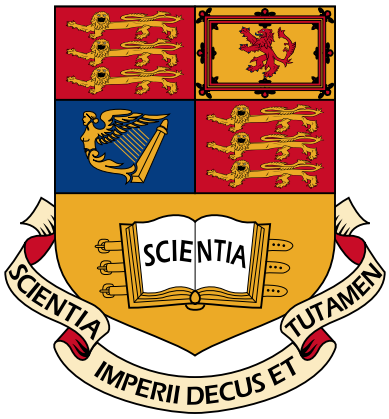

Imperial College London

Department of Computing

An Investigation into the Use of Reinforcement Learning Techniques within the Algorithmic Trading Domain

Author:
James Cumming

Supervisor:
Dr. Dalal Alrajeh

Co-Supervisor:
Dr. Luke Dickens

Submitted in part fulfilment of the requirements for the degree of
Master of Engineering in Joint Mathematics and Computing

June 18, 2015

Abstract

Algorithmic trading has seen a significant surge in interest given the rapidly increasing capabilities of modern day computers. From reducing trading latency by purchasing highly demanded property as near to stock exchanges as possible to continuously honing technical methods over many years, both market participants and academic researchers are constantly seeking novel and successful methods to help them achieve greater success.

We propose a novel reinforcement learning approach to the algorithmic trading problem which we define in terms of the classic reinforcement learning problem framework. Reinforcement learning methods, which aim to optimise an agent's performance within an unknown environment, are very much in active development and cutting edge solutions are regularly being introduced and improved upon. Using state-of-the-art techniques based upon least-squares temporal difference learning, we aim to evaluate the success of our approach in the foreign exchange market and identify its limitations. To achieve this, we design and implement a purpose built algorithmic trading system with support for the reinforcement learning concepts that we require.

Whilst the results that we obtain only show slight profitability, we believe that the framework that we outline carries many merits as a proof of concept and could easily be expanded upon in future work. We discuss some suggestions for these and justify how they could complement our approach. We also feel that the systems we have produced could be adapted and employed by those with interests in both algorithmic trading and reinforcement learning in order to pursue their ideas.

Acknowledgements

I would like to thank the following people for their invaluable contributions to this project.

- Dr Dalal Alrajeh and Dr Luke Dickens, for their unparalleled guidance and enthusiasm throughout the entire duration of the project.
- Mr Kamran Usmani, Mr Zaheer Akbar and the Swiss Finance Corporation, for educating me on the world of foreign exchange and for providing me with the data sets.
- The community of `/r/algotrading`, for their knowledge and assistance.
- Ray, my family and my friends, particularly those within JMC4, for supporting me throughout my time at Imperial, especially during my final year.

Contents

Chapter 1

Introduction

The Foreign Exchange (Forex) market dominates as the largest financial market in the world. With millions of transactions taking place 24 hours a day from Sunday to Friday evening across its decentralized marketplace, the forex market facilitates the exchange of the world's currencies between a multitude of varied participants [1]. Historically, the exchange of currencies took place in open outcry markets where members of financial institutions would use numerous forms of communication, including hand signals and shouting, to agree on an exchange. With the expansion of commercial computer technology into finance in the 1990s however, it has become significantly easier for those involved to communicate and broadcast prices which has greatly encouraged new participants to enter [2]. This has led to much larger trading volumes as well as greater transparency and consistency with regards to pricing which increases the liquidity in the market and leads to a reduction in the transactions costs. Nowadays, it is simple for an independent trader to set-up an online forex brokerage account and begin trading from their own personal computer.

With this increase in market volume and the widespread availability of computer technology in finance, developing tailored algorithms that trade based on programmed logic and time series data has become increasingly popular as prices are easily available digitally from a variety of online sources. This paper therefore investigates and evaluates the use of reinforcement learning techniques within the algorithmic trading domain.

