a_t \in A(s_t)$. Upon executing this action, the agent finds itself in a new state $s_{t+1} \in S$ and receives some reward $r_{t+1}$. Figure 2.5 shows the general framework for the RL process as described, with the agent-environment interaction.

The agent follows a policy $\pi(s)$ which is, in general, a mapping from states to probabilities of executing each possible action. The policy is defined over the entire state-space and will decide which action the agent should take being in state $s$. The policy can be stochastic, presented in Equation (2.21), or deterministic, presented in Equation (2.22):

$$\pi(a \mid s) = P(A_t = a \mid S_t = s),$$  \hspace{1cm} (2.21)

$$a = \pi(s).$$  \hspace{1cm} (2.22)
Experience is defined as the sequence of observations, actions and rewards. As a result of its experience, the agent will change its policy. The ultimate goal of RL is to find an optimal policy $\pi^*$ which maximizes the long-term accumulated reward. This means, maximizing not immediate reward, but cumulative reward in the long run.

2.4.2 State

State is the information used to determine what happens next. It is a signal that contains the available information about the environment to the agent. In other words, it’s the way the agent senses the environment.

If the state does not contain enough information, then the agent will be unable to make a fully informed decision and use all of the knowledge it has previously acquired about its current situation. However, the state should not be expected to inform the agent of everything about the environment, or even everything that would be useful to it in making decisions. Ideally, a state signal should include immediate sensations and summarize past sensations compactly in a way that all relevant information is retained. A state signal that is capable of retaining all relevant information is said to have the Markov property.

**Definition 1.** *(The Markov Property).* Given any sequence of actions $a_t$, state signals $s_t$, and reward signals $r_t$, RL problem satisfies the Markov property if and only if Equation (2.23) is truth for all possible values of $s'$ and $r$.

$$P(S_{t+1} = s', R_{t+1} = r | S_t, A_t, R_{t-1}, ..., R_1, S_0, A_0) = P(S_{t+1} = s', R_{t+1} = r | S_t, A_t) \quad (2.23)$$

This means that a current state satisfies the Markov property if the knowledge of any previous states does not provide any additional information about the future states to the actual state.

Markov states provide the best possible basis for choosing actions. However, a true Markov state is not easily achieved. Nonetheless, it is often valuable to consider them as Markov states because it is still possible to apply RL algorithms to them and obtain meaningful results [16, 17]. For this reason, will be assumed in this work that all the states are Markov states. The states that are not true Markov states are in fact approximations to a Markov state. As a state approaches the Markov property it becomes a
better basis for better performance choosing the actions.

Regarding this work, true Markov states are not feasible. Financial markets are influenced by way too many factors and there is huge amounts of historical data from the past making it impossible to represent a market environment state capable of retaining all the relevant information from previous history. The main goal for the representation of the state signal will be the construction of a state signal that is an approximation to a Markov state.

### 2.4.3 Goals and Rewards

Reward is a signal from the environment to the agent, normally represented as a simple number. The use of a reward signal to formalize the idea of a goal might at first appear limited but in reality it has proved to be flexible and widely applicable.

One of the RL challenges is designing a reward signal that makes sure to accurately reflect the aims of the overall goal, rather than various sub-goals. It is critical that the reward truly indicates what the agent must achieve and not how it is supposed to achieve. The Reward Hypothesis states that all goals can be described by the maximization of expected cumulative reward [18].

In [18], Sutton and Barto give an example of the challenge to design a reward signal for an agent in a Chess game having the intention of winning games. As described earlier, an agent seeks to maximize the long-term cumulative rewards. If a “+1” reward was assigned for each opponent pieces captured, the agent might focus on capturing the other opponent pieces, achieving subgoals, rather than aiming to win the game, the overall goal. Assigning credit for these subgoals, there is a chance the agent might find a way to achieve them without achieving the real goal. This is a mistake, as the reward is reflecting on “how to do it” instead on “what to do”. A reasonable way of defining the rewards would be simple assign a “+1” for a win, “-1” for a loss and “0” for a tie, a “what to do” reward signal.

It is unusual for an agent to receive accurate information about the reward directly after performing each action. Most of the times a future cumulative reward is delayed. Going back to the Chess game example, it is hard to estimate how a single move will affect the overall game state. In that case, an agent may not receive any reward information until the end of the game. When the reward is received, it must be propagated to all the previous states and actions that implicitly caused this reward. As the delay between the action and when the rewards are received grows, credit assignment becomes more difficult. This RL limitation is known as temporal credit assignment problem.

In financial trading, the price difference between two points in time (might be minutes, hours, days...) can be used as a instantaneous reward. So, it is possible to assign credit between any state transitions. However, another problem arises because most of these instantaneous rewards are quite noisy and may not provide reliable supervision for the model training.
2.4.4 Returns

The sequence of rewards an agent expects to receive from a given time period $t$ to some terminal period $T$ is defined as the return $G_t$:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T.$$  \hspace{1cm} (2.24)

For every time period $t$, the agent seeks to maximize the return $G_t$.

However, there are some agent-environment interactions in which there isn’t a final time step. In this case $T = \infty$, and the return which the agent will try to maximize could diverge to infinity very easily, providing no value whatsoever to the agent. In order to solve this problem, it is introduced the concept of discounting. This concept makes use of a parameter known as discount factor ($\gamma$), represented as a real number between 0 and 1. Using this concept the return can now be formulated as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$ \hspace{1cm} (2.25)

Using the discount factor, a reward received $k$ time steps in the future is worth only $\gamma^{k-1}$ times what it would be worth if it were received immediately. As long as $\gamma < 1$ and the rewards are bounded, i.e. $| R_k | < \infty : \forall k > t$, the return sum will converge to a finite value solving the divergence problem.

A discount factor equal to 0 means that at time step $t$, the agent only considers immediate reward, any future reward is worth nothing. The agent’s goal is the maximization of immediate rewards. As $\gamma$ gets close to 1, values of future rewards and immediate rewards become more equally important. In this way, the agent is able maximize long-term rewards.

2.4.5 Value Functions

Most RL algorithms use Value functions to evaluate and improve policies. A Value function is a prediction of future reward used to evaluate the goodness/badness of states and therefore improving policies selecting actions. There are two type of value functions: State-Value and Action-Value functions.

The State-Value function for policy $\pi$. Aims to estimate how good it is for the agent to be in a given state.

**Definition 2.** (State-Value function). Given a state $s$ at the time step $t$, the value of the state under a fixed policy $\pi$ is given by the expected return starting in the given state $s$ and then following the policy $\pi$:

$$V_\pi (s) = \mathbb{E}_\pi [G_t \mid S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right].$$ \hspace{1cm} (2.26)

The Action-Value function for policy $\pi$. Aims to estimate how good it is for the agent to perform a given action in a given state.
Definition 3. (Action-Value function). Given a state $s$, action $a$ at time step $t$, the value of the state-action pair under a fixed policy $\pi$ given action $a$ is given by the expected return starting in the given state $s$, taking the given action $a$ and then following the policy $\pi$:

$$Q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right].$$  \hspace{1cm} (2.27)

A fundamental property of the value functions is that they satisfy important recursive relationships for any policy $\pi$ and any state $s$:

$$V_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$

$$= E_\pi \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s \right]$$

$$= E_\pi \left[ R_{t+1} + \gamma E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s' \right] \mid S_t = s \right]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) [r(s, a, s') + \gamma V_\pi(s')] ,$$  \hspace{1cm} (2.28)

$$Q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

$$= E_\pi \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a \right]$$

$$= E_\pi \left[ R_{t+1} + \gamma E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s' \right] \mid S_t = s, A_t = a \right]$$

$$= E_\pi \left[ R_{t+1} + \gamma V_\pi(s') \mid S_t = s, A_t = a \right]$$

$$= \sum_{s'} P(s' \mid s, a) [r(s, a, s') + \gamma V_\pi(s')] .$$  \hspace{1cm} (2.29)

The equations (2.28) and (2.29) are known as the Bellman equations for the respective value functions. The Bellman equations are very important because they form the basis to approximate and learn the value functions, using $\sum_{s'} P(s' \mid s, a) [r(s, a, s') + \gamma V_\pi(s')]$, as it is going to be shown afterwards.

**2.4.6 Optimal Value Functions and Optimal Policies**

Now that the value functions were defined, it’s easy to understand that the policy the agent wants to learn is based on choosing the action that it expects to lead to the state with the higher state-value, this is, with the highest expected return, in order to maximize the long-term accumulated reward. The agent should prefer state $s_1$ over state $s_2$ whenever $V_\pi(s_1) > V_\pi(s_2)$, because the expected long-term accumulated reward will be greater from $s_1$.

There will be some policies better than or equal to others in a way that the expected return is greater than or equal to all the other policies for all the states. This implies that exists a policy $\pi^*$, denoted as optimal policy, that is better than or equal to all the other policies, assigning to each state, or state-action...
pair, the largest expected return achievable by any policy. So, the agent's learning task is to learn the optimal policy. The optimal policy maximizes \( V_\pi(s) \) for all states \( s \):

\[
\pi^* = \max_\pi V_\pi(s), \ (\forall s).
\]  

(2.30)

In this case the state-value function is said to be the *optimal state-value function*, \( V^* \):

\[
V^*(s) = \max_\pi V_\pi(s), \ (\forall s)
= \max_a \sum_{s'} P(s' | s, a) [r(s, a, s') + \gamma V^*(s')],
\]  

(2.31)

However, the agent must choose among actions, not among states. The optimal action in state \( s \) is the action that maximizes the sum of the immediate reward plus the discounted optimal state-value function of the immediate successor state. So, the optimal policy can be defined as:

\[
\pi^*(s) = \max_a [r(s, a, s') + \gamma V^*(s')].
\]  

(2.32)

Unfortunately, learning \( V^* \) is a useful way to learn the optimal policy only when the agent is able to perfectly predict the reward and the next state for every possible state-action transition, as it can be seen in Equation (2.31). However, comparing Equation (2.31) with Equation (2.29) it is possible to notice that \( Q_\pi(s, a) \) is exactly the quantity that is maximized:

\[
V^*(s) = \max_a Q_\pi(s, a).
\]  

(2.33)

Therefore, it is important to define the optimal policy in terms of \( Q_\pi(s, a) \):

\[
\pi^*(s) = \max_a Q_\pi(s, a).
\]  

(2.34)

In this case the action-value function is said to be the *optimal action-value function*, \( Q^* \):

\[
Q^*(s, a) = \max_\pi Q_\pi(s, a), \ (\forall s)(\forall a)
= \sum_{s'} P(s' | s, a) [r(s, a, s') + \gamma \max_{a'} Q_\pi(s', a')].
\]  

(2.35)

Equation (2.35) means that the optimal state-action value for a pair of state \( s \) and action \( a \) is the sum of the expected reward for executing the action and the discounted best possible value from the subsequent state.

Finally, it is possible to conclude that the policy that the agent aims to learn is given by:

\[
\pi^*(s) = \max_a Q^*(s, a).
\]  

(2.36)
2.4.7 Model

A model is the agent's representation of the environment. It predicts what the environment will do next. $P$ predicts the next state and $R$ predicts the next reward.

\[ P_{s,a}^{s'} = P[S_{t+1} = s' | S_t = s, A_t = a], \]  
\[ R_a^s = E[R_{t+1} | S_t = s, A_t = a]. \] (2.37) (2.38)

2.4.8 Learning Methods

The goal of the learning process is to find the optimal policy $\pi^*$ that maximizes the expected return. Explicitly solving the Bellman equations is a way to find an optimal policy. An alternative method is to search directly in the space of policy representations to find a good policy, without estimating values for particular states or state-action pairs. This approach is called *Direct Policy Search*, commonly referred as *Direct Reinforcement Learning*, and will not be addressed in this work.

*Temporal-Difference Learning* (TD learning) are learning methods that don’t require the agent to have a complete and accurate model of the environment’s dynamics. So, these type of methods do not require that $P(s' | s, a)$ and $r(s, a, s')$ are known for all $s$, $s'$ and $a$. They can learn directly from raw experience and update estimates based in part on other learned estimates without waiting for a final outcome – *bootstrap*. Also, rather than having to wait for a terminal state before updating the value function, TD methods can update the value function at the very next time step, in a step-by-step sense. This can be critical for applications with large episodes because TD methods do not need to wait until the end of the episode to change the value estimates and policies. Another critical advantage of TD methods is that they can be used for non-episodic tasks. Some tasks might not be even divisible into separate episodes.

Regarding this work, TD methods are suitable for non-episodic and very long tasks such as financial trading and don’t require the agent to have a complete knowledge of a financial trading environment. It is impossible for the agent to predict the exact outcome of applying an arbitrary action to an arbitrary state in a financial trading environment. In other words, its impossible for the agent to know all the next-state probability distributions and the rewards in a financial trading environment.

2.4.9 Q-Learning

In RL, there is a distinction between *behavioral policy* which is used by the agent to interact with its environment in order to collect experience, and the *target policy* which is the policy that we want to learn about. Algorithms in which the agent learns about a target policy, which is different from the behavioral policy, are called *off-policy* algorithms. In contrast, in *on-policy* methods, the target policy learned is the policy followed by the agent (behavioral policy).

*Q-learning* is an off-policy TD algorithm in which the target policy is the optimal action-value function $Q^*$. The algorithm learns the action-value function, denoted as $Q$-function in this algorithm, by iteratively
approximate the Q-function at each state-action pair using the Bellman equation, and this iterative update will converge to the optimal Q-function independently of the policy being followed. The update rule that is used in every step of the Q-learning algorithm is given by:

\[
Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha \left[ r(s, a, s') + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a) \right].
\] (2.39)

The motivation for this rule is to calculate the temporal difference between the predicted Q-value given by \([r + \gamma \max_{a'} Q_t] \) and its current value \(Q_t(s, a)\) – bootstrap. This problem can be solved by an iterative collection of transitions and using the update rule for every transition. The learning rate, \(\alpha\), is a parameter that determines how quickly the existing value will be adjusted regarding the significance of new information in relation to the existing value, with \(\alpha \in [0, 1]\).

The environment, or at least parts of it, are assumed to be unknown. The policy still has an effect in that it determines which state-action pairs are visited and updated. All that is required for correct convergence of the algorithm is that all state-action pairs continue to be updated. As long as all transitions are experienced sufficiently often, the bad experiences are eventually all ignored. So, the agent needs to explore the environment in order to update all the state-action pairs. However, a RL limitation is raised because the agent constantly faces the choice between exploiting its current knowledge of the environment or exploring parts it has not experienced enough. After some exploration the agent might have found a set of apparently rewarding actions but it cannot be sure that the actions found were actually the best available. This limitation is known as the Exploration-Exploitation Dilemma.

Most of the strategies of behavioral policies to explore the environment are employed for the naive agent starting with exploration and its exploration-drive gradually diminished over time, turning it more towards exploitation. The most used strategy is the \(\epsilon\)-greedy policy that selects a random action with a probability of \(\epsilon \in [0, 1]\) at each time step, and otherwise the action according to the current belief about the optimal policy (target policy). For example, the agent can start with \(\epsilon = 1\) (full exploration), always selecting a random action, then, progressively with each episode, \(\epsilon\) is decreased linearly towards a final value \(\epsilon_f\), reaching it after a selected number of episodes.

The \(\epsilon\)-greedy policy is given by:

\[
\text{action} = \begin{cases} 
\text{random action} & \text{with probability } \epsilon \\
\text{action according to } \pi(s) & \text{with probability } 1 - \epsilon
\end{cases}.
\] (2.40)

Algorithm 1 presents the pseudo-code of the Q-learning algorithm using a \(\epsilon\)-greedy policy.
Algorithm 1: Q-learning algorithm with $\epsilon$-greedy policy

1. Initialize $Q(s, a)$ arbitrarily, $\forall s \in S, a \in A(s)$, and $Q(\text{terminal state}, .) = 0$;

2. repeat for each episode:
   
   3. Observe current state $s$
   
   repeat for each step of episode:
   
   4. $\text{action} = \begin{cases} 
   \text{random action} & \text{with probability } \epsilon \\
   \text{action according to } \pi(s) & \text{with probability } 1 - \epsilon 
   \end{cases}$
   
   Take action $a$, observe $r$ and $s'$
   
   6. $Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha \left[ R_{t+1} + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a) \right]$
   
   7. $s = s'$
   
   until $S$ is terminal;

3. until convergence or a certain amount of episodes is reached;

Once either all Q-values have converged or a certain amount of episodes is reached, the Q-learning algorithm terminates. The proof of the Q-learning convergence is demonstrated in [19].

Q-learning has many widely applications, including financial markets. Neuneier [20] used a Q-learning approach to make asset allocation decisions. Neuneier and Mihatsch [21] incorporated a notion of risk sensitivity into the construction of Q-function. A portfolio management system using Q-learning where absolute profit and relative risk-adjusted profit were considered as performance functions to train a system [22]. An approach incorporating multiple agents, allowing them to effectively divide and conquer the stock trading problem was proposed in [23].

However, in 2001 Moody et al. [24, 25] proposed an adaptive algorithm called Recurrent Reinforcement Learning (RRL) for discovering investment policies using Direct Reinforcement Learning. This approach that learns the action policy directly differed from dynamic programming and RL algorithms such as Q-Learning which attempt to estimate a value function for the control problem. All approaches had their limitations but Moody [26] observed that relative to Q-Learning, RRL enables a simpler problem representation and offers compelling advantages in efficiency, concluding that RRL produced better trading strategies than systems utilizing Q-learning. This conclusion profoundly changed the view of the scientific community in applying RL to the financial field. Since then, Q-Learning has been almost obsolete being the approaches mainly based in RRL [27].

2.4.10 Illustrative Example

In order to exemplify all the previously introduced concepts, an illustrative example is depicted. The focus is a simple grid-world environment that is presented in Figure 2.6.

Each of the squares represents a state, or location, for the agent and each of the arrows represents...
a possible action that the agent can take to move from a state to another. The number associated with each arrow is the immediate reward the agent receives if it executes the corresponding state-action transaction. The goal state is labeled as $G$, because the only reward the agent can receive is in transitions that lead into the state $G$. Once the agent enters the state $G$, the only action available to the agent is to remain in this state.

Once the states, actions, and immediate rewards are defined, and choosing a discount factor $\gamma = 0.9$, it is possible to determine the optimal policy $\pi^*$ and both value functions.

Figure 2.7 depicts a possible optimal policy for this simple grid-world environment. The policy specifies exactly one action that the agent will select in any given state, depicted as an arrow, and will direct the agent towards the goal state $G$. In this environment there is other possibilities of optimal policies.

Figure 2.8 shows the values of $V^*$ for each state. The goal state $G$ has value 0 because once the agent is in that state, it will remain in that state without receiving any further rewards. The state in the "top middle" has the value 100 because the optimal policy in this state selects the "move right" action that receives immediate reward 100. The "top left" state has value 90 because the optimal policy selects the "move right" action without receiving any immediate reward, and further another "move right" that
receives an immediate reward of 100, as it was seen previously. So, the discounted cumulative reward from the “top left” is given by:

$$V^*_{\text{top left}} = 0 + \gamma \times 100 = 0.9 \times 100 = 90.$$  

The optimal state value-function for the other state are given in the same way, by example:

$$V^*_{\text{middle left}} = 0 + \gamma \times 0 + \gamma^2 \times 100 = 0.9^2 \times 100 = 81,$$

$$V^*_{\text{bottom left}} = 0 + \gamma \times 0 + \gamma^2 \times 0 + \gamma^3 \times 100 = 0.9^3 \times 100 = 72.9.$$

Figure 2.9 shows the values of $Q(s,a)$ for each state-action pair. The state-action pair that keeps the agent within the goal state has value 0 because the agent will remain in that state without receiving any reward for the action and for any other further actions. The action that takes the agent from the state in the “top middle” to the goal state has the value 100 because the agent receives immediate reward 100 but will not receive any further reward from any further actions. The action that takes the agent from “top
left” to the “top middle” state has value 90 because the agent will not receive any reward for the action but the optimal policy will select the “move right” action in the next state that receives an immediate reward 100 that will be discounted. So, the discounted cumulative reward from the “top left” is given by:

\[ Q(\text{top left}, \text{move right}) = 0 + \gamma \times 100 = 0.9 \times 100 = 90. \]

The action-value functions for other state-action pairs are given in the same way, by example:

\[ Q(\text{middle left}, \text{move up}) = 0 + \gamma \times 0 + \gamma^2 \times 100 = 0.9^2 \times 100 = 81, \]
\[ Q(\text{bottom left}, \text{move up}) = 0 + \gamma \times 0 + \gamma^2 \times 0 + \gamma^3 \times 100 = 0.9^3 \times 100 = 72.9. \]

It is also important to illustrate an operation of the Q-learning algorithm, represented in Figure 2.10. The agent is in the “top left” state and will take the action “move right”, receiving an immediate reward with value 0 for this transition.

\[
\begin{align*}
Q(\text{top left}, \text{move right}) &= Q(\text{top left}, \text{move right}) + \alpha \left[ R_{t+1} + \gamma \max_{a'} Q(s', a') - Q(\text{top left}, \text{move right}) \right] \\
&= 73 + 0.1 \left[ 0 + 0.9 \times \max(66, 81, 100) - 73 \right] = 74.7.
\end{align*}
\]

Then, in order to approximate the estimated \( Q(s, a) \) to the optimal action-value function \( Q^* \), the algorithm will update the value estimated using the update rule given in the Equation (2.39). Considering the learning rate \( \alpha = 0.1 \), the update is given by:

\[
Q(\text{top left}, \text{move right}) = Q(\text{top left}, \text{move right}) + \alpha \left[ R_{t+1} + \gamma \max_{a'} Q(s', a') - Q(\text{top left}, \text{move right}) \right] \\
= 73 + 0.1 \left[ 0 + 0.9 \times \max(66, 81, 100) - 73 \right] = 74.7.
\]

Each time the agent moves forward from an old state to a new one, an estimate of the Q-function will be updated and eventually given a sufficient number of updates in all the state-action pairs the Q-function will converge to the optimal state-value function \( Q^* \) shown in the Figure 2.9.

### 2.4.11 Value Function Approximation

For finite MDPs the Q-values of the Q-learning algorithm are defined over every state-action pair. In problems in which the state and action spaces are small enough, the Q-function is represented in a look-up table with one entry for each current estimate of the Q-value for a given state-action pair.
However, in many practical cases, there are far more states than could possible be entries in a table. This is because the memory available is an important constraint. When the complexity of the state-space grows, the memory required to represent the table also increases because of the growing number of state-action pairs. Eventually, it is not possible to represent the Q-function because of the excessive amount of memory needed. This RL limitation is known as the curse of dimensionality.

RL algorithms such as Q-learning can only operate in discrete state-action spaces. The state-action space of a financial market is continuous, which makes the Q-learning algorithm not usable at all. To tackle the limitations encountered, the technique of function approximation is used. In general, function approximation is an instance of supervised learning that aims to generalize from examples of a function to construct an approximate of the entire function. In the context of Q-learning, the function that should be approximated is the Q-function. This way, one can deal with a continuous or high-dimensional state-space by generalizing the Q-function based on the available samples. One of the most common function approximation technique used is called artificial neural networks, because they are good nonlinear function approximators, being a natural approach to consider with modeling time series which have nonlinear dependence on inputs [28]. The integration of RL and NNs is termed Q-network and it has been used as a common approach [29, 30].

2.5 Artificial Neural Network

The concept of artificial neural network (ANN), as the name suggests, is a mathematical model used to emulate biological neural networks of the human brain. The term neural network (NN) is therefore used to stand for ANN in the remainder of this work.

The human brain is an incredibly impressive information processor, being capable of processing a wide variety of data and also capable of pattern recognition, perception and motor control. One important capability of the brain is that it is able to learn, memorize and generalize. Several of these tasks can be done simultaneously and faster by the brain than a computer, allowing to speculate that the human brain must have highly parallel processes operating on representations that are distributed over many neurons [19]. With the motivation to capture this kind of highly parallel computations, the NN was developed as an effort to imitate the human brain.

To this end, a NN contains a set of interconnected artificial neurons, termed neurons throughout this work, that models the biological neurons. Given an input signal into the input neurons, a chain of signals goes through the whole neural system, culminating in an output signal from the output neurons. Fundamentally, NNs are really good function approximators that receiving an input signal, performs a series of operations, and produces an output signal, that corresponds to a value estimation of a complex function. It suits financial environment because not only are capable of generalizing but also to be robust against noisy or missing data [31].
2.5.1 Neural Network Topology

A neural network topology refers to the way in which neurons are connected to form a network. In other words, the neural network topology is the relationship between the neurons by means of their connections. It is an important factor in the neural network functionality and performance.

Most neural networks, including many biological ones, have a layered topology. At the framework level, neurons are considered as abstract entities, thereby not considering possible differences between them. The framework of a neural network can therefore be described by the number of neuron layers and the number of neurons in each of the layers.

The neurons can be distinguished in three types: input neurons, output neurons and hidden neurons. Input neurons receive external inputs from outside the network. Output neurons produce some of the outputs of the network. Hidden neurons have no direct interaction with the “outside world”, only with other neurons. A similar terminology can be used at the layer level for multilayer neural networks. An input layer is consisted of input neurons. An hidden layer is consisted of hidden neurons and an output layer is consisted of output neurons. The reason for the existence of hidden layers is to increase complexity and representational power to the NN.

The number of neurons within a layer and the number of layers in a neural networks varies from one model to another and there are even some models that adapt their topology dynamically during the training process, NeuroEvolution of Augmenting Topologies (NEAT), varying the number of layers and the number of neurons within each layer [32].

The interconnection structure of a neural network determines the way in which the neurons are linked. There are several kinds of connections. An interlayer connection connects neurons in adjacent layers. A supralayer connection connects neurons that are in distinct layers that are not adjacent, this is, separated at least by one hidden layer. Each connection has a weight associated that reflects its importance. This weight is a scalar value, which can be positive (excitatory) or negative (inhibitory). If a connection has a zero weight, it is considered to be nonexistent at that point in time. A neural network is said to be fully-connected if neurons at a layer \( n \) take all outputs of the neurons at layer \( n - 1 \) as input and sends his outputs to all neurons in layer \( n + 1 \).

Feedforward and recurrent are two types of NNs. Feedforward neural networks (FFNN) have a forward flow of information from the input to the output layer. The connections between neurons do not form feedback loops. A FFNN does not depend on past input, it responds only to its present input. Recurrent neural networks (RNN) are networks in which there is at least one feedback loop. They have the ability to retain memory, which enables them to learn temporal characteristics of the data, making it suitable to use with data that has a time dimension, like financial time series. However, a problem with recurrent neural networks is that they tend to focus on the most recent data, thus lowering their ability to learn temporal structures. For this reason, RNN is most commonly used when the inputs exhibit a strong temporal correlation, like in speech recognition.

An example of a simple fully-connected FFNN with three layers is represented in Figure 2.11. Each circle is a neuron, and the arrows are connections between neurons in consecutive layers (interlayer connections). The first layer (input layer) contains input neurons which send information to the second
layer of neurons. The second layer (hidden layer), consisted of hidden neurons, sends information to the third layer (output layer) that contains output neurons.

### 2.5.2 Neurons

A neuron receives one or more signals as input and transforms them to produce an output. Usually, this transformation is a weighted sum of all inputs processed by a non-linear activation function. Each neuron of the NN can have a different activation function. Figure 2.12 presents the mathematical model of a neuron.

\[ y = f(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \]

Figure 2.12: Mathematical model of a neuron.

The main purpose of an activation function is to introduce non-linearity into the neural network in order to allow the network to capture the non-linear relationship between input and response variable. Without non-linearity, even with an infinite number of layers, the neural network would behave like a single layer, because the sum of linear functions is another linear function as well, being limited in power and not performing as desired most of the times. Also another important feature of an activation function is that it should be differentiable in order to compute gradients as we are going to see afterwards.
Section 2.5.3.

There are many possible and widely used activation functions. In its simplest form, this function is binary, either the neuron is firing or not. The most common activation functions used are the sigmoid function, the hyperbolic tangent function (TanH) and rectified linear unit (ReLU). Figure 2.13 presents these activation functions and their respective formulas.

![Activation Functions](image)

**Figure 2.13: Activation functions.**

In fact, one of the recent advances in neural networks is to use the ReLU activation functions. It has been observed empirically that this activation allows for better approximation quality [33]. At the present, the most popular non-linear function is the ReLU [34].

Neurons in the input layer are normally passive. Each neuron represents a dimension of the input, feeding to whichever neurons it has a connection to without transforming them through an activation function. Neurons in the hidden layers are invisible from the “outside world” as they have no direct interaction with it. Their inputs come from other neurons in the network, are transformed, and their output goes to other neurons in the network. Neurons in the output layer also transform the input they receive and their activation function should be chosen in accordance with the desired kind of output the network wants to produce.

### 2.5.3 Training

When using a NN to approximate a function, the data is forwarded through the network layer-by-layer until it reaches the final layer (forward pass). The final layer’s activation’s are the predictions that the network actually makes. The key to make the right decisions is finding the right set of weights for all the connections in order to make the right decisions.

During this process, known as training, it’s often convenient to have some metric of how good or bad the process is doing. This metric is known as the loss or cost function ($L$) and looks at the function the network has inferred and uses it to estimate values for data points. The discrepancies between the outputs in the estimations and the data points are the principle values for the loss function that measures the network performance with respect to accuracy. So, the goal of the training process is to get the loss function value as low as possible. Generally speaking, the loss function should be more or less convex.

Denoting the predicted output estimations as a vector $p$ and the actual output as vector $a$, a usual loss function is the mean square error (MSE) and it can be defined as the “error” of the neural network:
The loss function $L(W)$ is parameterized by the network's weights, any changes in the weights will also change the loss function. The number of weights in a set $W$ can be gigantic, so solving this optimization problem analytically is unfeasible [35]. *Gradient descent* is an iterative optimization algorithm capable of solving the problem by progressively work the weights towards the optimal solution. Recalling that the loss function will be essentially convex and that the goal is to get as close as possible to the global minimum, gradient descent follows the derivatives to essentially “roll” down the slope until it finds its way to the center. The gradient descent update rule for a set of weights $W$ is given by:

$$W = W - \alpha \frac{\partial L}{\partial W}.$$  

Every time there is an update, the weight is subtracted by the derivative of the loss function with regard to the weight itself, scaled by a *learning rate* $\alpha$. The learning rate is one of the most important *hyperparameters* in a NN because if it is too high, it could jump too far in other direction and never get to the searched minimum, contrarily, if it is too low the network will take much time to find the right weights, or worse, get stuck in a local minimum. Repeating this process for every gradient of the loss function with regard to each of the weights, and updating enough each of the weights accordingly will get closer and closer to the center, the derivative terms gets smaller and smaller, converging to zero as it approaches the solution.

There are lots of other iterative optimization algorithms that are commonly used with neural networks like *Adam* or *RMSProp* (for Root Mean Square Propagation). The basic principle remains the same but regardless of which optimization algorithm is used there is still the need to be able to compute the gradient of the loss function with regard to each weight. This is a complicated procedure because the loss function is a many-dimensional function, and worst, the output of a NN is a composite function of the weights, inputs and activation functions. Figure 2.14 presents a NN with functional dependencies.

Figure 2.14: Neural Network for functional dependencies example.

Considering the NN represented in Figure 2.14, if $w_1$ is changed, both *Hidden1, Hidden2* and *Hidden3* outputs would also change:

$$output = f(w_4 \ast f(w_3 \ast f(w_2 \ast f(w_1 \ast input)))).$$  

(2.44)
Backpropagation Introduced by Werbos [36] is an algorithm capable of solving the problem previously introduced. The key insight is that the derivative (or gradient) with respect to a weight can be computed by working backwards from the gradient with respect to the output. It can be applied repeatedly to propagate gradients through all the layers. Once the gradients have been computed, it is straightforward to compute the gradients with respect to the weights.

It was seen that the NN has functional dependencies. Taking the derivative of the function with respect to a weight consists in the application of the chain rule. For example, the application for $w_1$ of the NN represented in Figure 2.14 would be:

$$\frac{\partial \text{Error}}{\partial w_1} = \frac{\partial \text{Error}}{\partial \text{output}} \cdot \frac{\partial \text{output}}{\partial \text{hidden}_3} \cdot \frac{\partial \text{hidden}_3}{\partial \text{hidden}_2} \cdot \frac{\partial \text{hidden}_2}{\partial \text{hidden}_1} \cdot \frac{\partial \text{hidden}_1}{\partial w_1}.$$ (2.45)

Considering now a more complex NN represented in Figure 2.15. It will be considered that the activation function for all the neurons is the sigmoid function. For matters of simplification, the notation used is a generalization. This is, if a variable has the subscript $i$, it means that the variable is the input to any arbitrary neuron and if it has the subscript $j$, it means it is the output of any arbitrary neuron. For example, $x_i$ refers to any input value entering the network and not a exact neuron in the layer $x$. $W^{layer}_{ij}$ refers to any arbitrary, single weight at a given layer, connecting arbitrary neuron $i$ at a given layer to an arbitrary neuron $j$ at the next layer.

\[\text{Figure 2.15: Neural Network for backpropagation example.}\]

The derivative of the sigmoid function is given by:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x} \left( \frac{e^{-x}}{1 + e^{-x}} \right)^{-1} = e^{-x} \left( \frac{1 + e^{-x}}{e^{-x}} \right)^{-2} = \frac{e^{-x}}{(1 + e^{-a})^2} = \frac{1 + e^{-x} - 1}{(1 + e^{-a})^2}$$ (2.46)

To acquire a more comprehensive intuition of backpropagation, the derivative of the error with regard to any arbitrary weight between the input layer and the first hidden layer $W_{ij}^{1}$ will be solved. This equation is given by:

$$\frac{\partial \text{Error}}{\partial W_{ij}^{1}} = \frac{\partial L}{\partial W_{ij}^{1}} = \frac{\partial L}{\partial p_j} \frac{\partial p_j}{\partial W_{ij}^{1}}.$$ (2.47)

This means that the derivative of the error with regard to the weight can be written as the derivative of the error with regard to the output prediction multiplied by the derivative of the output prediction with
regard to the weight. The derivative of the error with regard to the output prediction is given by Equation (2.42). As a consequence, Equation (2.47) is now given by:

\[
\frac{\partial L}{\partial W_{ij}} = (p_j - \vec{a}) \frac{\partial p_j}{\partial W_{ij}}.
\] (2.48)

Iterating one layer backwards:

\[
\frac{\partial p_j}{\partial W_{ij}} = \frac{\partial p_j}{\partial p_i} \frac{\partial p_i}{\partial W_{ij}}.
\] (2.49)

This means that the derivative of the output prediction with regard to the weight is the derivative of the output with regard to the input of the output layer multiplied by the derivative of that value with regard to the weight. The first term is given by the derivative of the activation function that was already solved in Equation (2.46):

\[
\frac{\partial p_j}{\partial p_i} = p_j(1 - p_j).
\] (2.50)

Consequently:

\[
\frac{\partial p_j}{\partial W_{ij}} = p_j(1 - p_j) \frac{\partial p_i}{\partial W_{ij}}.
\] (2.51)

\[
\frac{\partial L}{\partial W_{ij}} = (p_j - \vec{a}) \cdot p_j(1 - p_j) \frac{\partial p_i}{\partial W_{ij}}.
\] (2.52)

There are multiple different weights that contribute to the value of \(p_i\) and that must be taken into account in the derivative:

\[
\frac{\partial p_i}{\partial W_{ij}} = \sum_j \frac{\partial p_i}{\partial z_j} \frac{\partial z_j}{\partial W_{ij}}.
\] (2.53)

Since \(p_i\) is the weighted sum of each \(z_j\), the result of the derivative of \(p_i\) with any arbitrary \(z_j\) is the connecting weight.

\[
\frac{\partial p_i}{\partial z_j} = W_{ij}^3,
\] (2.54)

\[
\frac{\partial p_i}{\partial W_{ij}} = \sum_j W_{ij}^3 \frac{\partial z_j}{\partial W_{ij}}.
\] (2.55)

From this point on the logic is the same. Since it’s still not possible to put the derivative of \(z_j\) with regard to \(W_{ij}^3\) into a numerical term, the chain rule is applied:

\[
\frac{\partial z_j}{\partial W_{ij}} = \frac{\partial z_j}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} = z_j(1 - z_j) \frac{\partial z_i}{\partial W_{ij}}.
\] (2.56)

The derivative of \(z_i\) with regard to \(W_{ij}^3\) follows the same reasoning from equation 2.53 and 2.54 and the derivative of \(y_j\) with regard to \(W_{ij}^3\) from equation 2.51.

\[
\frac{\partial z_i}{\partial W_{ij}} = \sum_j \frac{\partial z_i}{\partial y_j} \frac{\partial y_j}{\partial W_{ij}^3} = \sum_j W_{ij}^3 \frac{\partial y_j}{\partial W_{ij}^3}.
\] (2.57)

\[
\frac{\partial y_j}{\partial W_{ij}^3} = \frac{\partial y_j}{\partial y_i} \frac{\partial y_i}{\partial W_{ij}^3} = y_j(1 - y_j) \frac{\partial y_i}{\partial W_{ij}^3}.
\] (2.58)

For the first time it’s possible to directly deriving with regard to the weight \(W_{ij}^3\). Opposing what was
done previously, now the coefficient of $W^1_{ij}$ will be considered as being $x_j$ in the weighted sum. Of course, since each $x_j$ in the layer $x$ contributes to the weighted sum of $y_i$, the value must be summed:

$$\frac{\partial y_i}{\partial W^1_{ij}} = \sum_j x_j .$$

(2.59)

Finally, with no more partial derivative terms left, backpropagation work is complete. The final expression for the derivative of the error with regard to any weight in $W_1$ is given by:

$$\frac{\partial L}{\partial W^1_{ij}} = (p_j - \bar{a}) \ast p_j (1 - p_j) \ast \sum_j \left[ W^3_{ij} \ast z_j (1 - z_j) \ast \sum_j \left[ W^2_{ij} \ast y_j (1 - y_j) \ast \sum_j x_j \right]\right] .$$

(2.60)

**Overview**

Now that the most important NN training principles were previously introduced, it is important to overview the training procedure:

1. First, a NN is created with a proposed topology and initializing all the weights with random values.

2. Then, one forward pass is performed using the training data, calculating values of the network at different points, including the output predictions.