1.1 Motivation

With prices being much more available, the time between each price update has decreased significantly, often occurring within fractions of a second. This has made it near impossible for a human to respond and act on these rapidly changing conditions whereas a machine can do so with relative ease [3]. Additionally, computers have the advantage of taking the emotional aspect out of trading as even the most experienced financial traders may find themselves operating sub-optimally due to their emotions. It also decreases the chances of human error as was famously the case when a Japanese trader at Mizuho Bank attempted to sell 610,000 shares at ¥1 instead of the intended 1 share at ¥610,000 [4]. The growing

popularity of algorithmic trading has been indicated by the trends of major financial institutions where they have replaced many of their traditional human traders with electronic counterparts. Algorithms are developed by those experienced in financial markets and then implemented in computing languages. Some may be as simple as taking a position based on a set price level whilst others may examine millions of points in a past series of data to predict future movements and execute orders based on those predictions. Developing these trading algorithms involves a significant amount of trial and error and they are often incrementally improved and adapted to any issues that they encounter.

Similar to the expansion in forex activity and financial technology, machine learning and the various disciplines that fall under it have seen a recent surge in interest. From the use of artificial neural networks that attempt to replicate the structure of the brain in pattern recognition, popularized by Geoffrey Hinton [5], to more recent methods such as reinforcement learning [6] and Bayesian probability methods, machine learning techniques have been explored in a large number of current applications. The rise of the internet has aided this by providing researchers with larger and more varied datasets that these algorithms can be trained on, improving their effectiveness and encouraging their further research. Each method has generated its own advocates and comes with its unique advantages and disadvantages that make it more suited to certain applications. The field of pattern recognition is often considered to be analogous to modern machine learning and given the requirement for accurate prediction and trend recognition methods in algorithmic trading, machine learning has proven to be a profitable technique.

1.2 Objectives

The scope of this project is to investigate the effectiveness of reinforcement learning techniques within the domain of algorithmic trading. We will examine the existing use of reinforcement learning in algorithmic trading and then use this established information to experiment with current methods and a novel approach. This investigation will not evaluate an existing trading algorithm as this has most likely been done in detail by its creators, nor will it examine high frequency trading techniques as this would shift the focus away from pattern recognition and trend prediction. We wish to evaluate how different reinforcement learning techniques are suited to the problem, how the various parameters involved affect the outcome and whether this leads to a profitable strategy that could be used in production. We also hope to provide a stable framework which can be used for further investigations into the applications of reinforcement learning, not only in the financial domain, but also on general datasets as well. This will include the necessary components of an algorithmic trading system including backtesting functionality and support for general models.

1.3 Possible Challenges

As discussed, algorithmic trading has seen a surge in popularity over the past decade due to the increase in availability and performance of modern computer systems. There are numerous existing systems and techniques that have been proven, through their usage,

to provide considerable return rates. However, when developing new ideas from scratch, methods of trial and error as well as market intuition and experience are often required. These techniques are then continuously developed and refined over time to adapt to any performance issues that arise during their usage. Considering we are proposing a novel algorithmic trading strategy using reinforcement learning techniques that have not previously been explored within this domain before, there are no guarantees of profitable success, nor are there explicit recipes to guide us through any problems that we will face.

Similarly, several of the reinforcement learning techniques that we will be using have only been recently established and finding optimal choices for many of the parameters is therefore still an open problem. We aim to experiment with different parameter choices, investigate how these affect the algorithm and reason about why this may be. Unfortunately, as the number of parameters and possible values increases, the number of combinations of parameter choices grows significantly faster and it becomes infeasible to investigate all possible combinations. For example, with 5 parameters that each take 20 possible values, we would have $3,200,000$ combinations. Assuming each test takes only 1 second to complete, it would take over a month to completely test each possible combination! Additionally, as we will see when looking at the least-squares temporal difference techniques, many of the parameters are continuous and we will therefore need to discretise them in order to test them.

1.4 Contributions

We outline the key contributions to the reinforcement learning and algorithmic trading fields that we hope to have provided through this investigation.

1. We propose a viable reinforcement learning framework for forex algorithmic trading that clearly defines the state space, action space and reward structure for the problem. It provides details of a concrete implementation of one possible design choice which we use to evaluate the reinforcement learning algorithms with. However, it is designed to be extendable so that more information can be included within the state definitions, the representations of a state can be changed to suit the needs of a particular problem and the objective function with the associated reward signal can also be altered.