3. After this, backpropagation is performed to get the error derivatives with regard to each and every weight in the neural network.

4. Finally, gradient descent is performed to update each weight by the negative scalar reduction of the respective error derivative (gradient descent update rule), using the derivatives calculated by the backpropagation algorithm.

5. This process is repeated a numerous of times, starting again in step 2, until a defined termination condition is met.

A NN algorithm is composed of a dataset that is divided into non-overlapping training, validation, and testing subsets. The training process already described, in which the goal of the process is to make the error on the training dataset small — training error. After training, the performance of the model is measured on a different set of examples — validation and testing datasets. This process tests the generalization ability of the NN, measuring the ability to produce sensible answers on new inputs that it has never seen during training — generalization error or test error. The validation dataset is used to provide an evaluation of a model test error while tuning the model during training. However, the validation test error may fluctuate during this process. Finally, the test dataset is used to provide an evaluation of a final model fit on the training dataset.

A NN algorithm not only tries to make the training error small but also the gap between training and test error. If a model cannot achieve a low training error it is said to be underfitting. A model is overfitting if the model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not
apply to new data and negatively impact the models ability to generalize. If there is overfitting, the gap between training error and test error is large.

2.6 Q-network

At this point, RL and NNs were introduced and it was also explained the importance of combining a NN with RL for Q-function approximation, termed Q-network.

Q-network parameterizes an approximate value function $Q(s, a; W_i)$ in which $W_i$ are the weights of the Q-network at iteration $i$. The input of this network is a state representation and the output is the Q-value of a possible action for that state. The Q-network can be trained by adjusting the weights $W_i$ at iteration $i$ to minimize the MSE of the Q prediction error (the difference between the left and right side of the Bellman equation, presented in Equation (2.61)):

$$Q^*(s, a) = r_{t+1} + \gamma \max_{a'} Q^*(s', a').$$  \hspace{1cm} (2.61)

In other words, the Q prediction error can be formulated as the difference between the predicted $Q(s, a; W_i)$ (left side) and the optimal target values (right side). The optimal target values ($y$) are substituted with approximate target values:

$$y = r + \gamma \max_{a'} Q(s', a'; W_i).$$ \hspace{1cm} (2.62)

Consequently, the loss function of the Q-network is given by:

$$L(W_i) = \mathbb{E}_{(s,a,r,s')} \left[ (y - Q(s, a; W_i))^2 \right].$$ \hspace{1cm} (2.63)

However, in practice RL is known to be unstable or even diverge when a non-linear function approximator such as an NN is used to represent the action-value [37]. This instability has several causes: the correlations present in the sequence of observations; the fact that small updates to Q may significantly change the policy and therefore change the data distribution; and, the correlations between the action-values $Q(s, a; W_i)$ and the optimal target value $y = r + \gamma \max_{a'} Q(s', a'; W_i)$, because both of them are predicted with the same set of weights $W_i$, at iteration $i$, constantly shifting in a correlated manner, which makes learning more difficult and it may also spiral out of control through feedback loops.

2.7 Feature Learning

For decades, conventional machine learning techniques were limited in their ability to process natural data in their raw form. Constructing a pattern recognition or machine learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data into a suitable internal representation or features from which the learning system could detect or classify patterns in the inputs without overfitting training samples and generalizing poorly to new
samples. But this can all be avoided if good features can be learned automatically using a general-purpose learning procedure.

*Feature Learning* or *Representation learning* is a set of methods that allows a machine to be fed with raw data and to automatically discover representations that preserve as much information about the original data as possible, and at the same time to keep the representation simpler or more accessible than the original data, with low-dimensional, sparse, and independent representations. This replaces manual feature engineering, because the features do not require prior knowledge about the learning task at hand, and allows a machine to both learn the features and use them to perform a specific task.

### 2.7.1 Deep Learning

In contrast to *shallow learning* in which machine learning algorithms have an input layer, an output layer, a single hidden layer in the middle, and the inputs may be transformed with manual feature engineering before training, deep learning allows computational models composed with more than one hidden layers between input and output layers to learn representations of data with multiple levels of abstraction [34]. It discovers intricate structures in large datasets by using the backpropagation algorithm to indicate how a model should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.

Deep learning methods are feature learning methods with multiple levels of representation, obtained by composing simple but non-linear functions. Each of them transform the representation at one level, starting with the raw input, into a representation at a higher, slightly more abstract level. A NN with more than one hidden layer is termed a *deep neural network* (DNN) and it is considered a deep learning method. In fact, most of the NNs previously presented in this work are DNNs. However, models may not necessarily be more effective when they become more complicated because of overfitting.

Deep learning methods have dramatically improved the state-of-the-art in many domains like image recognition [38, 39], speech recognition [40, 41], natural language understanding [42], and many more, including financial investments [43, 44]. However, deep learning existed for a long time but only recently began to attract attention. These recent successes of deep learning can be attributed to benefiting from big data, powerful computation, new algorithmic techniques, strong financial support and mature software packages and architectures [34].

### 2.7.2 Principal Component Analysis

The input data for many machine learning problems, including RL, is often very high-dimensional. Therefore, it is in most cases not feasible to apply learning algorithms directly on this input data as the amount of samples that is required to the model generalize accurately increases exponentially with the amount of data dimensions (curse of dimensionality). This leads to excessive computation times and overfitting.

*Principal Component Analysis* (PCA) is a dimensionality reduction algorithm that can be used to significantly speed up feature learning. It focus on reducing the dimensionality of a dataset consisting
of a large number of interrelated variables, while retaining as much as possible of the variation present in the dataset. This is achieved by transforming the data into a new set of variables, the principal components, which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables.

It uses an orthogonal linear transformation to transform the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (first principal component), the second greatest variance on the second coordinate (second principal component), and so on. The obtained principal components are a linear combination of the original features that reflect as much as possible the original information.

The number of principal components is equal to the original data’s number of dimensions but the goal of PCA is to represent the data with only some of the principal components, the ones that have the biggest original data variance, constructing a low representation of the data, with most of the original data variance retained. For example, if the original data contains 20 dimensions and the first 5 principal components can explain 95% of the original data variance, then PCA can be used to reduce the dimensionality of the data by retaining only the first 5 principal components.

Mathematically, the principal components are the eigenvectors $W$ of the covariance matrix $\text{Cov}(X)$ of the original multidimensional data $X$ (dimension $n \times n$, where $n$ is the dimension of data $X$). The principal components correspond to the direction (in the original $n$-dimensional space) with the greatest variance in the data. Each eigenvector has a corresponding eigenvalue $\lambda$ that is equal to how much variance there is in the data along that eigenvector. In other words, a larger eigenvalue means that the principal component explains a large amount of the variance in the data. This way, eigenvalues can be used to arrange the principal components in order of significance. Equation (2.64) present PCA eigenproblem:

$$\text{Cov}(X) \ast W = \lambda \ast W.$$ (2.64)

As it was demonstrated by [45], each principal component of the data can be obtained with PCA as presented in Equation (2.65), where $X_i$ is the original variable, $Y_i$ is the principal component, $\alpha_i$ is the coefficient vector and $n$ is the number of original features. The coefficient vector $\alpha_i$ can be estimated by maximizing the variance captured by the corresponding principal component, $Var(Y_i)$, with the constraint that it has to be orthogonal to the preceding principal components, as presented in Equation (2.66). A practical example of the PCA method can also be seen in [45].

$$
\begin{align*}
Y_1 &= \alpha_1^T \cdot \vec{X} = \alpha_{11} \ast X_1 + \alpha_{12} \ast X_2 + \ldots + \alpha_{1n} \ast X_n \\
Y_2 &= \alpha_2^T \cdot \vec{X} = \alpha_{21} \ast X_1 + \alpha_{22} \ast X_2 + \ldots + \alpha_{2n} \ast X_n \\
&\vdots \\
Y_n &= \alpha_n^T \cdot \vec{X} = \alpha_{n1} \ast X_1 + \alpha_{n2} \ast X_2 + \ldots + \alpha_{nn} \ast X_n 
\end{align*}
,$$ (2.65)
Maximize \( Var(Y_i) \)

w.r.t. \( \tilde{\alpha}_i \), \( \tilde{\alpha}_i \), \( \tilde{\alpha}_i \)

subject to \( \tilde{\alpha}_i^T \tilde{\alpha}_i = 1 \)

\( Cov(Y_i, Y_j) = \tilde{\alpha}_i^T Cov(\tilde{X}).\tilde{\alpha}_i, \quad j \in \{1, 2, ..., i - 1\} \)

PCA has been frequently applied to machine learning algorithms. Zhong and Enke [46] concluded that the combination of PCA with NNs gives a higher classification accuracy to forecasting the daily direction of future market return than other two dimensionality reduction algorithms. Curran et. al [47] concluded that by projecting an RL agent’s state onto a low dimensional manifold, the state space can be represented in a smaller and more efficient representation with the algorithm converging to a good policy much faster. Recently, Nadkarni [45] concluded that the PCA method for dimensionality reduction can reduce the number of features while maintaining the essence of the financial data, being crucial to the system performance.

2.8 Deep Q-network

The integration of RL and NNs has a long history [30], although, as it was previously explained, RL is known to be unstable or even divergent when Q-function is approximated with a nonlinear function as NNs [37]. For this reason, the combination of RL and NNs did not gather much enthusiasm for many years. With the recent achievements of deep learning, it has similarly accelerated progress in RL, witnessing a RL renaissance [48], specially, the combination of DNNs and RL, termed deep reinforcement learning (deep RL).

In 2015, a team of researchers at Google DeepMind developed a RL agent called deep Q-network (DQN) that combined Q-learning with a deep NN for Q-function approximation and ignited the field of deep RL [3]. DQN made several contributions including stabilizing the training of Q-function approximation with deep NNs. What makes DQN remarkable is that it showed how a single RL agent can achieve high levels of performance in many different problems without relying on different problem-specific feature sets. To demonstrate this, DQN learned how to play 49 different Atari 2600 video games by interacting with a game emulator, using the same raw input, the same architecture, and the same hyperparameters values. These games varied widely in different effects of actions, different state-transition dynamics, and they needed different policies for earning high scores. The deep NN learned to transform the raw input common to all the games into features specialized for representing the action-values required for playing. DQN achieved levels of play at or beyond human professional players on most of these games.

The ancient Chinese game of Go has challenged AI researchers for many decades. Methods that achieve human-level, or even superhuman-level skill in other games have not been successful in producing strong Go programs. Despite prior methods had only achieved amateur level performance, AlphaGo [49] developed by Google DeepMind using deep RL broke this barrier, defeating an 18-time world champion Go player, Lee Sedol, winning 4 out of 5 games in a challenge match. This success was a break-
through as was the historic achievements of IBM's *Deep Blue* in chess [50] and IBM's *Watson DeepQA* system in *Jeopardy!* [51]. However, unlike the handcrafted rules that have dominated these systems, AlphaGo did not rely on any handcrafted feature.

First, instead of using a separate forward pass to compute the Q-value of each action, resulting in a cost that scales linearly with the number of actions, DQN uses a topology in which the state representation is the input and there is a separate output unit for each possible action for the input state. The outputs correspond to the predicted Q-values of the individual actions. This constitutes an advantage to this type of topology that is the ability to computing the Q-values for all possible actions in a given state with only a single forward pass through the network.

To address the instabilities of Q-function approximation with NN, DQN uses two key ideas: *Experience Replay* and *Separate Target Network*.

**Experience Replay** Is a technique in which the agent's experiences are stored at each time step, \( e_t = (s_t, a_t, r_{t+1}, s_{t+1}) \), in a data set \( D_t = \{e_1, ... e_t\} \) pooled over many episodes into a replay memory. A classic approach to Q-learning is only using an experience once for learning purposes. DQN applies minibatch updates, to samples of experience, \( (s, a, r, s') \sim U(D) \), drawn uniformly at random from the pool of stored samples, re-using these experiences.

Minibatch learning consists in learning more than one training experience at each time step. In this case, it learns from samples drawn from the replay memory. Generally, determining the gradient of a batch involves computing the loss function over each training sample in the batch and then, at the end, summing the results of these functions.

This technique has many advantages. First, learning from random samples breaks the correlations between learning from consecutive samples and therefore reduces the variance of the updates. Second, each experience is potentially used in many weight updates, which allows for greater data efficiency. Third, it makes the learning process less prone to outliers and noise, as the gradient computed at each time step uses more training examples. Fourth, the behaviour distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.

**Separate Target Network** Or only target network \( \tilde{Q} \), is used to compute the optimal target values \( y \). Initially it contains the weights of the network enacting the policy \( Q \), but is kept frozen every \( C \) updates, where \( C \) is a numerical hyperparameter, and only then the target network \( \tilde{Q} \) is updated to match the policy network \( Q \). This way, target network \( \tilde{Q} \) is used to compute the optimal target values \( y \) for the next \( C \) updates to \( Q \).

This approach makes the learning procedure more stable because in the standard procedure if an update increases \( Q(s_t, a_t) \) it often also increases \( Q(s_{t+1}, a) \) for all \( a \) and hence also increases the optimal target values \( y \). This may lead to divergence or oscillations of the policy. If the targets are computed using an older set of parameters, target network, it adds a delay between the time step an update affects the target values \( y \), making divergence or oscillations much more unlikely [3].

Due to the application of these two key ideas, the DQN loss function is different from standard Q-
network loss function. Equation (2.67) presents the DQN loss function where the optimal target values are now computed using the target network set of weights $W^-_i$ at iteration $i$ and $(s, a, r, s') \sim U(D)$ denotes the distribution of the training examples:

$$L(W_i) = E_{(s,a,r,s') \sim U(D)} \left[ \left( r + \gamma \max_{a'} Q(s', a'; W^-_i) - Q(s, a; W_i) \right)^2 \right]. \quad (2.67)$$

The full algorithm for DQN training is presented in Algorithm 2.

**Algorithm 2: DQN learning algorithm**

1. Initialize replay memory $D$ to capacity $N$
2. Initialize action-value function $Q$ with random weights $W$
3. Initialize target action-value function $\tilde{Q}$ with weights $W^- = W$
4. repeat for each episode:
   5. Observe current state $s_1$
   6. repeat for each step $t$ of episode:
      7. \[ action a_t = \begin{cases} \text{random action} \quad & \text{with probability } \varepsilon \\ \arg\max_a Q(s_t, a; W) \quad & \text{with probability } 1 - \varepsilon \end{cases} \]
      8. Execute action $a_t$, observe $r_{t+1}$ and $s_{t+1}$
      9. Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $D$
    10. Sample random minibatch of transactions $(s_i, a_i, r_i, s_{i+1})$ from $D$
    11. \[ y_j = \begin{cases} r_j \quad & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \tilde{Q}(s_{j+1}, a'; W^-) \quad & \text{otherwise} \end{cases} \]
    12. Perform a gradient descent step on $(y_j - Q(s_j, a_j; W))^2$ with respect to the network parameters $W$
    13. Every $C$ steps reset $\tilde{Q} = Q$
    14. $s_t = s_{t+1}$
5. until $S$ is terminal;
6. until convergence;

Since these major works, Google DeepMind has been proposing some improvements.

Double Q-learning [4] was proposed to tackle an overestimate problem in Q-learning. The max operator in Equation (2.67) uses the same values both to select and to evaluate an action from step $j$ at iteration $i$:

$$y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; W_i) = r_j + \gamma \tilde{Q}(s_{j+1}, a'; W^-_i; W_i). \quad (2.68)$$
As a consequence, it is more likely to select overestimated value estimates. To solve this problem, it was proposed to evaluate the greedy policy according to the online policy network, but to use the target network to estimate its value:

\[ y_j = r_j + \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; W_i) ; W_i^-) . \] (2.69)

In DQN, experience transitions are uniformly sampled from the replay memory, regardless of the significance of the experiences. Prioritized Experience Replay [5] proposed that important experience transitions, with high expected learning progress, can be replayed more frequently in order to learn more efficiently. The importance of experience transitions can be measured by the magnitude of their temporal-difference (TD) errors (\( \delta \)):

\[ \delta_j = r_j + \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; W_i) ; W_i^-) - Q(s_j, a_j; W_i) . \] (2.70)

Deep RL algorithms have been successfully applied to a wide range of problems, such as video games [3], robotics [52, 53], indoor navigation [54], and many more, having tremendous positive results. Deep RL has already been used to approach all manner of machine learning tasks and it seems likely that in the future, deep RL will be an important component in constructing general Artificial Intelligence systems [55].

To the author best knowledge there are only two deep RL approaches to trade in financial markets. The first approach combined deep NN with Direct RL [56], concluding that the system was able to select features from raw data due to the DL mechanism and learn effectively. This approach also uses fuzzy learning to summarize the market trend in order to reduce uncertainties in the raw data. The second approach used DQN to trade in the EUR-USD Forex market [6], using feature extraction to extract features from the raw data. However, the system did not consider transaction costs. Both systems verified the potential of deep RL approaches and concluded that this is a promising future direction.

2.9 Relevant Studies

As the application of this thesis is investing in Forex markets, the more relevant background work to the development of this thesis includes studies that apply the presented algorithms to financial markets. In Table 2.1 it is presented the results and applications of some relevant studies applied to financial markets. Note that all these studies have already been introduced and explained.
<table>
<thead>
<tr>
<th>Ref.</th>
<th>Year</th>
<th>Method</th>
<th>Data</th>
<th>Period</th>
<th>Financial Market</th>
<th>Algorithm performance</th>
<th>B&amp;H performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>[27]</td>
<td>2006</td>
<td>Direct RL combined with dynamic optimization</td>
<td>Forex price (daily)</td>
<td>Jan 2000 - Dec 2001</td>
<td>EUR-USD</td>
<td>26% (annualized profit)</td>
<td>-8%</td>
</tr>
<tr>
<td>[23]</td>
<td>2007</td>
<td>Multiagent Approach to Q-Learning</td>
<td>Stock price (daily)</td>
<td>June 2001 - Nov 2005</td>
<td>KOSPI index (Korean stock market)</td>
<td>1138.7 (asset growth rate)</td>
<td>NA</td>
</tr>
<tr>
<td>[56]</td>
<td>2017</td>
<td>Deep Direct RL combined with fuzzy learning</td>
<td>Stock price (minute)</td>
<td>Jan 2014 - Jan 2015</td>
<td>Stock-IF contract (China index-based future contract)</td>
<td>3256.6 (total profit)</td>
<td>739</td>
</tr>
<tr>
<td>[45]</td>
<td>2017</td>
<td>NEAT combined with PCA for dimensionality reduction</td>
<td>Stock / Commodities price (daily)</td>
<td>30/01/2015 - 13/04/2017</td>
<td>S&amp;P 500, Brent, CAC40, Sugar</td>
<td>18.89%, 37.91%, 4.69%, 23.15%</td>
<td>15.71%, -9.94%, 9.82%, 24.56%</td>
</tr>
<tr>
<td>[46]</td>
<td>2017</td>
<td>ANN combined with PCA for dimensionality reduction</td>
<td>Stock price (daily)</td>
<td>30/11/2011 - 31/05/2013</td>
<td>SPY (S&amp;P 500 index ETF)</td>
<td>37.1% (without transaction costs)</td>
<td>30.8%</td>
</tr>
</tbody>
</table>

Table 2.1: Results of some studies relevant for this thesis.
Chapter 3

Implementation

This chapter presents the implemented system in order to maximize financial gains trading in Forex markets. The trading system uses the DQN algorithm combined with recent training methodologies and the PCA technique for timely selecting the best trading actions in order to maximize returns while trading in the markets. The process of trading is well depicted as an online decision making problem, involving two critical steps: market condition summarization and optimal action execution.

Financial environment summarization and representation is a big challenge because financial data is dynamic, noisy, non-stationary, and non-linear. However, this is a critical step to the system's performance. To mitigate data noise and uncertainty, technical analysis is used to extract features from the raw financial data because it may enhance the learning of market patterns and trends, increasing the system's prediction capability. However a widely known drawback of machine learning algorithms is overfitting. Using too many technical indicators, plus the raw financial data, in other words, many features, may lead to this phenomenon. On the other side, if only some technical indicators are used as features, the environment summarization may not retain all the relevant information.

In order to exploit the features into a robust feature representation, the feature learning PCA technique is applied to the many technical indicators and raw financial data features, reducing its dimensionality while explaining the majority of the original data's variance, not only facilitating the system's learning but also avoiding overfitting, and learning the input features of the DQN algorithm automatically.

With the DQN algorithm, abstract and robust market patterns can be extracted from the features. These patterns form a much better representation for the market status than the input features. The DQN algorithm has two major components. The first, a deep learning component, where the DNN learns the market status from the input features, being capable of learning hierarchical patterns and the patterns learned by the high-level layers tend to be abstract and invariant against unexpected disturb, essential in financial trading. The second, a Q-learning component that learns the Q-function in order to predict the best trading action for the market status. However, these two components are integrated as an entire DQN implementation. DQN uses recent improvements to the standard DQN algorithm known as the Double Q-learning and Prioritized Experience Replay.

The second challenge is due to the dynamic behavior of trading action execution. This challenge
is tackled with the DQN component that learns to predict the best trading action for the market status. However, frequently changing the trading positions might contribute nothing to the profits but lead to great losses due to the transaction costs. So, the policy learning part must be modeled in order that the system learns from historic actions and from the holding trading positions.

Finally, one of the main goals of the system’s implementation is creating a system’s framework that can perform good not only in a specific Forex market but also in many and distinct Forex markets with the same network topology, system’s hyperparameters values and raw financial data parameters.

First, in this chapter, the overall system and its architecture is explained and then a detailed description of each of its components and methods is given.

The system was developed in Python programming language.

3.1 System’s Architecture

The system’s architecture is presented in Figure 3.1 and is composed by three major layers: user layer, business logic layer and data layer. Defining a multilayer system architecture with clear separation between layers and modules is a key requirement for any system because it provides flexibility to extend and change the code of modules without affecting the others modules. Each module is described in detail in the next sections.

The system is composed of four major components: data preprocessing, deep transformation, Q-learning, and trading. These parts, play the roles of reducing data uncertainty and noise (feature extraction), feature learning, trading policy making (RL) and trading execution, respectively.

The execution of the implemented system can be summarized by the following steps:
1. The user starts by defining the configuration data of the system. May choose which financial market to trade (input to the financial data module), which technical indicators to use (input to the technical analysis module), and if PCA is used (input to the PCA module). Regarding DQN the user may choose its hyperparameters values, if the Double Q-learning is used, and if the Prioritized Experience Replay is used (input to the DQN module). Finally, it may choose the trading parameters (input to the trading module).

2. After the user has defined the configurations of the system, the system loads the financial data (financial data module) and it starts by calculating the predefined technical indicators on the raw financial data (technical analysis module). The result is a dataset containing both raw financial data and the defined technical indicators.

3. Then, all this data is fed to the normalization module where it is normalized.

4. The normalized data is then fed to the PCA module where it can be applied the PCA technique reducing the dimensionality of the dataset. The data preprocessing is concluded in this step and the final dataset consists in the features, learned automatically from the normalized data, that are going to be fed as input to the DQN algorithm (DQN module).

5. DQN algorithm is applied, learning deep representations of the input features and outputting trading predictions (DQN module).

6. Based on the trading predictions of the DQN algorithm, the trading module executes a trading action, investing in the market and calculating the returns. The returns are then fed to the DQN module so that the DQN algorithm can use it to learn from it.

7. At the end of the system’s execution, the results of DQN and trading module are saved so that the user can analyze them.

### 3.2 Financial Data

The financial data module is responsible for the acquisition and storage of the historical raw financial data of a specified market by the user.

The raw financial data is composed by different fields depending on its frequency and source. Usually it is composed by the date and time of the record (Date). Regarding the trading frequency, the opening price (Open), the closing price (Close), the maximum price (High), the lowest price (Low) and the Volume of shares, contracts or currency units traded. In some financial markets there is another field known as adjusted close (Adj. Close) that represents a more accurate reflection of a stock's value because stock splits, reverse stock splits and new stock offerings can change the closing price.

The data is provided to the system in a comma-separated values (.csv) format. Figure 3.2 presents a .csv format file with S&P500 historical data downloaded from Yahoo Finance.
Figure 3.2: S&P500 historical data on .csv format file.

The data is acquired from the .csv format file and stored in a dataframe using Python data analysis library Pandas. Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language that was used for all the operations using dataframes. Finally, the dataframe containing some of the raw financial data components is fed into to the next module, the technical analysis module. Table 3.1 presents the list of raw financial data components that will be fed into the technical analysis module.

Table 3.1: List of variables fed to the technical analysis module.

<table>
<thead>
<tr>
<th>Raw Financial Data</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Adj. Close</td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td></td>
</tr>
</tbody>
</table>

3.3 Technical Analysis

Unlike other types of signals, such as images or speech, financial data contains high amounts of unpredictable uncertainty and noise. Besides, a number of other factors may also affect the direction of the financial signal in real time. These factors can be market sentiment, the psychology of market participants, fundamental factors, and many others. Therefore, reducing the uncertainties and noise in the raw financial data is an important approach to increase the robustness for financial signal mining. The quality of this data affects the system’s learning performance and therefore needs to be preprocessed. If the data is not preprocessed, feature learning is more difficult or it may even lead to misleading results.

Technical analysis has the capability of providing historical information in a compressed way, essential to machine learning problems because of the overfitting phenomenon, and filtering out the noise from
random price fluctuations, making it much easier to understand the underlying trend. These properties may enhance the learning of market patterns and trends facilitating the system’s prediction capability.

The technical analysis module receives the raw financial data and computes a set of 26 technical indicators. Then, these technical indicators are stored in a new dataframe, together with the raw financial data, and fed into the next module, the data normalization module. The reason why this set of 26 technical indicators were used and not another set, was that this precise set of indicators was used by the most important related work on data preprocessing [45]. Its author selected this set because it gathers the most common technical indicators used in the state-of-art of machine learning algorithms that forecast financial markets. All the indicators used were previously explained in Section 2.3.3.

The indicators are computed recurring to TA-Lib Python library. The computed technical indicators set are presented in Table 3.2.

<table>
<thead>
<tr>
<th>Technical Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMA20, SMA50, SMA100</td>
</tr>
<tr>
<td>EMA20, EMA50, EMA100</td>
</tr>
<tr>
<td>Middle, Upper and Lower Bollinger Bands</td>
</tr>
<tr>
<td>PSAR</td>
</tr>
<tr>
<td>ATR</td>
</tr>
<tr>
<td>MACD, Signal Line and MACD Histogram</td>
</tr>
<tr>
<td>PPO</td>
</tr>
<tr>
<td>RSI</td>
</tr>
<tr>
<td>ADX</td>
</tr>
<tr>
<td>CCI</td>
</tr>
<tr>
<td>Momentum</td>
</tr>
<tr>
<td>ROC</td>
</tr>
<tr>
<td>Stochastic %D and %K</td>
</tr>
<tr>
<td>Williams %R</td>
</tr>
<tr>
<td>OBV</td>
</tr>
<tr>
<td>MFI</td>
</tr>
<tr>
<td>Chaikin Oscillator</td>
</tr>
</tbody>
</table>

### 3.4 Data Normalization

The technical indicators and raw financial data range of values varies widely. It results in unnecessarily complex relationships by making the nature of the mapping along some of them much different from others. This difficulty can be circumvented by normalizing each of the variables so that the variance of each variable is equal. In theory, it is not necessary to normalize these features. However, when performing the PCA technique, it is critical to normalize the data. PCA seeks to identify the principal components with the highest variance, if the data are not properly normalized, attributes with large values and large variances in absolute terms will end up dominating the first principal component. It
was also concluded that when the features are normalized, the gradient descent converges much faster, and consequently neural network training is more efficient, which leads to better system’s predictions [57, 58].

The following steps detail the implementation of the data normalization in this system:

1. The data normalization module receives as input the dataframe, containing the set of technical indicators and the raw financial data, and will divide their length into three distinct sets: training set, validation set and testing set. The training set is the set used to train the system – in-sample data. The validation and testing sets are used to test the generalization capability of the system in out-of-sample data.

2. Then, a rescaling method is applied to the training set. This method is the Min-Max normalization that rescales the data values to the range of [0,1]. This method starts by finding in the dataset, in this case, the training set $X$, the maximum $X_{\text{max}}$ and minimum $X_{\text{min}}$ values of the set. Then, it normalizes all the values from the training set using Equation (3.1):

$$\text{Normalized value} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}.$$  
(3.1)

3. After this procedure, the validation and testing datasets are also normalized using the same logic as equation 3.1. However the maximum and minimum values of out-of-sample datasets are unknown. A simple way to overcome this problem is considering the maximum with the same value of the maximum from the training set $X_{\text{max}}$ and following the same logic to the minimum, $X_{\text{min}}$. A problem may arise because if the out-of-sample is above $X_{\text{max}}$ or below $X_{\text{min}}$ the value will be out-of-scale, as it is going to seen in Figure 3.3, because NNs are very sensible to out-of-scale values.

4. Finally, the normalized sets are concatenated and stored into a dataframe that is fed to the next module, the PCA module.

The Min-Max normalization method is computed recurring to the Scikit-learn Python library. Figure 3.3 presents an example of the Min-Max normalization method on fictitious market close prices. Note that the training set maximum and minimum used for the normalization procedure are 20 and 10, respectively, and because of that some out-of-sample values will be normalized out-of-scale.

<table>
<thead>
<tr>
<th>Training close prices</th>
<th>Validation close prices</th>
<th>Testing close prices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw value</td>
<td>14 17 20 17 12 10 11</td>
<td>8 12 15 19 21 22 20 18</td>
</tr>
<tr>
<td>Normalized value</td>
<td>0.4 0.7 1 0.7 0.2 0 0.1</td>
<td>-0.2 0.2 0.5 0.9 1.1 1.2 1 0.8</td>
</tr>
</tbody>
</table>

Figure 3.3: Example of the Min-Max normalization method.
3.5 Principal Component Analysis

PCA module receives as input the normalized extracted features. These features contain 26 technical indicators and 4 types of raw financial data, combining for a total of 30 features. Therefore, the input features have a high dimensionality. Using this many features will provide the learning system more information about the current market status that may enhance the learning of market patterns and trends and consequently increase the system's prediction capability. However, as it was mentioned before, the system will be prone to overfitting, because the amount of samples that is required to generalize accurately increases exponentially with the amount of features.

PCA module is responsible for applying the PCA technique in order to reduce the high dimensionality of the features into a low dimensional features while explaining the majority of the original feature’s variance, facilitating the system’s learning and avoiding overfitting. The following steps detail the application of the PCA technique to the high dimensional received features:

1. PCA technique will start by dividing the normalized features into training, validation and test datasets. Then it will determine the normalized training dataset covariance matrix and its eigenvalues. This allows to compute the orthogonal principal components, the components that represent the directions of maximum variance on the training dataset. This process is known as fitting of the data.

2. After the normalized training dataset is fitted, the resulting data is a set containing the principal components, corresponding to linear combinations of the original features. As it was mentioned in section 2.7.2, the number of principal components is equal to the number of features fitted by the PCA technique, thereby the set will have 30 principal components. The principal components set is ordered by the amount of variance of the original training data they explain. Mathematically, PCA technique orders the eigenvectors (principal components) by their eigenvalue (variance explained).

3. The PCA goal is selecting the principal components that explain the most variance of the original data, easy procedure because they are ordered this way. Normally, the PCA technique chooses the minimum number of principal components that explain a certain percentage of the variance in the original data. This system will select the minimum number of principal components that explain at least 95% of the training dataset variance.

4. Finally, all the normalized datasets are transformed, projecting this sets on the principal components extracted from the normalized training dataset, reducing the dimensionality of the datasets to a dimensionality equal to the minimum number of principal components that explain at least 95% of the set variance. Then, the datasets are concatenated and the final set is fed to the next module, the DQN module, as input features of the learning algorithm. Notice that the input features of the learning algorithm are automatically learned by the PCA technique.

Table 3.3 presents the principal components of 30 features obtained from applying technical indicators plus raw financial data of the EUR-USD data from 12/06/2003 to 05/12/2012. It can be noted that
the minimum number of principal components that explain at least 95% of the dataset variance is 8, with a cumulative variance of 96.28%. Therefore, the original 30 features can be reduced to only 8 features while explaining most of the original features variance. The PCA technique is also computed with the Scikit-learn Python library.

Table 3.3: Principal components of the 30 features obtained from the EUR-USD market data (period of 12/06/2003 to 05/12/2012).

<table>
<thead>
<tr>
<th>Principal Component</th>
<th>Variance (%)</th>
<th>Cumulative Variance (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56.2205210</td>
<td>56.2205210</td>
</tr>
<tr>
<td>2</td>
<td>23.7089141</td>
<td>79.9294351</td>
</tr>
<tr>
<td>3</td>
<td>6.00290207</td>
<td>85.9323372</td>
</tr>
<tr>
<td>4</td>
<td>2.97002620</td>
<td>88.9023624</td>
</tr>
<tr>
<td>5</td>
<td>2.45816506</td>
<td>91.3605284</td>
</tr>
<tr>
<td>6</td>
<td>1.81557018</td>
<td>93.1760986</td>
</tr>
<tr>
<td>7</td>
<td>1.68287496</td>
<td>94.8589736</td>
</tr>
<tr>
<td>8</td>
<td>1.42515932</td>
<td>96.2841329</td>
</tr>
<tr>
<td>9</td>
<td>0.97972957</td>
<td>97.2638625</td>
</tr>
<tr>
<td>10</td>
<td>0.87406864</td>
<td>98.1379311</td>
</tr>
<tr>
<td>11</td>
<td>0.63653385</td>
<td>98.7744650</td>
</tr>
<tr>
<td>12</td>
<td>0.50006014</td>
<td>99.2745251</td>
</tr>
<tr>
<td>13</td>
<td>0.39715884</td>
<td>99.6716839</td>
</tr>
<tr>
<td>14</td>
<td>0.15223916</td>
<td>99.8239231</td>
</tr>
<tr>
<td>15</td>
<td>0.05875497</td>
<td>99.8826781</td>
</tr>
<tr>
<td>16</td>
<td>0.04538224</td>
<td>99.920603</td>
</tr>
<tr>
<td>17</td>
<td>0.03275164</td>
<td>99.9608119</td>
</tr>
<tr>
<td>18</td>
<td>0.02386038</td>
<td>99.9846723</td>
</tr>
<tr>
<td>19</td>
<td>0.00790255</td>
<td>99.9925749</td>
</tr>
<tr>
<td>20</td>
<td>0.00320888</td>
<td>99.9957837</td>
</tr>
<tr>
<td>21</td>
<td>0.00211755</td>
<td>99.9982096</td>
</tr>
<tr>
<td>22</td>
<td>0.00112146</td>
<td>99.9993310</td>
</tr>
<tr>
<td>23</td>
<td>0.00040186</td>
<td>99.9997329</td>
</tr>
<tr>
<td>24</td>
<td>0.00019639</td>
<td>99.9999297</td>
</tr>
<tr>
<td>25</td>
<td>0.00002118</td>
<td>99.9999504</td>
</tr>
<tr>
<td>26</td>
<td>0.00001726</td>
<td>99.9999677</td>
</tr>
<tr>
<td>27</td>
<td>0.00001662</td>
<td>99.9999843</td>
</tr>
<tr>
<td>28</td>
<td>0.00001567</td>
<td>100.0000000</td>
</tr>
<tr>
<td>29</td>
<td>4.405E-32</td>
<td>100.0000000</td>
</tr>
<tr>
<td>30</td>
<td>2.962E-33</td>
<td>100.0000000</td>
</tr>
</tbody>
</table>

3.6 Deep Q-network

The DQN module receives the preprocessed data from the PCA module, applies the DQN algorithm and outputs the trading action signal to the trading module. It receives back the trading execution data and the reward signal from the trading module. This module is the core of the system.
The goal of the DQN agent is to interact with the trading module by selecting actions in a way that maximizes future rewards. The future rewards are discounted by a discount factor ($\gamma$) per time step. The optimal action-value function $Q^*(s, a)$, or Q-function, is defined as the maximum expected return achievable by following any policy, after experiencing a state representation $s$ and then taking some action $a$. It is given by

$$Q^*(s, a) = \max_\pi \mathbb{E}[R_{t+1} | s_t = s, a_t = a, \pi],$$

(3.2)

where $\pi$ is the policy mapping sequences to actions.