2. We implement a backtesting system in Python for evaluating algorithmic trading models. It provides detailed feedback to the user on the details of the positions taken, the financial profits obtained from these and other relevant metrics such as return rates. It should also be generic enough so that a model only needs to define a method that receives the current price and no other restrictions are imposed by the system. Finally, whilst we will be evaluating its performance on forex candlestick data, it is designed such that it can accept any form of time-series data.

3. We investigate and evaluate the effectiveness of reinforcement learning techniques, specifically least-squares temporal difference learning methods, on the reinforcement learning problem defined above using our backtesting system. We identify the strengths and weaknesses associated with our approach and how these affect the performance

of our algorithm. We also suggest future modifications that could be made as well as relevant techniques that could be explored and explain the differences that these alternative approaches could offer.

1.5 Outline of Contents

We conclude this introduction with a brief outline of the contents of our paper.

- Chapter 2 will aim to introduce the relevant background material covering finance and the forex market, the relevant work in algorithmic trading system, and the necessary reinforcement learning concepts that we will make use of.
- In chapter 3, we will propose our novel approach to extending the reinforcement learning framework into the domain of algorithmic trading, justifying our design choices for the system. We will also introduce the relevant least-squares temporal difference learning material.
- Chapter 4 provides an evaluation of our approach using the least-squares temporal difference learning methods, as well as documenting our design choices for the proposed systems. We also outline some of the problems that we faced and discuss available solutions to these.
- Chapter 5 concludes the project with discussions regarding the success of our objectives and a summary of our achievements. We also present some opportunities for future work and suggest how these could be used to extend the contributions presented in our project.

Chapter 2

Background Material

2.1 Financial Markets and Foreign Exchange

The foreign exchange market will be the primary domain of this investigation and we therefore provide relevant background on the key concepts that we will use.

2.1.1 Products

The products traded on the foreign exchange market are the world's currencies. The majority of transactions involve the 4 major currency pairs that set the United States Dollar (USD) against the Euro (EUR), Great British Pound (GBP), Swiss Franc (CHF) and Japanese Yen (JPY) currencies. There are also opportunities involving the USD against relatively minor currencies such as the Australian and New Zealand dollar. Additionally, cross currency trading also occurs which pairs non USD currencies against one another such as the GBP and JPY. Many of the pairs have been given colloquial nicknames over time by traders including the *Cable*[1] for the GBP/USD pair and the *Barney* for the USD/RUB (Russian Ruble) pair. Traders can simply buy these exchanges outright, known as spots, or they can use them to sell more complex derivative instruments such as futures and opens which may appeal to different participants [7].

The opportunity to trade in more exotic and emerging currencies, such as the Thai Baht (THB), is also available, however there may be restrictions in place that limit the liquidity which are not present in more well established currency markets that can therefore act as barriers to entry. For example, the Chinese Renminbi has been historically strictly controlled by the government and there has been little opportunity for international trade. Nowadays, they are starting to open up their currency internationally to encourage international participants [8].

[1] A reference to the original Transatlantic cable that transmitted this exchange rate between the UK and US

2.1.2 Market Participants

As with any financial marketplace, there are varied participants, each with their own motivations. It is important to understand these different parties and the types of transactions that they tend to execute as this will affect the market.

Commercial Transactions

It should be remembered that the financial sector of an economy is highly important but should not be considered its driving force; actual economic activity is what grows an economy [9]. Those involved in global commercial transactions form a key part of the foreign exchange market. This can range from individuals requiring foreign currency for their holidays to businesses executing financial transactions between economies that operate with different currencies. Its primary characteristic is that the foreign exchange transaction is used as part of some economic activity [10]. If an airline based in Britain wishes to buy fuel from America then it needs to execute some foreign exchange transaction to achieve this. They start with £10 and $0 and need to purchase $15 dollars worth of fuel. Initially, they are *long* GBP and *short* USD. Suppose the current rate at which they can sell GBP for USD is 1.5. They then would sell £10 to buy $15, giving them a position where they are short £10 and long $15. This would allow them to purchase their fuel. Whilst this example may seem simple and commonplace, it represents a very large number of transactions that take place daily across the forex market.