The DNN of the DQN system learns abstract and robust market patterns extracted from the preprocessed input financial data features (deep learning), forming a much better representation for the market status than just the input financial data features usually extracted in most of the forecasting financial markets algorithms [56]. At the same time, the DNN will also learn the Q-function $Q(s, a)$ to maximize the long-term accumulated profit in order to predict the best trading action for the market status learned.

To learn the Q-function, DQN agent interacts with the financial environment (trading module) in a sequence of actions, observations and rewards. At each time step the agent makes the predictions about the actions $a_n$, with $n \in [0, 1, 2]$ that represent a neutral, long and short position respectively. The trading predictions are passed to the trading module where an action is executed and the reward is computed. Afterwards, the DQN module receives the reward signal from the trading module and computes the next state with the information received from the trading module and with the features received from the PCA module.

### 3.6.1 Model Architecture

As it was mentioned, for any given input state of the market, the system will only consider neutral, long and short financial trading actions. Therefore, DQN model topology consists in receiving only the state representation as input, that is going to be addressed afterwards, and having three separate output units (neurons) for each possible action. The output units correspond to the predicted Q-values of the individual actions for the input state. The input layer has a number of neurons defined by the elements of the state representation.

Despite many improvements in designing and training a NN, there is no foolproof recipe for deciding how many hidden layers, number of neurons and which activation functions have to be used to achieve good results. Therefore, these decisions have to be made empirically during the design process. In terms of network topology, the first decision was choosing a feedforward topology. As it is going to be presented in Chapter 4, many topologies were tested and the topology that suits best the goal of this thesis is using a fully-connected three hidden layer network with 200 neurons in each of the layers. Finally, all the neurons in the network have the same activation function, the ReLU activation function. This decision was made because ReLU was concluded to have superiority in training DNN over other activation’s functions [59]. To implement the NN, a high-level NN API (Application Program Interface) named Keras was used. It provides public benchmark implementations of different types of NNs and focus on enabling fast experimentation, being able to go from an idea to a result with the least possible
delay. It is mainly known for its easy extensibility, mainly because it is easy to add new modules and works with Python.

### 3.6.2 State

As it was mentioned, the state representation is the available information about the environment that the agent uses to determine what happens next. It must contain enough information for the agent to make a fully informed decision and ideally it should include immediate sensations and summarize past sensations compactly in a way all relevant information is retained.

With this in mind, it is logical to use a state representation based on the input features received from the PCA module. The dataframe received from the PCA module containing the input features has a length equal to the combined length of the datasets (training, validation and test). However, only one feature’s sample is used each time a state representation is constructed. Nonetheless, the dimension of the samples is not an exact value because it depends on the PCA technique. For example, if the length of the combined datasets is equal to 1000 samples because the system will learn and test in 1000 samples, corresponding to 1000 trading time periods, the received dataframe will also have length equal to 1000 features samples. But when constructing the state representation, for example to the third trading time step, only the 3rd features sample of the received dataframe is used to construct the state representation, where the features have an inaccurate number of dimensions. Two different state representations are proposed:

- The first, is solely a features sample, and thereby, from now on it is going to be denominated **features state**. Considering the example presented for the PCA technique where 30 features of the EUR-USD data were reduced to only 8 features, Figure 3.4 presents the features state representation for the 3rd time step \(s_3\) that is equal to the third features sample. In this case the features sample has 8 dimensions and are represented as \(F_{tn}\), with \(t\) representing the time step and \(n\) the dimension number.

<table>
<thead>
<tr>
<th>(s_3)</th>
<th>(F_{31})</th>
<th>(F_{32})</th>
<th>(F_{33})</th>
<th>(F_{34})</th>
<th>(F_{35})</th>
<th>(F_{36})</th>
<th>(F_{37})</th>
<th>(F_{38})</th>
</tr>
</thead>
</table>

Figure 3.4: Example of the features state representation for a third trading time step.

- In addition to the current market condition, the historical actions and the holding trading positions are required to be explicitly modeled in the policy learning part in some applications. Therefore, the second state representation not only uses the features sample but also uses two components that will provide information about the holding trading positions and historic actions. So from now on will be denominated **historical state representation**. The first component is an integer scalar \((A)\) with three possible discrete values to make the system aware if a trading position is currently...
open. Equation (3.3) presents first component possible values at time step $t$:

$$A_t = \begin{cases} 
0 & \text{if short position is open} \\
0.5 & \text{if neutral position} \\
1 & \text{if long position is open}
\end{cases} \quad (3.3)$$

The second component is a float scalar ($P_t$) to inform the system how much unrealized profit percentage the trading position currently open has made, in a normalized manner. In order to normalize the unrealized profit, the system computes the maximum and minimum possible profit percentages in the training dataset, independently of how many time steps are needed to have these profits. Every time the historical state representation is constructed, Min-Max normalization is applied to the unrealized profit percentage, using the maximum and minimum computed, exactly as it was showed in Section 3.4. Equation 3.4 presents how the second component is computed at time periods $t$ before the normalization is applied:

$$P_t = \begin{cases} 
0 & \text{if neutral position} \\
\frac{\text{current price} - \text{position open price}}{\text{position open price}} & \text{if long position} \\
\frac{\text{position open price} - \text{current price}}{\text{position open price}} & \text{if short position is open}
\end{cases} \quad (3.4)$$

Still, considering the example presented for the PCA technique in Section 3.5, Figure 3.5 presents the historical state representation for the third time step ($s_3$) that consists in the two components ($A_3$, $P_3$) plus the third features sample ($F_{3n}$).

$$s_3 = \{ A_3, P_3, F_{31}, F_{32}, F_{33}, F_{34}, F_{35}, F_{36}, F_{37}, F_{38} \}$$

Figure 3.5: Example of the historical state representation for the third trading time step.

It should be noted that naming the state representations is only to facilitate their identification and use in this work.

### 3.6.3 Reward

One of, if not, the biggest and crucial challenge of the DQN algorithm applied to financial markets is designing a reward signal that takes into account the overall goal: financial gains.

The noisy characteristics of the price signal may not provide reliable supervision for the model training. Also, frequently changing the trading positions may contribute nothing to the profits but lead to great losses due to the transaction costs. So, an accurate reward signal should reflect price fluctuations in order to acknowledge profit possibilities in a way that prevents the agent from paying too many transaction costs that may lead to profit losses.
Taking this into account, several reward functions were tested and analyzed. However, in most of them, the algorithm was not capable of learning the Q-functions even with different topologies and hyperparameters values in an attempt to find a way to the algorithm to converge. The conclusion taken for these functions is that they weren’t suited for this purpose. If the system don’t converge it would always choose the long position from the first to the last trading time step, resembling a Buy and Hold (BnH) strategy. The reward functions that were able to learn and converge to good solutions are the ones that are proposed:

- The first proposed reward function is the *Period return* reward function. It consists on using the price difference percentage of a frequency period as reward. Despite some doubts that this instantaneous reward would provide reliable supervision to the model learning, it proved to be consistent with the proposed goal. Equation (3.5) presents how the period return reward is computed, where \( a_t \in \{\text{short, neutral, long}\} = [-1, 0, 1] \) is the action executed and \( p_t \) is the close price at time step \( t \):

\[
R_{t+1} = a_t \times \frac{p_{t+1} - p_t}{p_t}.
\]  

- *Transaction profit* is a reward function that uses the profit percentage obtained from each transaction as reward. However, the system only receives negative rewards when it closes a position with negative unrealized profit percentage. This way, the system would learn to keep a position open until its unrealized profit is positive, as long as it takes. To solve this issue an instantaneous reward component is added to the reward signal. This component is in fact the period return reward signal. Equation (3.6) presents how the transaction profit is calculated:

\[
R_{t+1} = \begin{cases} 
    a_t \times \frac{p_{t+1} - p_t}{p_t} & \text{if position is not closed} \\
    a_t \times \frac{p_{t+1} - p_o}{p_o} & \text{if position is closed}
\end{cases}.
\]  

If a position is open (long or short) and it is not closed, it is computed the same way period return reward signal. If a position is open and it is closed, \( p_o \) is the close price at the specific time step where the position was opened.

It is important to notice the reward value that the agent receives when a position is closed, is the exact same value of the unrealized profit component present in the historical state representation.

- *Cost* reward function is a reward function that takes into account the transaction costs. It simulates that the system agent always flips the trading position every time step. This way, the reward signal will acknowledge the agent if the returns obtained in a time period can overcome its transaction costs. The goal of this reward is yielding a negative value even if the period return is positive but it cannot overcome the transaction cost. It tries to avoid the agent to profit from small market fluctuations and spending a considerable number of days with a open position. Equation (3.7)
presents how cost reward function is computed:

\[ R_{t+1} = \begin{cases} 
  \frac{p_{t+1} - p_t - \text{TransactionCosts}}{p_t} & \text{if long position is open} \\
  0 & \text{if neutral position} \\
  \frac{p_t - p_{t+1} - \text{TransactionCosts}}{p_t} & \text{if short position is open} 
\end{cases} \quad (3.7) \]

The reward signal for all the proposed reward functions is clipped to the \([0, 1]\) range as proposed by [3] that concluded by clipping the rewards, error derivatives are limited. To clip the reward signal, the Min-Max normalization method is applied. In all the reward functions the signal is normalized using the maximum and minimum possible period return percentage in the training set. However, in the transaction profit reward signal, when a position is closed, the reward signal is the unrealized profit percentage value. As it was mentioned before, this is the exact same value that the historical state representation contains. So, it is performed the same procedure for its normalization as it is performed for the unrealized profit percentage value of the historical state representation. The maximum and minimum used are the maximum and minimum possible profit percentage in the training dataset, independently of how many frequency periods are needed to have that profits.

The reward normalization procedure is done exactly as it was shown in Section 3.4. However, if a out-of-sample value is above \(X_{\text{max}}\) or below \(X_{\text{min}}\) then the value is mapped with the high and low value respectively of the normalization range. This approach leads to significant information loss and to a concentration of values on certain parts of the normalized range [60], which implies more computation effort and loss of quality in learning techniques [61, 62], essential to the very important reward function.

Note that, the reward signal is computed in the trading module. However, the reward is so important to the DQN algorithm that it was mentioned in this section.

### 3.6.4 Learning Algorithm

The implemented algorithm is based on the standard DQN algorithm proposed by [3]. However, recent improvements known as Double Q-learning [4] and Prioritized Experience Replay [5] were also implemented.

Like the standard DQN approach, the algorithm uses two separate NNs, an online policy network \(Q\) and a separate target network \(\tilde{Q}\) with the same topology. In order to implement Double Q-learning, the difference from the standard approach is that instead of selecting and evaluating an action with the same network, the greedy policy is evaluated according to the online policy network but it is the target network that is used to estimate its value. This way, as it was mentioned in Section 2.8, the optimal target values, \(y\), are computed using equation:

\[ y_j = r_j + \gamma Q(s_{j+1}, \arg \max_{a'} Q(s_{j+1}, a'; W_1); W_2^{-}) \quad (3.8) \]

The system uses the experience replay technique but, instead of sampling the experience transitions in a uniformly manner from the replay memory \(D\), Prioritized Experience Replay samples experience
transitions with high expected learning progress more frequently. The central component of prioritized replay is the criterion by which the importance of each transition is measured. The ideal criterion would be the amount the DQN agent would learn from a transition in its current state. However, this measure is not accessible. A reasonable criterion is the magnitude of a transition’s TD error ($\delta$), as it was mentioned in Section 2.8:

$$\delta_j = r_j + \gamma Q(s_{j+1}, \text{argmax}_{a'} Q(s_{j+1}, a'; W_i); W_i^r) - Q(s_j, a_j; W_i).$$  \tag{3.9}$$

A way of implementing Prioritized Experience Replay is a greedy TD-error prioritization. This approach stores the TD error along with each transition in the replay memory. Then, it always replays the transitions with the highest absolute TD errors. However, this approach has several issues. A transition with a low TD error on the first time it is sampled may not be replayed for a long time. It also focuses on a small subset of the transitions, meaning that the initially high error transitions get replayed more frequently. This lack of transition diversity makes the system prone to overfitting.

A way to overcome these issues is interpolating between a pure greedy prioritization and uniform random sampling, known as proportional prioritization. The priority of a transition $i$ is given by

$$p_i = (|\delta_i| + \epsilon) \alpha,$$  \tag{3.10}$$

where $\epsilon$ is a small positive constant that guarantees that every transition may be replayed even if its TD error is zero, and $0 \leq \alpha \leq 1$ controls how much prioritization is used. If $\alpha = 1$ a total prioritization is used and $\alpha = 0$ would be the uniform case.

The probability of sampling a transition $i$ is given by

$$P(i) = \frac{p_i}{p_{\text{total}}}$$  \tag{3.11}$$

where $p_i > 0$ is the priority of transition $i$ and $p_{\text{total}}$ is the sum of all the priorities in $D$. This way, during each learning step, a batch will be sampled containing samples with probability distribution $P(i)$.

To sample the transitions, a possible implementation is having an array with $N$ transitions. A random number $s$, $0 \leq s \leq p_{\text{total}}$, is picked and the array is “walked” left to right, summing up a priority of the current element until the sum is exceeded and that element is chosen. This process selects a sample with the desired probability distribution. However, this implementation has a terrible efficiency for sampling – $O(N)$.

DQN applied to financial markets has a number of transitions higher than normal. It would not be feasible to sample from distribution presented in Equation (3.11) if the complexity depends on the number of transitions ($N$). To implement the sampling process minimizing additional run-time and memory overhead, an efficient implementation based on a sum-tree data structure was constructed. The sum-tree is used in a similar way to the array representation of a binary heap. However, instead of the usual heap property, every node is the sum of its children, with the priorities as the leaf nodes. This way, internal nodes are intermediate sums, with the parent node containing the sum of all priorities $p_{\text{total}}$. This provides a efficient way of calculating the cumulative sum of priorities, allowing $O(\log N)$ updates.
and sampling.

To sample a minibatch of size $k$, the range $[0, p_{total}]$ is divided equally into $k$ ranges. Then, a value is uniformly sampled from each range. The algorithm used to find the transition corresponding to the sampled value is presented in Algorithm 3, where each node has three components: its value, a left children (left) and a right children (right).

**Algorithm 3: Prioritized Experience Replay sampling algorithm**

1. Initialize $n$ as the parent node
2. Sample a value $s$
3. While $n$ is not a leaf node do
   4. If $n.left.value \geq s$ then
      5. $n = n.left$
   6. else
      7. $s = s - n.left.value$
      8. $n = n.right$
   9. end
4. Return $n$

Figure 3.6: Prioritized Experience Replay sampling example.

Figure 3.6 presents an example of the Algorithm 3 approach. In this example 8 transitions are stored in the replay memory, corresponding to the leaf nodes where its priority is identified inside the node. The internal nodes are the sum of its children and the parent node has total priority $p_{total} = 44$. The value sampled $s$ is equal to 24 that corresponds to the third leaf node (from left to right) because the sum of the priorities of the leaf nodes from the left to right is exceeded at the third leaf node. A variable $n$ starts at the parent node and compares the value sampled with its left child node value. Because the left node value (29) is bigger than 24, $n$ variable goes to the left child node. But then, its left child value (13) is lower than 24. So, the variable moves to the right child node but first it decreases the sampled value with the left child value (24-13=11). Now, the left children node value is 11 and it is lower than 11, so $n$ moves into the left children. Finally, $n$ is analyzing a node that doesn’t have children, concluding that it is a leaf node storing the transition that is going to be sampled.
The sum tree is initialized with a capacity $N$ and with all leaf nodes empty. Every time a new transition is stored, the first empty leaf node from “left” to “right” is filled and the transition is stored with maximal priority value and its experience $(s, a, r, s')$. The maximal priority value is attributed as $1\%$ higher than the last maximal priority attributed to a transition on the tree. If all the leaf nodes are unfilled then the maximal given priority is $1$. Every time a transition is sampled and used to train the network, the new TD error and its new priority is computed, updating the sum-tree priorities according to the new priority. When the sum tree is completely filled, the oldest transitions will be removed, emptying the leaf node, and the newest transitions will be stored in the empty leaf.

Therefore, with these improvements the loss function of the implemented DQN algorithm is given by

$$L(W_i) = \mathbb{E}_{(s,a,r,s') \sim P(D)} \left[ r + \gamma \max_{a'} Q(s', a'; W_i) - Q(s, a; W_i)^2 \right],$$

where $P(D)$ denotes the distribution of the transitions according to Equation (3.11) and the optimal target values, $y$, are computed using Equation (3.8).

The full implemented algorithm is presented in Algorithm 4.