Hedgers

Hedging is a key usage of any financial marketplace that both individual investors and large corporations have a need for. Hedging refers to the execution of a financial transaction in an attempt to minimize the risk of an existing position in the market [7]. Speculative traders use hedging to manage their risk so rather than aiming for large profits which also come with a large risk of losses, they hedge their trades so that they receive a smaller profit but at a much lower risk. An example of forex hedging would be a business that needs to make a future payment in a foreign currency. At some point, it will need to purchase this foreign currency with their own domestic currency. If they are concerned that the exchange rates for this pair will move unfavourably against them, or they just wish to fix the price now, they can hedge this risk. They can buy a *futures* contract that gives them the right and obligation to buy the foreign currency with their domestic currency at an agreed rate at some time in the future. They could also use an *options* contract which gives them the right but not the obligation to purchase at that rate meaning they would be able to benefit if the price moved favourably as they could then purchase the currency at the current market price.

Speculators

Speculative trades make up the majority of the volume that flows through the forex market. Speculation refers to traders taking positions purely based on the expectation of a favourable price movement on the currency pair. When hedgers attempt to minimise their risk, it is usually speculators who take on this risk in an attempt to increase their profits [7]. These expectations in price movements can be over various time periods from seconds to days. Fundamentally, it is interest rates that drive the price of a currency within an economy as this represents the price of an economy's money. Nowadays, with so many market participants, any news related to the economy or possible changes in interest rates can cause significant changes in the demand and supply for a currency and this will have an effect on the price. Factors such as the overall economic outlook, the performance of major companies within an economy or general political policies can cause major changes in exchange rates. For example, if the European Central Bank announced that interest rates would rise sooner than expected then it would generate more demand for the EUR, increasing the price and so the currency would appreciate against others. Additionally, the orders of other participants in the market will generate supply and demand signals that alter the rate.

It is important to understand the importance of foreign exchange when executing international transactions as it may have an unforeseen effect on one's overall activity. Say the interest rate in Thailand was 10% compared with 2% in the UK. Someone might choose to purchase THB with GBP, leave it in a Thai bank account to accumulate more interest and then purchase back more GBP with THB. However, if the THB has depreciated against the GBP, by more than 10% in this example, then you will not be able to buy back as much GBP as you had initially so you will have lost money even though you gained interest in Thailand. Exchange rates can move fast and sometimes unexpectedly and so it is important to consider this when using forex in a transaction, for example, in late 2014 the Russian Ruble suffered a huge 11% decrease in one trading day due to falling oil prices.

2.1.3 Positions

In financial markets, when a participant executes a buy or sell transaction, they are said to take a *position* that describes which price movements will be beneficial to them and which will not. If someone buys an asset, they do so with the desire for it to increase in value so that they can later sell it for a profit. They are said to hold a *long* position in the asset. Alternatively, if someone sells an asset, they hope for a decrease in the price so that they can buy it back cheaper at a later date. This is referred to as a *short* position. Short positions are often considered to be more risky as there is no upper bound on the amount one can lose if the price increases whereas with a long position, the most once can lose is the amount that they have invested. If the participant has no interest in the price movement they are said to hold *no* position. When one enters a transaction they are said to *open* a position and at a later date when then they choose to end their position, they are said to *close* it. The success of a position is determined by the *Profit and Loss (PnL)* produced upon closing that position.

Definition (Profit and Loss)**.** The Profit and Loss of a position is given by the following formulas, depending on whether the position is *long* or *short*.

$$\text{PnL}_{\text{long}} = \text{value}_{\text{closing}} - \text{value}_{\text{opening}}$$

$$\text{PnL}_{\text{short}} = \text{value}_{\text{opening}} - \text{value}_{\text{closing}}$$

From the above definition we can see that in a long position, the participant makes a profit if the value increases and a loss if it decreases; the converse is true for a short position. We often refer to the *Unrealised PnL* as the value of positions that are still open.

Definition (Unrealised Profit and Loss)**.** The Unrealised Profit and Loss of a position is given by the following formulas, depending on whether the position is *long* or *short*.

$$\text{Unrealised PnL}_{\text{long}} = \text{value}_{\text{current}} - \text{value}_{\text{opening}}$$

$$\text{Unrealised PnL}_{\text{short}} = \text{value}_{\text{opening}} - \text{value}_{\text{current}}$$

2.1.4 Pips, Bid, Offer and Spread

Pips (Percentage in points) are used to measure the change in an exchange rate. Typically, the major currency pairs are priced to 4 decimal places and a pip is therefore 0.0001; the JPY is an exception which is 0.001 due to the Yen being fairly weak compared to the other currencies [11]. As financial technology has advanced, prices have become more transparent, the number of decimal places used has increased and many platforms nowadays offer rates to 5 decimal places, providing more flexibility and opportunities to market participants. It is common for traders to refer to short term profits in terms of pips.

The term *bid* and *ask* are used to signal how much you can immediately sell something for, the bid price, and how much you can buy something for, the ask price. The difference between these is known as the bid-ask *spread* and represents the liquidity in the market and the associated cost for a transaction [11]. If you bought something and then immediately sold it, you would not regain all of your original value back due to the bid price being slightly lower than the ask price and the spread between them acting as an implied transaction cost. We illustrate this in figure 2.1. As trading volumes have increased and prices have become transparent, this spread has reduced significantly indicating that the market has become much more liquid. This is beneficial for market participants as the cost of a transaction has been reduced but harmful to brokers and market makers who made a significant income based on the spread.

GBP/USD	
Bid Price	**Ask Price**
$1.56\,39^{7}$	$1.56\,41^{8}$

Figure 2.1: A possible example of the information displayed to a trader for the GBP/USD rate. In this example, the bid-ask spread is $1.56397 - 1.56418 = 0.00021 = 2.1$pips

2.1.5 Market Evolution

The advent of financial technology has significantly changed the way the forex market operates. Prior to the widespread adoption of computer technology in the 1990s, trading would take place on trading floors of financial centres with orders being shouted across the hall in open outcry markets [2], similar to those shown in figure 2.2.

Figure 2.2: An example of participants in an open outcry markets [12].

Nowadays, most transactions are executed electronically at the click of a button and this has greatly reduced the barriers of entry into the market. This has been a key factor in the large increase in transaction volume and liquidity within recent years. It is now very easy for anyone to set up a forex brokerage account to trade with and the use of margin and leverage allow them to trade large amounts with a relatively small input of capital.

Figure 2.3: Modern day traders monitoring the performance of trading algorithms [13].

Additionally, as this investigation aims to explore, the use of algorithmic trading has become much more widespread and applications have been developed to facilitate this including the Financial Information Exchange (FIX) API that allows computer systems to connect to a brokerage account and execute transactions[2]. Prior to this, traders themselves would use technical analysis methods to manually calculate the levels at which they should open or close a position whereas now, computers can perform the same mathematical calculations far more efficiently and react accordingly in a much shorter time period.

2.1.6 Order Types

There are two main order types that can be executed when taking a position; *Take Profit* and *Stop Loss*. In a take profit order, you set a limit at what amount of profit you want to make and then close the position when you reach that to secure the profit. In a stop loss order you set a limit on the loss level that is acceptable to you. When you reach this limit, you close the position to limit your losses. We illustrate how these levels could be set for a given entry position for both long and short positions in figure 2.4.