### Algorithm 4: DQN learning algorithm with Double Q-learning and Prioritized Experience Replay

1. Initialize replay memory $D$ to capacity $N$
2. Initialize action-value function $Q$ with random weights $W$
3. Initialize target action-value function $\tilde{Q}$ with weights $W^- = W$
4. repeat
   5. for each episode:
      6. repeat
         7. Observe current state $s_t$
         8. action $a_t = \begin{cases} \text{random action with probability } \varepsilon, & \text{with probability } 1 - \varepsilon \end{cases}$
         9. Execute action $a_t$, observe $r_{t+1}$ and $s_{t+1}$
         10. Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $D$ with maximal priority $p_t = \max_{i < t} p_i$
         11. if $t \% \text{update\_frequency} = 0$ and $D.\text{size} >= k$ then
         12. Sample a minibatch with size $k$ using Prioritized Experience Replay method
         13. Perform a gradient descent step on $(y_j - Q(s_j, a_j; W))^2$ with respect to the network parameters $W$
         14. Computes TD error: $\delta_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; W_i) - Q(s_j, a_j; W_i)$
         15. Update transition priority $p_j \leftarrow |\delta_j|$
     16. end
   17. Every $C$ steps reset $\tilde{Q} = Q$
5. until $S$ is terminal;
until convergence;

At each time step $t$, for a given state $s_t$, the agent selects and executes actions $a_t$ according to an
ε-greedy policy based on $Q$. Then receives a reward $r_{t+1}$ from the trading module and computes the new state representation $s_{t+1}$. The transition is stored in $D$ with maximal priority. As proposed by [3], gradient descents updates are only performed every update_frequency steps, with update_frequency being a integer scalar. Another condition is having at least $k$ transitions stored in $D$ in order to sample the minibatch. If gradient descent is going to be performed, a minibatch with $k$ size is sampled, dividing $p_{total}$ in $k$ ranges and selecting a transition from every range. Then, for every transition $j$ of the minibatch, the optimal target value $y$ is computed and a gradient descent update on all the minibatch is performed. After the update, the new TD error $\delta$ is computed for every transition $j$ and their priority is updated. Every $C$ steps, the weights $W^{-}$ of the target network $\tilde{Q}$ are updated to the weights $W^{+}$ of the policy network.

At the next time step, the state is computed new state representation of the previous time step. This procedure is performed for every time step till the end of an episode and till convergence or a termination condition is achieved.

### 3.6.5 Training Procedure and Test

During the learning algorithm, DQN agent learns a model. The model is a representation of the financial environment, predicting what happens next in the environment. It predicts the next state and which rewards it receives choosing paths of actions. However, if the learning algorithm is performed many times, it does not exactly mean the model gets better. The reason behind this is the overfit phenomenon. There will be a point where the model is learning and it will start to lose its generalization capability, needed for the out-of-sample test, where the "real" test is performed. To mitigate the disturbances of overfitting, the training procedure is structured into epochs with three distinct phases:

1. Learning algorithm applied on training dataset.
2. Test on the training dataset.
3. Test on the validation dataset.

In each epoch the model learns and then is tested. In each epoch the learning algorithm uses only one episode that is the training dataset. The tests are performed on the training and on the validation datasets, selecting actions according to the current belief about the optimal policy (model $Q$ predictions), without using an ε-greedy policy. Over the epochs, the indicator of how well the system is learning is the profit generated by the agent. The test on the training dataset evaluates the learning on the training data. However, since the goal is to find the model having the best performance on new data, the simplest approach to compare different models is to evaluate the profit using data which is independent of that used for training – validation test. The validation dataset corresponds to the period of time immediately following the training dataset period of time, as presented in Figure 3.3. If the model was only tested on the training dataset, the observed progress may mislead that the model is progressively getting better because the model will start memorizing every transition from the training dataset, profiting from small transitions noise, rather than underlying relationships – overfitting.
The validation test can also be used for regularization by early stopping, shown in [63] to reduce overfitting. Early stopping method stops the training procedure when the profit on the validation dataset decreases, as this is a sign of overfitting to the training dataset [64]. However, this procedure is complicated in practice by the fact that the validation profit may fluctuate during training, producing multiple local minima. This complication has led to the creation of many ad-hoc rules for deciding when overfitting has truly begun. In this implementation the model is trained for 100 epochs but the learning may stop early if the validation test profit does not increase to a record high over 25 consecutive epochs. In each epoch if the current model records the highest validation test profit till that epoch, the model is saved.

After the training procedure a “real” test is performed on a out-of-sample dataset named test dataset. This is the test that this thesis proposes to solve and accomplish with good profits. The model used in this final test is the one that registered the highest validation test profit. The premise is that a model that has learned to perform well on the training dataset and then also performs well on a validation dataset, has a better likelihood to be profitable on the test dataset.

Figure 3.7 presents the implemented learning system and test architecture using early stopping method.

![Figure 3.7: System learning and test architecture.](image)

### 3.6.6 Online Learning

A particular reason why many supervised learning approaches fail in financial markets prediction and algorithmic trading is that the market is very volatile. This volatility is not only caused by short-term noise but also by the intrinsic market fundamentals, such as economic growth and technology growth. This means that any models trained with historical data, no matter how accurate they are on the training set, will ultimately fail on new market conditions. For example, if an algorithm was trained using historical data prior to the 2008 crisis and then was tested in this period, probably the test was going to be a huge failure because the 2008 crisis was an unprecedented market condition.

Usually, in an attempt to solve this problem, the test dataset is set to be as temporally close as
possible to the training and validation datasets so that market conditions have a smaller chance to have significantly change. However, it may not be enough and the bigger the test dataset, the higher are the probabilities of new market conditions be present. A possible approach is using a rolling window that uses small test datasets and each time a test is performed the window is rolled in order to test in a next period. However, the system always needs to start training from the beginning of the new window and this method is not practical and very difficult to implement in live trading. An online learning approach that can quickly adapt the model to new market conditions is therefore essential for a practical trading system. Fortunately, it is easy to implement an online version of the DQN algorithm. Basically, the learning algorithm can still be applied to the model during the test as it was applied during the learning on the training dataset. A new replay memory is going to be used so that the agent only samples test transitions instead of training transitions, that the agent probably has already experienced, facilitating and increasing the learning speed. In conclusion, during the test on the test dataset, the learning algorithm presented in Algorithm 4 is applied, without using an \( \epsilon \)-greedy policy, selecting actions according to the current belief about the optimal policy (model \( Q \) predictions).

### 3.6.7 Emulation

Only three actions are allowed to be performed in each state, and after each state, the reward associated with each action is actually known. This means that a random exploration method, \( \epsilon \)-greedy, used to sample an action is not needed because all the three actions can be emulated and the Q-network updated with the three of them. For the features state representation instead of only storing \((s, a, r, s')\), all the three emulated rewards are stored \((s, r_n, r_l, r_s, s')\) with \(r_n\) being the reward for the neutral action, \(r_l\) for long and \(r_s\) for short. Note that there is an absence of the action because all the three Q-values are going to be updated being needless to store the action to inform which Q-value is going to be updated. However, for the historical state representation the next state depends on the executed action. So, the actions must be emulated and there will be three different next states and three different rewards, storing all of them in the transition \((s, r_n, r_l, r_s, s'_n, s'_l, s'_s)\), where \(s'_n, s'_l, s'_s\) are the next states if the executed action was a neutral, long and short respectively. This not only enriches the training data, but also makes the training more stable.

The test results allowed to conclude that in fact, emulating the actions helps to stabilize the training and increase the learning speed, comparing to the use of \( \epsilon \)-greedy method, but only if Prioritized Experience Replay isn’t used. With Prioritized Experience Replay, emulating all the three actions did not improve the system. Therefore, emulation will not be part of the final system implementation but it was important to address this topic and provide some conclusions.

### 3.6.8 Different Paths

Most trading systems in the literature use daily trading frequency. At the beginning of this implementation a daily trading frequency was desired. However, with that frequency it was found in practice that the number of samples available is not enough to make the Q-learning algorithm converge. This con-
clusion is consistent with the fact that the only DQN approach to financial markets uses tick frequency trading (lower than second). However, tick frequency is not practical for live trading because of the expended time the algorithm takes to learn. With these two criteria in mind, a two hours trading frequency was chosen for the system because it was found in practice it has enough samples for the Q-learning algorithm to converge and the number of samples, however very high, does not make the learning time impractical for live trading.

To support this two hours trading frequency, the financial data frequency given to the system is minute frequency. It offers advantages because the user can simply adjust the trading frequency with a parameter named time_skip, meaning the number of minutes skipped between trading time steps. For example, in the two hours frequency time_skip = 120. Another advantage of this approach is that instead of using always the same data path during training, it is possible to train in each epoch with a different path through the data by starting each epoch in a different starting point of the training dataset. This methodology was proposed by [6], and it was concluded that by using a different path through the data each time the network is trained, the system is supplied with a larger variety of data, offering many advantages, including mitigating the overfitting phenomenon.

The standard testing approach is based on a single historical price path through the data, delivering a path-wise distribution of profits that may increase the possibilities of a “lucky” model. To avoid this, both training and validation tests will be performed on various paths reducing the chances of a “lucky” model. The profit average on these paths is the test result. It is logic that if a model performs well in a variety of different paths it may be a better predictor of future success. The number of paths is defined by a integer scalar variable named num_of_paths. Figure 3.8 presents an example of two different time_skip values while using two different paths. In the above system time_skip = 3 and in the bellow system time_skip = 2.

![Figure 3.8: Different Paths.](image)

3.6.9 Hyperparameter Selection

Selecting the hyperparameters of the system in order to optimize it, is a complex and hard procedure. A systematic grid search means that the network needs to be fully trained for each configuration of hyperparameters in order to be analyzed and conclusions taken. However, each test takes a considerable amount of time, proportional to the size of the dataset and the number of neurons in the DQN. It may take a single hour or even two days for a single test.
To tune the system hyperparameters, tests must be performed on different datasets and varying the parameters on the different tests. This allied to the fact that by changing an hyperparameter, other may no longer be optimal has made a systematic grid search unfeasible because of the high computational cost, as concluded by [3]. Thereby, the hyperparameters were selected using an informal search on different datasets from the EUR-USD financial market, although it is conceivable that better results could be obtained by systematically tuning the hyperparameter values.

Table 3.4 presents the list of hyperparameters and their values. RMSProp algorithm is the iterative optimization algorithm chosen and is used with minibatches of 32 transitions. The behaviour policy during training is performed with $\epsilon$ annealed linearly from 1.0 to 0.1 over the training episodes.

Table 3.4: List of hyperparameters and their values.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_skip</td>
<td>120</td>
<td>Interval of minutes skipped from each trading moment</td>
</tr>
<tr>
<td>num_of_paths</td>
<td>10</td>
<td>Number of different paths</td>
</tr>
<tr>
<td>minibatch_size</td>
<td>32</td>
<td>Number of transitions of each gradient descent.</td>
</tr>
<tr>
<td>training_D</td>
<td>50000</td>
<td>Training replay memory size.</td>
</tr>
<tr>
<td>test_D</td>
<td>5000</td>
<td>Test replay memory size.</td>
</tr>
<tr>
<td>C</td>
<td>5000</td>
<td>Target network update frequency.</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.99</td>
<td>Discount factor.</td>
</tr>
<tr>
<td>update_frequency</td>
<td>4</td>
<td>Number of actions between successive gradient descent updates.</td>
</tr>
<tr>
<td>learning_rate</td>
<td>0.00025</td>
<td>Learning rate used by RMSProp.</td>
</tr>
<tr>
<td>initial_exploration</td>
<td>1</td>
<td>Initial value of $\epsilon$ in $\epsilon$-greedy exploration.</td>
</tr>
<tr>
<td>final_exploration</td>
<td>0.1</td>
<td>Final value of $\epsilon$ in $\epsilon$-greedy exploration.</td>
</tr>
<tr>
<td>epochs</td>
<td>100</td>
<td>Number of epochs.</td>
</tr>
<tr>
<td>early_epoch</td>
<td>25</td>
<td>Early stop number of epochs</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>0.000001</td>
<td>Prioritized Experience Replay small positive constant.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.6</td>
<td>How much prioritization is used in experience replay.</td>
</tr>
</tbody>
</table>

3.7 Trading

The trading module is responsible for emulating trading in financial markets. It receives as input the trading signal outputted by the DQN module and simulates the execution of the action with the highest predicted Q-value on the trading signal in a financial market. The trading signal outputted by the DQN module has 3 components that represent the estimated Q-function for neutral, long and short actions. In the trading module it is verified which of them has the highest value and the action that it represents is the chosen action for the trading execution. Transaction costs are applied to simulate a real transaction and their value depends on the traded financial market. In Forex, a percentage of the total transaction value is charged. The transaction costs were applied according to the Interactive Brokers (largest electronic brokerage firm in the United States) low-cost online platform transaction costs and are given by:

$$\text{Transaction Cost} = 0.00002 \times \text{Transaction Value}. \quad (3.13)$$
Depending on the previous executed action, or in other words, the current market position, the actions will have different effects, presented in Table 3.5.

**Table 3.5: System's actions and their trading effect.**

<table>
<thead>
<tr>
<th>Action</th>
<th>Previous Action</th>
<th>Action Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>Nothing</td>
<td>Neutral</td>
</tr>
<tr>
<td>Neutral</td>
<td>Long</td>
<td>Close Long</td>
</tr>
<tr>
<td>Short</td>
<td></td>
<td>Close Short</td>
</tr>
<tr>
<td>Long</td>
<td>Neutral</td>
<td>Open Long</td>
</tr>
<tr>
<td>Short</td>
<td>Long</td>
<td>Hold Long</td>
</tr>
<tr>
<td>Short</td>
<td></td>
<td>Close Short and Open Long</td>
</tr>
<tr>
<td>Neutral</td>
<td>Long</td>
<td>Close Long and Open Short</td>
</tr>
<tr>
<td>Short</td>
<td></td>
<td>Hold Short</td>
</tr>
</tbody>
</table>

At the beginning, the system starts with a default capital and every time it performs an action it will trade as much of the available capital as possible. The default initial capital is 100000 USD. Through time, the portfolio of the system will only be able to trade one financial market. This module computes at each time step the available capital, the portfolio value, the number of transactions the system performed and the number of time periods with capital invested in the market.

Figure 3.9 presents a trading execution example in Forex, more specifically in the EUR-USD market. Every time the system's position changes, a transaction cost is applied, given by Equation (3.13). The Q-values used were randomly generated just for the example purpose.

![Figure 3.9: Trading execution example in EUR-USD market from 27/11/17 to 15/12/17.](image)
Chapter 4

Results

In this chapter the experiments and the results of the implemented DQN system described in the previous chapter are presented. First, the Forex markets, financial data and metrics used to evaluate the system are addressed. Then, the results of different combinations of the proposed state representations and reward functions in the DQN algorithm are presented and analyzed. Finally, the importance of some components in the system’s performance are evaluated.

The amount of time each test takes is dependent on the size of the dataset, the trading frequency and on the number of neurons in the DQN. Using the proposed 2 hours trading frequency and the final solution topology with three hidden layers with 200 neurons in each layer, the execution time normally takes between 1 to 4 hours. The same network topology, learning algorithm and hyperparameter settings were used across all experiments with the objective of showing that the approach is robust enough to work on a variety of different markets.

The implemented DQN system is not deterministic because it has a probabilistic and random nature when sampling transitions from the memory replay. This non-deterministic component makes the test results different from test to test. So, in order to accurately test the system’s performance, each experiment was repeated twenty times in order to obtain an average of the system's performance. From now on, all the results presented are the average of the system’s performance.

Presenting the return plots for all the tested markets would be too extensive. Therefore, only the return plots of two markets are presented in all experiments. The essential information about the other tested markets results can be obtained from the presented tables. The rest of the return plots are present in the Appendix’s.

4.1 Financial Markets and Financial Data

One of the goals of this thesis is to test the robustness of the proposed system in many Forex markets with different characteristics from each other. The financial data was acquired from Dukascopy Bank SA, which is considered to have a good quality and is reliable for backtests of strategies.

The financial data used to train and test the system was divided into 75% of the data to train the
system, 15% to perform the validation test and 10% to test it. Note that the data obtained does not have the same time periods for each market. Therefore, each experiment will be performed in different periods of time and with different number of train and test time steps. The experiments were performed in the following markets in the corresponding period of time:

- **EUR-USD** (Euro / U.S. dollar exchange pair), train from 26/08/2003 to 10/04/2014 (33371 trading time steps), validation from 10/04/2014 to 31/05/2016 (6690 trading time steps) and test from 31/05/2016 to 02/11/2017 (4460 trading time steps).

- **USD-JPY** (U.S. dollar / Japanese Yen exchange pair), train from 15/05/2003 to 19/03/2014 (33361 trading time steps), validation from 19/03/2014 to 23/05/2016 (6810 trading time steps) and test from 23/05/2016 to 02/11/2017 (4540 trading time steps).

- **AUD-USD** (Australian dollar / U.S. dollar exchange pair), train from 14/08/2003 to 10/04/2014 (33361 trading time steps), validation from 11/04/2014 to 30/05/2016 (6810 trading time steps) and test from 30/05/2016 to 02/11/2017 (4540 trading time steps).

- **GBP-USD** (Great Britain pound / U.S. dollar exchange pair), train from 15/05/2003 to 27/06/2014 (34041 trading time steps), validation from 27/06/2014 to 02/06/2016 (6820 trading time steps) and test from 02/06/2016 to 03/11/2017 (4550 trading time steps).

- **USD-CAD** (U.S. dollar / Canadian dollar exchange pair), train from 14/08/2003 to 21/04/2014 (33451 trading time steps), validation from 21/04/2014 to 13/06/2016 (6710 trading time steps) and test from 13/06/2016 to 15/11/2017 (4470 trading time steps).

### 4.2 Evaluation Metrics

The main goal of the proposed system is to maximize the financial gains. However, it also try to minimize associated risks and therefore minimize the number of trading periods spent with capital invested in the market and the maximum capital drawdown. The system is also evaluated for its capacity to profit without incurring too many transaction costs changing its trading position many times. Therefore, the metrics used to evaluate the system's performance are:

- The number of transactions evaluates if the system is paying too many transaction costs or not.

- The **Rate of Return** (ROR), presented in equation 4.1, is the gain or loss on an investment over a specified time period, expressed as a percentage of the investment’s cost. It is usually compared with the BnH strategy returns because Market Efficient Hypothesis [2] state that it is impossible to beat the market using any kind of analysis, with the best available strategy for any investor being a BnH strategy.

\[
ROR = \left( \frac{\text{Final Capital} - \text{Initial Capital}}{\text{Initial Capital}} \right) \times 100
\]  

- The number of trading periods spent with capital invested in the market (Market Periods).
• The ROR per trading period spent in the market, presented in equation 4.2, evaluates if the returns obtained are in line with the goal of maximizing returns while minimizing the number of trading periods with capital invested in the market.

\[ \text{ROR/period} = \frac{\text{ROR}}{\text{Market Periods}} \] (4.2)

• The Maximum Drawdown (MDD), presented in equation 4.3, is the maximum loss of capital from a peak to a trough before a new peak is attained. It is an indicator of downside risk over a specified time period and is desirable to have low values.

\[ \text{MDD} = \max_t (\text{ROR}_t - \text{ROR}), \text{ with } t \in \text{trading period} \] (4.3)

• The Risk Return Ratio (RRR), presented in equation 4.4, evaluates the amount of risk undertaken to capture returns. It is in line with the goal of maximizing returns while minimizing risk and maximum drawdown. High values of RRR are desirable.

\[ \text{RRR} = \frac{\text{ROR}}{\text{MDD}} \] (4.4)

### 4.3 Topology Study

One of the main goals of the system’s implementation was creating a system’s framework robust to a variety of different Forex markets using the same network topology, learning algorithm and hyperparameter settings. Therefore, a suitable topology that allows the system to learn in any Forex market data and achieve good test results, not sporadically but consistently, is desired. The evaluation of the network topologies were made with the profit results during training, validation and test. Each experiment was repeated five times and therefore the profit results are the average of the five experiments results.

Experiments using combinations of the proposed state and reward functions were performed, but not only in the proposed system described in chapter 3, but also with DQN systems with some component differences. The reason for performing these experiments was to conclude if the more suitable topologies for the proposed system would also be the best if the DQN system would have some component differences. The proposed system described in chapter 3 without PCA, and the proposed system without Prioritized Experience Replay were the system’s component differences tested. The experiments were conducted in all the presented Forex markets on their experimentation periods of time.

The tested topologies varied on the number of hidden layers and on the hidden layers number of neurons. All the tested topologies were fully-connected. However, because of time and computational cost constrains, all the experiments had the same number of neurons in each hidden layer when the topology had more than one hidden layer. A possible approach would be experimenting topologies using hidden layers with different number of neurons from each other. The experimented number of hidden layers and number of neurons in a layer are presented in table 4.1.
Table 4.1: Number of experimented hidden layers and neurons in a layer.

<table>
<thead>
<tr>
<th>Number of hidden layers</th>
<th>1; 2; 3; 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of neurons in a layer</td>
<td>3; 5; 7; 10; 15; 20; 30; 50; 100; 200; 350; 500; 1000</td>
</tr>
</tbody>
</table>

The experiments conclusions were that using a single hidden layer or four hidden layers, the system would not converge in some tests, independently of the hidden layers number of neurons. Using two and three hidden layers the system converged in all the experiments. However, generally, the performance of the system was found to be better when using three hidden layers. Regarding the number of neurons, it was much more difficult to reach a conclusion. There isn’t a specific number of neurons that performs better than the others on all the experiments. However, all the different number of neurons had experiments that the profit results were not good, except for a single number of neurons. Using three hidden layers with 200 neurons in each layer is one of, if not the best, topologies in terms of profit results on all the performed experiments. Therefore, it was found to be the most robust solution and it is the adopted topology for the proposed system.

Note that the results of these experiments are not presented because if they were presented for all the tested markets, systems and topologies it would be too extensive and the essential information about the conclusions were summarized and presented.

4.4 Case Study I - State and Reward Functions

Two different state representations and three different reward functions were proposed in chapter 3. The first experiment tested combinations of these functions to test the quality of the system’s results obtained by each of these combinations. To easily identify each of the combinations they are going to be named as:

- Features-period: Features state representation and period return reward function.
- Historical-period: Historical state representation and period return reward function.
- Features-profit: Features state representation and transaction profit reward function.
- Historical-profit: Historical state representation and transaction profit reward function.
- Features-cost: Features state representation and cost reward function.
- Historical-cost: Historical state representation and cost reward function.

The average results obtained by each system combination on the test data of the Forex markets are presented in Table 4.2. The results of the BnH strategy are also presented for comparison.

Analyzing the results obtained, the following observations can be made:

- In the EUR-USD market all the systems clearly outperform the BnH strategy. It can be noted that Historical-profit system has the best performance having the highest ROR, RRR and ROR/period
Table 4.2: Results of the Buy and Hold (BnH) strategy and the different state and reward functions combinations (average).

<table>
<thead>
<tr>
<th></th>
<th>BnH</th>
<th>Features-period</th>
<th>Historical-period</th>
<th>Features-profit</th>
<th>Historical-profit</th>
<th>Features-cost</th>
<th>Historical-cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EUR-USD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>819.6</td>
<td>1083.6</td>
<td>636.8</td>
<td>1395.2</td>
<td>642.6</td>
<td>893</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>3.92</td>
<td>19.21</td>
<td>12.87</td>
<td>10.28</td>
<td>21.81</td>
<td>6.67</td>
<td>16.61</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4460</td>
<td>2630.4</td>
<td>2490.6</td>
<td>2294.9</td>
<td>2764.2</td>
<td>1864.4</td>
<td>4009.0</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>0.0009</td>
<td>0.0069</td>
<td>0.0057</td>
<td>0.0047</td>
<td>0.0076</td>
<td>0.0021</td>
<td>0.0026</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>10.50</td>
<td>5.36</td>
<td>5.33</td>
<td>6.41</td>
<td>5.28</td>
<td>7.60</td>
<td>7.30</td>
</tr>
<tr>
<td>RRR</td>
<td>0.37</td>
<td>2.72</td>
<td>1.66</td>
<td>1.03</td>
<td>3.64</td>
<td>0.98</td>
<td>2.47</td>
</tr>
<tr>
<td><strong>USD-JPY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>566.2</td>
<td>891.6</td>
<td>603.7</td>
<td>442.0</td>
<td>716.7</td>
<td>882.7</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>3.63</td>
<td>5.15</td>
<td>3.06</td>
<td>4.63</td>
<td>1.10</td>
<td>3.60</td>
<td>4.66</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4540</td>
<td>1941.8</td>
<td>2318.4</td>
<td>1969.3</td>
<td>4045.3</td>
<td>2045.4</td>
<td>1744.1</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>0.0008</td>
<td>0.0030</td>
<td>0.0037</td>
<td>0.0027</td>
<td>0.0001</td>
<td>0.0025</td>
<td>0.0023</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>10.34</td>
<td>7.11</td>
<td>8.73</td>
<td>6.82</td>
<td>8.78</td>
<td>8.41</td>
<td>7.26</td>
</tr>
<tr>
<td>RRR</td>
<td>0.35</td>
<td>1.88</td>
<td>0.7</td>
<td>1.08</td>
<td>0.14</td>
<td>0.77</td>
<td>0.95</td>
</tr>
<tr>
<td><strong>AUD-USD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>951.8</td>
<td>1521.7</td>
<td>943.8</td>
<td>1340.3</td>
<td>1020.8</td>
<td>1261</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>4.84</td>
<td>14.94</td>
<td>10.23</td>
<td>11.19</td>
<td>12.88</td>
<td>6.81</td>
<td>9.89</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4540</td>
<td>3507.1</td>
<td>3458.3</td>
<td>2707.5</td>
<td>4127.2</td>
<td>2368.4</td>
<td>3746.4</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>0.0011</td>
<td>0.0044</td>
<td>0.0021</td>
<td>0.0038</td>
<td>0.0030</td>
<td>0.0023</td>
<td>0.0024</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>6.05</td>
<td>4.79</td>
<td>5.99</td>
<td>4.52</td>
<td>5.90</td>
<td>5.78</td>
<td>5.75</td>
</tr>
<tr>
<td>RRR</td>
<td>0.8</td>
<td>3.56</td>
<td>2.10</td>
<td>3.04</td>
<td>2.77</td>
<td>1.49</td>
<td>2.05</td>
</tr>
<tr>
<td><strong>GBP-USD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>953.8</td>
<td>891.74</td>
<td>868.3</td>
<td>1350.8</td>
<td>902.2</td>
<td>1015.6</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>-12.83</td>
<td>5.59</td>
<td>3.26</td>
<td>1.59</td>
<td>10.15</td>
<td>-2.08</td>
<td>0.74</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4550</td>
<td>3507</td>
<td>3048.5</td>
<td>3110.9</td>
<td>3840.8</td>
<td>3084.9</td>
<td>2914.9</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>-0.0028</td>
<td>0.0016</td>
<td>0.0005</td>
<td>0.0013</td>
<td>0.0025</td>
<td>-0.0020</td>
<td>-0.0022</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>29.1</td>
<td>15.61</td>
<td>15.98</td>
<td>16.90</td>
<td>17.48</td>
<td>16.98</td>
<td>15.88</td>
</tr>
<tr>
<td>RRR</td>
<td>-0.44</td>
<td>0.62</td>
<td>0.45</td>
<td>0.45</td>
<td>0.89</td>
<td>-0.06</td>
<td>0.33</td>
</tr>
<tr>
<td><strong>USD-CAD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>819.7</td>
<td>895.8</td>
<td>734.7</td>
<td>1100.8</td>
<td>829.4</td>
<td>904</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>-0.29</td>
<td>6.79</td>
<td>4.78</td>
<td>6.62</td>
<td>5.97</td>
<td>5.54</td>
<td>3.17</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4470</td>
<td>3025.1</td>
<td>2924.0</td>
<td>2737.0</td>
<td>3157.5</td>
<td>2990</td>
<td>2636.3</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>-0.0001</td>
<td>0.0022</td>
<td>0.0017</td>
<td>0.0019</td>
<td>0.0015</td>
<td>0.0017</td>
<td>0.0017</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>13.47</td>
<td>6.70</td>
<td>7.36</td>
<td>6.20</td>
<td>7.93</td>
<td>7.53</td>
<td>6.96</td>
</tr>
<tr>
<td>RRR</td>
<td>-0.02</td>
<td>1.42</td>
<td>1.14</td>
<td>1.26</td>
<td>1.09</td>
<td>1.21</td>
<td>0.73</td>
</tr>
</tbody>
</table>

and the lowest MDD. Features-period system has also a good performance having the second highest ROR, RRR and ROR/period.

- In the USD-JPY market only some of the systems outperform the BnH strategy but none of them clearly outperforms it. Japan is extremely dependent on the export/import ratio because of a lack of
natural resources in Japan. When the commodities prices rise it causes the JPY to depreciate. The economy of Japan also grows much slower than its trading partners causing the Japan’s export-dependent economy and the JPY to weaken. Whenever there is movements in the JPY that might threaten Japanese exports or economic growth, the Bank of Japan has a reputation for intervening in the Forex market. These interventions can cause the market to make a U-turn movement without any possible prediction capability. For this reason USD-JPY market is usually difficult to predict and it might explain the difficulty in outperforming the BnH strategy. Features-period is the best performing system outperforming the BnH strategy and having the highest ROR and RRR. Features-period and Historical-cost also outperform the BnH strategy but their returns and the other criteria are not as good as Features-period. Historical-period and Features-cost are slightly outperformed and Historical-profit is completely outperformed by the BnH strategy. Historical-profit not only has the lowest returns but also has the worst ROR/period, MDD and RRR.

- In the AUD-USD market all the systems outperform the BnH strategy. Features-period is the best performing system clearly outperforming the BnH strategy and having the highest returns, ROR/period and RRR and the second lowest MDD of all systems. Features-cost has the worst performance with the lowest returns and RRR. Features-profit and Historical-profit have similar performances but Features-profit is slightly better because, despite having slightly inferior returns than Historical-profit, it has higher RRR and ROR/period and lowest MDD, producing a solution with almost the same returns but with lowest investment risk. Historical-period and Historical-cost have intermediate performances.

- In the GBP-USD market none of the systems outperform the BnH strategy because despite this strategy produces negative results, if the trader performed a short and hold (traded all his money to USD), from the beginning till the end of the trading period, would have produced 12.83% positive returns, above all the systems returns. Historical-profit is the best performing system clearly outperforming the returns of all the other systems and having the highest ROR/period and RRR. Features-period has slightly better performance than all the other systems (except Historical-profit) having the highest ROR/period and lowest MDD and its RRR is the second highest of all systems. Features-cost performance is the only experiment in all the markets with negative returns. All the other systems have intermediate performances.

- In the USD-CAD market all the systems clearly outperform the BnH strategy. All systems have similar performances but Features-period has the best performance with the highest returns, ROR/period and RRR. Features-profit has a very similar performance to Features-period in all the criteria and has the lowest MDD. Historical-cost has clearly the worst performance having the lowest returns and MDD. All the other systems have intermediate performances.

It can be concluded that in the tested markets, all the systems have very similar performances outperforming the BnH in almost all the experiments except in the GBP-USD market. Despite the similar performances it can be noted that Features-period is the best system in the USD-JPY, AUD-USD and USD-CAD markets and Historical-profit is the best system in EUR-USD and GBP-USD. It can also be
observed that none of the other systems proved to consistently outperform these systems in all the evaluated criteria. Historical-period, Features-period, Features-cost and Historical-cost are outperformed by Features-period in all markets and by Historical-profit in all markets except in the USD-JPY.

In GBP-USD market, Historical-profit clearly outperforms Features-period and it is the system closer to the BnH strategy returns. However, in the USD-JPY Historical-profit is the worst system of them all and is outperformed by the BnH strategy. In the EUR-USD market, Historical-profit has a slightly better returns performance but regarding RRR it clearly outperforms Features-period. In AUD-USD and USD-CAD markets, Features-period has slightly better performances in all the evaluated criteria comparing to Historical-profit. In all markets, Features-period spent less market trading periods with capital invested when compared to Historical-profit.

With these observations, Features-period is considered to be the best system. The reason behind this choice is that Features-period achieves more consistent overall results in maximizing the returns, outperforming the BnH strategy in all the markets except in the GBP-USD, where all the systems were also outperformed, and produces at least 5.15% returns on all the markets. Historical-profit is also considered to be a good system but its performance is not as consistent with the proposed goals, specially in the USD-JPY market where it was outperformed by the BnH strategy having just 1.1% returns. Features-period also presents consistent results in minimizing the risk and the number of trading periods with capital invested in the market when compared to Historical-profit. Thereby the final proposed system uses features state and period return reward functions (Features-period).

In Figure 4.1 it’s presented the evolution of the returns obtained by the BnH strategy and the average returns obtained by Features-period, Historical-profit and Features-cost in the EUR-USD market during the test period. The reason for only presenting the evolution of these three systems is that if all the systems were displayed, the system’s lines would overlap in each other and would be difficult to observe and analyze the evolution of the returns. Thereby, all the figures showing the evolution of the returns will only present Features-period and Historical-profit, because they were considered to be the best performing systems, and also the system with the worst performance in that market, Features-cost in this case.

Analyzing Figure 4.1, it can be observed that the BnH strategy returns has a downward trend in the first third of the test period and an upward trend in the rest of the period with the exception of a small decrease in the end. So, there is a clear opportunity to outperform considerably the BnH strategy, specially in the downward trend. When a market moves in a upward or downward trend it is usually easier to make investments decisions capable of profiting and outperforming the BnH strategy. However, when there is a sideways trend, decision-making is markedly more difficult and therefore more difficult to outperform the BnH strategy. It can be observed that Features-period and Historical-profit systems clearly outperform the BnH strategy. However, Features-cost only outperforms slightly the BnH strategy. In the first third of the period, short is mainly used by all systems to profit from the fall of prices. In the rest of the period, the upward trend, Features-period and Historical-profit continue to profit but Features-cost is not capable of profiting.
Figure 4.1: Returns obtained by Features-period, Historical-profit, system with the worst performance (Features-cost) and the Buy and Hold strategy in the EUR-USD exchange market.

In Figure 4.2 it’s presented the evolution of the returns obtained by the BnH strategy and the average returns obtained by Features-period, Historical-profit and Features-cost (worst performance) in the GBP-USD market during the test period.

Despite GBP-USD pair being one of the most liquid trades in Forex, Britain’s surprising decision to withdraw from the European Union, also known as Brexit, had a massive effect on weakening the GBP in a short period of time. Analyzing Figure 4.2, it can be observed that the BnH strategy returns has a
big downward trend in the first third of the test period (Brexit), and a sideways trend (from 20/10/2016 to 09/03/2017) followed by a slow recovery of the market (down from -25% returns to -12%) in the rest of the period. So, there is a clear opportunity to outperform considerably the BnH strategy, especially in the downward trend and in the slowly upward trend. It can be observed that Features-cost could not predict the downward trend recording only 5% returns contrarily to the -15% returns of the BnH. Then, in the sideways trend it starts slowly losing profits and in the upward trend the system could not understand the underlying trend, losing all the profits and recording a -2% return. Both Features-period and Historical-profit predicted the downward trend, recording 12-13% returns, but considering that the BnH strategy lost 25% in returns, there is a big space for improvement. However, in the rest of the test period both systems did not understand the underlying trends, with Historical-profit being almost constant in terms of returns obtained till the end of the period, and Features-period loosing more than 5% of the profits obtained.

4.5 Case Study II - Importance of Each Component

In the first case study it was determined the more robust and suitable combination for state representation and reward functions to the application of this thesis. In the second case study, the importance of some of the implemented components on the system’s performance were tested. Note that this experiment uses the final proposed system using the state and reward combination found to be the more robust. It uses Features-period with features state and period return reward functions. More concretely, the importance of the PCA technique, Prioritized Experience Replay and Double Q-learning were tested and analyzed. Each experiment tests the proposed system without one of these components in order to analyze and compare their results with the proposed system results.

To test the importance of the PCA technique in the performance of the system, instead of applying dimensionality reduction with the PCA technique, the normalized 30 features combining technical indicators and raw financial data are presented as input to the DQN algorithm in the DQN module.

To test the importance of the Prioritized Experience Replay method in the performance of the system, instead of sampling more frequently experience transitions from the replay memory with high expected learning progress, the transitions are sampled in a uniform manner.

To test the importance of the Double Q-learning in the performance of the system, instead of evaluating the greedy policy according to the online policy network and estimate its value according to the target network, both evaluation and estimation are made by the same network. The difference in the implementation is in the optimal target values that are calculated by:

\[
y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; W_i) = r_j + 1 + \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; W_i); W_i)
\]

Table 4.3 presents the results of the final system, and without the PCA technique, without Prioritized Experience Replay, and without Double Q-learning. Each of the components importance is going to be analyzed in the next subsections.
Table 4.3: Results of the Buy and Hold strategy and the final system, and without PCA, without Prioritized Experience Replay, and without Double Q-learning (average).

<table>
<thead>
<tr>
<th>Currency</th>
<th>BnH</th>
<th>Final System</th>
<th>w/o PCA</th>
<th>w/o Prioritized Experience Replay</th>
<th>w/o Double Q-learning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EUR-USD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>819.6</td>
<td>1424.1</td>
<td>461.3</td>
<td>610.6</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>3.92</td>
<td>19.21</td>
<td>-2.36</td>
<td>3.61</td>
<td>5.50</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4460</td>
<td>2630.4</td>
<td>4338.0</td>
<td>1842.4</td>
<td>3575.7</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>0.0009</td>
<td>0.0069</td>
<td>-0.0005</td>
<td>0.0005</td>
<td>0.0001</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>10.5</td>
<td>5.36</td>
<td>15.34</td>
<td>6.28</td>
<td>7.81</td>
</tr>
<tr>
<td>RRR</td>
<td>0.37</td>
<td>2.72</td>
<td>-0.15</td>
<td>10.67</td>
<td>0.19</td>
</tr>
<tr>
<td><strong>USD-JPY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>566.2</td>
<td>840</td>
<td>651.5</td>
<td>982.7</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>3.63</td>
<td>5.15</td>
<td>-1.92</td>
<td>1.41</td>
<td>1.07</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4540</td>
<td>1941.8</td>
<td>1943.6</td>
<td>1990.7</td>
<td>2623.24</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>0.0008</td>
<td>0.0030</td>
<td>-0.0001</td>
<td>0.0011</td>
<td>0.0005</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>10.34</td>
<td>7.11</td>
<td>8.74</td>
<td>8.40</td>
<td>10.62</td>
</tr>
<tr>
<td>RRR</td>
<td>0.35</td>
<td>1.88</td>
<td>-0.16</td>
<td>0.69</td>
<td>0.38</td>
</tr>
<tr>
<td><strong>AUD-USD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>951.8</td>
<td>665.4</td>
<td>999.2</td>
<td>979.3</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>4.84</td>
<td>14.94</td>
<td>5.02</td>
<td>9.04</td>
<td>8.67</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4540</td>
<td>3507.1</td>
<td>2069.4</td>
<td>2341.4</td>
<td>2338</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>0.0011</td>
<td>0.0044</td>
<td>0.0010</td>
<td>0.0039</td>
<td>0.0038</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>6.05</td>
<td>4.79</td>
<td>4.33</td>
<td>4.95</td>
<td>4.54</td>
</tr>
<tr>
<td>RRR</td>
<td>0.8</td>
<td>3.56</td>
<td>1.06</td>
<td>2.38</td>
<td>2.16</td>
</tr>
<tr>
<td><strong>GBP-USD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>953.8</td>
<td>608.7</td>
<td>923</td>
<td>1020.6</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>-12.83</td>
<td>5.59</td>
<td>-0.69</td>
<td>3.43</td>
<td>4.11</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4550</td>
<td>3507</td>
<td>2118.3</td>
<td>2499.46</td>
<td>2440</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>-0.0028</td>
<td>0.0016</td>
<td>-0.0004</td>
<td>0.0014</td>
<td>0.0016</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>29.1</td>
<td>15.61</td>
<td>10.66</td>
<td>13.02</td>
<td>13.48</td>
</tr>
<tr>
<td>RRR</td>
<td>-0.44</td>
<td>0.62</td>
<td>-0.06</td>
<td>0.48</td>
<td>0.57</td>
</tr>
<tr>
<td><strong>USD-CAD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>2</td>
<td>6.8</td>
<td>778</td>
<td>846.1</td>
<td>759.3</td>
</tr>
<tr>
<td>ROR (%)</td>
<td>-0.29</td>
<td>6.79</td>
<td>5.96</td>
<td>4.75</td>
<td>6.70</td>
</tr>
<tr>
<td>Market Periods</td>
<td>4470</td>
<td>3025.1</td>
<td>2191.2</td>
<td>2565</td>
<td>2752.3</td>
</tr>
<tr>
<td>ROR/period (%)</td>
<td>-0.0001</td>
<td>0.0022</td>
<td>0.0027</td>
<td>0.0019</td>
<td>0.0026</td>
</tr>
<tr>
<td>MDD (%)</td>
<td>13.47</td>
<td>6.70</td>
<td>4.45</td>
<td>7.38</td>
<td>6.17</td>
</tr>
<tr>
<td>RRR</td>
<td>-0.02</td>
<td>1.42</td>
<td>1.59</td>
<td>0.98</td>
<td>1.38</td>
</tr>
</tbody>
</table>

4.5.1 Case Study II.a - Influence of the PCA Technique

Analyzing the results obtained by the final system without the PCA technique, the following observations can be made:
• It is clearly outperformed in all markets by the final system using PCA and even has negative returns in the USD-JPY and GBP-USD markets. The system without PCA only outperforms the BnH strategy in the AUD-USD and USD-CAD.

• It has worst results in the ROR/period, MDD and RRR in all markets, except in the USD-CAD, when compared to the system with PCA.

• The only market in which the system without PCA and the system with PCA have similar performance is the USD-CAD, where the system without PCA has a higher ROR/period and RRR and lower MDD but in the ROR is still underperformed but for a difference lesser than 1% (5.96% without PCA and 6.79% with PCA).

It can be concluded that the importance of the PCA technique in the final system is significant. This is attested by the results showing that the system with PCA produces much higher returns in all markets, except in the USD-CAD where the system still produces slightly higher returns, when compared to the results produced by the system without PCA. In fact, without PCA the system would only outperform the BnH strategy in 2 markets and producing negative returns in other 3 markets. Regarding the other evaluated metrics, it can also be concluded that in general the system using PCA produces results with less financial risk having better MDD, RRR and ROR/period results.

4.5.2 Case Study II.b - Influence of Prioritized Experience Replay

Analyzing the results obtained by the final system without Prioritized Experience Replay, the following observations can be made:

• It is clearly outperformed in all markets by the final system using Prioritized Experience Replay. Without Prioritized Experience Replay the system only outperforms the BnH strategy in the AUD-USD and USD-CAD, the same markets as the system without PCA.

• Its lower returns are in the USD-JPY market where it produced only 1.41% returns. Its higher returns are in the AUD-USD market where it produced 9% returns but still distant of the 15% returns obtained by the system using Prioritized Experience Replay.

• The system without Prioritized Experience Replay has worst results in the ROR/period, MDD and RRR in all markets when compared to the system with Prioritized Experience Replay.

It can be concluded that the importance of Prioritized Experience Replay in the final system is also significant. This is attested by the results showing that the system with Prioritized Experience Replay produces much higher returns in all markets when compared to the results produced by system without Prioritized Experience Replay. Equally to the system without PCA, the system without Prioritized Experience Replay would only outperform the BnH strategy in 2 markets. Regarding the other evaluated metrics it can also be concluded that in all markets the system with Prioritized Experience Replay produces results with less financial risk having better MDD, RRR and ROR/period results.
4.5.3 Case Study II.c - Influence of Double Q-learning

Analyzing the results obtained by the final system without Double Q-learning, the following observations can be made:

- It is outperformed in all markets by the system using Double Q-learning. Without Double Q-learning the system outperforms the BnH strategy in the EUR-USD, AUD-USD and USD-CAD.

- Despite the system without Double Q-learning being outperformed in all markets, both systems have a similar performance in the USD-CAD with almost the same results in every evaluation metrics. Even in the GBP-USD, both performances are very similar despite the returns obtained by the system without Double Q-learning being slightly lower comparing to the system using it.

- In general, the system without Double Q-learning is outperformed in the other evaluation metrics in the EUR-USD, USD-JPY and AUD-USD markets having worst ROR/period, MDD and RRR results.

It can be concluded that the importance of Double Q-learning in the final system is also significant but not as significant as the importance of the PCA technique and Prioritized Experience Replay. This is attested by the results showing that the system with Double Q-learning produces higher returns in all markets when compared to the results produced by the system without Double Q-learning. However, the difference between both performances is not as high as the difference between the system and the system without PCA or Prioritized Experience Replay. In fact, the system without Double Q-learning would outperform the BnH strategy in 3 markets, only 1 less than the final system, having its lower returns in the USD-JPY where it only has 1% returns, but having its highest returns set in the AUD-USD where it has 8.67% returns. Also, it produces very similar performances when comparing to the system using Double Q-learning in two markets, GBP-USD and USD-CAD.

Figures 4.3 and 4.4 present the evolution of the returns obtained by the BnH strategy and the average returns obtained by the final system and the final system without PCA, without Prioritized Experience Replay (PER), and without Double Q-learning in the EUR-USD and GBP-USD markets respectively.

It can be observed that in both markets the system without PCA is clearly outperformed by the system with PCA. The system without Prioritized Experience Replay is also clearly, but not as much as the system without PCA, outperformed by the system using Prioritized Experience Replay. Finally, it can be observed that the system without Double Q-learning is also outperformed by the system using Double Q-learning but it’s the system with the closer performance to the final system.

4.6 Case Study III - Performance Comparison

The markets are always evolving and changing its characteristics. Economic and technology growth are one of the reasons, but the growth of algorithmic trading is also a key factor. The presence of many more financial markets trading algorithms in recent years was found to increase the markets volatility’s and that their actions and strategies seem to be less diverge, more correlated, than those of non-algorithmic traders [65]. To truly observe that the proposed system can perform in real-life markets it is
Figure 4.3: Returns obtained by the BnH strategy and the average returns obtained by the final system, and without PCA, without Prioritized Experience Replay (PER) and without Double Q-learning in the EUR-USD exchange market.

Figure 4.4: Returns obtained by the BnH strategy and the average returns obtained by the final system, and without PCA, without Prioritized Experience Replay (PER) and without Double Q-learning in the GBP-USD exchange market.

It is important to test it in recent periods of time where the market conditions are closer to the real-life market conditions. Comparing the performance of the final system with other related works only makes sense if the other works were tested in recent periods of time. Also, because the systems are probably better and advanced approaches. Comparing the final system performance in a distant period of time will only contribute to observe if the final system has better prediction capability in that distant market conditions.
than other works but it doesn’t give any conclusion regarding if the system is better in recent market conditions, where it can have a real practical use.

However, there is a lack of public available machine learning approaches to Forex trading and even lesser in recent years. Mainly, public machine learning trading algorithms are applied to stock markets. Therefore the system will only be compared with the DQN system applied to Forex trading proposed in 2017 by Carapuço [6]. That system was tested in different annual time periods in the EUR-USD market. Therefore, to establish a performance comparison, the final proposed system was tested in three different years as Carapuço system was. The time periods used for these experiments in the EUR-USD are:

- Train from 12/06/2003 to 31/12/2012, validation from 01/01/2013 to 31/12/2013 and test from 01/01/2014 to 31/12/2014.
- Train from 12/06/2003 to 31/12/2013, validation from 01/01/2014 to 31/12/2014 and test from 01/01/2015 to 31/12/2015.
- Train from 12/06/2003 to 31/12/2014, validation from 01/01/2015 to 31/12/2015 and test from 01/01/2016 to 31/12/2016.

The results of the BnH strategy, the final proposed system and of Carapuço system when trading in the EUR-USD market in the different time periods are presented in Table 4.4. Note that Carapuço system did not apply transaction costs.

Table 4.4: Results of the Buy and Hold (BnH) strategy, the final proposed system and of Carapuço system when trading in the EUR-USD market in the different time periods (average).

<table>
<thead>
<tr>
<th>ROR (%)</th>
<th>BnH</th>
<th>Final Proposed System</th>
<th>Carapuço System</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUR-USD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>-16.56</td>
<td>21.24</td>
<td>13.86</td>
</tr>
<tr>
<td>2015</td>
<td>-12.43</td>
<td>27.90</td>
<td>12.26</td>
</tr>
<tr>
<td>2016</td>
<td>-2.63</td>
<td>7.02</td>
<td>4.86</td>
</tr>
</tbody>
</table>

It can be observed that Carapuço system only outperforms the BnH strategy in 2016. The final proposed system clearly outperforms Carapuço system in all tested years, despite its system did not apply transaction costs, something that can make a huge performance difference. It also outperforms as well the BnH strategy in all tested years. These observations lead to the conclusion that the final proposed system is more capable of profiting in recent market conditions than Carapuço system, attested by the results in these three recent tested time periods.
Chapter 5

Conclusions

This thesis presents a DQN system framework, using Double Q-learning and Prioritized Experience Replay extensions, that uses the same network topology, system’s hyperparameters values and raw financial data parameters in every Forex market with the goal of maximizing financial returns while minimizing financial risk, by timely selecting the best trading actions when trading in Forex markets.

To achieve this goal, market environment is summarized, technical analysis is used to extract features from the raw financial data mitigating data noise and uncertainty and the feature learning PCA technique is then applied to the extracted features and raw financial data with the objective of learning automatically features that contain most of the data information in a low dimensional manner. The DQN algorithm uses these learned features to learn market patterns and trends by applying deep learning, while at the same time learns to predict the best trading actions.

The implemented system was tested in five Forex markets with very different characteristics to validate the robustness of the system’s performance.

5.1 Conclusions

In this work two different state functions and three different reward functions were proposed. Six combinations of these functions were tested to find the most robust solution to the DQN algorithm. The results obtained by these different combinations were much more similar than what was initially expected. This is because most of the tested reward functions didn’t make the system to converge and in the end the system’s combinations performances are not that different from what they could possibly be. From the results obtained, it can be concluded that using a state representation with the low dimensional features learned by the PCA technique and a reward function based on the return percentage of a trading period, allows the system to perform well and consistently in a variety of five Forex markets with different characteristics from each other, outperforming the BnH strategy in four of them and having at least 5.15% returns in any of them.

The importance of the PCA technique, Prioritized Experience Replay and Double Q-learning in the performance of the system was tested. The conclusion is that each of these components are essential
to system’s performance, specially the PCA technique and the Prioritized Experience Replay:

- The PCA technique allows the system to use many technical indicators containing essential market information, that enhance the learning of the market status by the DQN algorithm, by reducing their dimensionality while maintaining the original data essence, avoiding overfitting and enhancing the generalization capability of the DQN algorithm. Without PCA the system would be more prone to overfitting, loosing its generalization capability, or, to avoid overfitting, would only use some technical indicators that may not present all the essential market environment information to the system.

- Prioritized Experience Replay allows the DQN algorithm to learn much faster by selecting transitions with high expected learning progress more frequently.

- Double Q-learning allows the DQN algorithm to solve a problem of overestimated value estimates that could cause the algorithm to choose wrong trading actions.

Comparing to Carapuço system it was concluded that this system is more capable of profiting in recent market conditions, even if Carapuço system did not apply transaction costs, attested by the results in three recent annual tested periods.

### 5.2 Future Work

As a follow-up to this work, many directions of study could be taken. Some of those directions are:

- The system hyperparameters were selected using an informal search on the EUR-USD market. It may not be a coincidence that the best results shown by the system were also in the EUR-USD market. Doing an informal search on the system’s hyperparameters for every market would probably improve its results.

- The pursuit of more suitable state and reward functions for the proposed goal is one of the key future directions. Despite all the proposed state and reward combinations having similar results, most of the experimented functions throughout the implementation phase of this work didn’t make the algorithm to converge. Therefore it would be a mistake to conclude that the state and reward functions are not one of, if not the, key components of the system.

- The experiment of network topologies with different number of neurons in each hidden layer.

- A more suitable model candidate selection would also be of good direction to improve the system’s performance.

- The experiment of the proposed system in different type of financial markets like stock markets and commodities.

- An approach using a NEAT algorithm to select the network topology and system’s hyperparameters would also be an interesting future direction.
Bibliography


Appendix A

Return Plots of Case Study I

A.1 USD-JPY Exchange Market

Figure A.1: Average returns obtained by Features-period, Historical-profit and Features-profit systems and Buy and Hold returns in the USD-JPY exchange market.
A.2 AUD-USD Exchange Market

Figure A.2: Average returns obtained by Features-period, Historical-profit, the system with the worst performance (Features-cost) and Buy and Hold returns in the AUD-USD exchange market.
A.3 USD-CAD Exchange Market

Figure A.3: Average returns obtained by Features-period, Historical-profit, the system with the worst performance (Historical-cost) and Buy and Hold returns in the USD-CAD exchange market.
Appendix B

Return Plots of Case Study II

B.1 USD-JPY Exchange Market

Figure B.1: Returns obtained by the BnH strategy and the average returns obtained by the final system, by the final system without PCA, by the final system without Prioritized Experience Replay (PER) and by the final system without Double Q-learning in the USD-JPY exchange market.
Figure B.2: Returns obtained by the BnH strategy and the average returns obtained by the final system, by the final system without PCA, by the final system without Prioritized Experience Replay (PER) and by the final system without Double Q-learning in the AUD-USD exchange market.
B.3 USD-CAD Exchange Market

Figure B.3: Returns obtained by the BnH strategy and the average returns obtained by the final system, by the final system without PCA, by the final system without Prioritized Experience Replay (PER) and by the final system without Double Q-learning in the USD-CAD exchange market.
Appendix C

Return Plots

C.1 EUR-USD Exchange Market

Figure C.1: Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market.
C.2 USD-JPY Exchange Market

Figure C.2: Best and average returns obtained by the final system and Buy and Hold returns in the USD-JPY exchange market.
C.3 AUD-USD Exchange Market

Figure C.3: Best and average returns obtained by the final system and Buy and Hold returns in the AUD-USD exchange market.
C.4 GBP-USD Exchange Market

Figure C.4: Best and average returns obtained by the final system and Buy and Hold returns in the GBP-USD exchange market.
C.5 USD-CAD Exchange Market

Figure C.5: Best and average returns obtained by the final system and Buy and Hold returns in the USD-CAD exchange market.
C.6 EUR-USD 2014 Exchange Market

Figure C.6: Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market during 2014, used for performance comparison.
Figure C.7: Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market during 2015, used for performance comparison.
C.8 EUR-USD 2016 Exchange Market

Figure C.8: Best and average returns obtained by the final system and Buy and Hold returns in the EUR-USD exchange market during 2016, used for performance comparison.