[2]FIX API - `http://bostontechnologies.com/blog/what-is-fix-api`

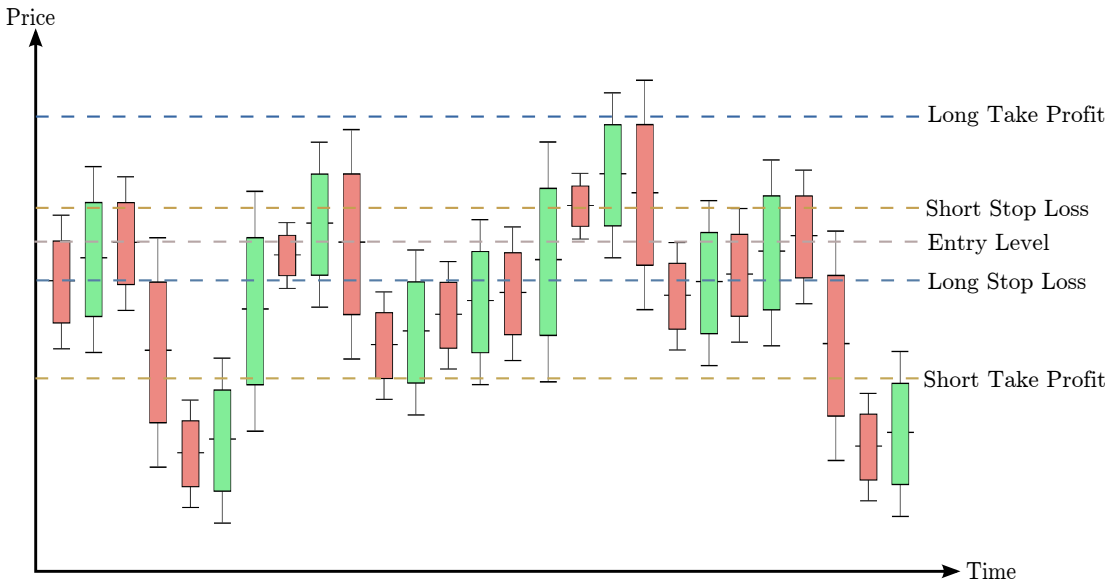

Figure 2.4: A candlestick chart demonstrating possible choices for take profit and stop loss levels.

If the profit on a take profit order is reached, the order is set to be *filled*. If the loss on a stop loss order is reached, the order is set to be *stopped*. If the order is closed for any other reason it is simply marked as closed. These can be combined to form a One-Cancels-The-Other-Order so that a take profit and stop loss order are placed simultaneously and if one is executed, the other is cancelled.

2.1.7 Leverage and Margin

When trading on the forex market, it can be useful to execute transactions based on more capital than one actually has in order to achieve greater possible profits. Of course, with this opportunity for larger profits, there comes the chance of larger losses as well. To do this, we use the concept of *margin*. If a participant is at a 2% margin levels then they have access to $\frac{100}{2} = 50$ times the money that they put in. This means that a 1% positive price movement would yield 50% of their input capital as profit. However, a 2% negative price movement would ruin their capital. This can be very useful in an algorithmic trading system that is able to accurately predict small price movements over a short period as even though the price movements may be small, if the algorithm is fairly accurate then appropriate margin levels can be used to magnify the potential profits.

2.2 Algorithmic Trading

As discussed above, algorithmic trading models are becoming an ever more popular component of not only financial institutions but also individuals with an interest in programming and financial engineering. Models can include simple yet effective methods such as setting a take profit and stop loss level based on the current price and the range of the previous days. Alternatively, more complex models such as those involving modern machine learning techniques can be used, however, greater complexity does not necessarily translate into a more accurate or profitable model.

2.2.1 Technical Analysis

Technical analysis underpins many trading strategies employed in the forex trading markets. As forex is a 24 hour market, there is plenty of information which one can use to predict future price movements and this is the underlying concept of technical analysis [1]. When looking at historical data, a variety of charts can be used, each with their own advantages in terms of detail and range covered; traders often remain loyal to one chart type. Candlesticks are a commonly used chart that represent the open, high, low and close prices over a given time period for a currency pair. Their main advantage is that one can easily notice a trend and each candlestick is able to convey four useful points of information for the given period [14]. Figure 2.5 demonstrates an example candlestick chart plotting the time epoch against price as well as explaining the key features of candlesticks.

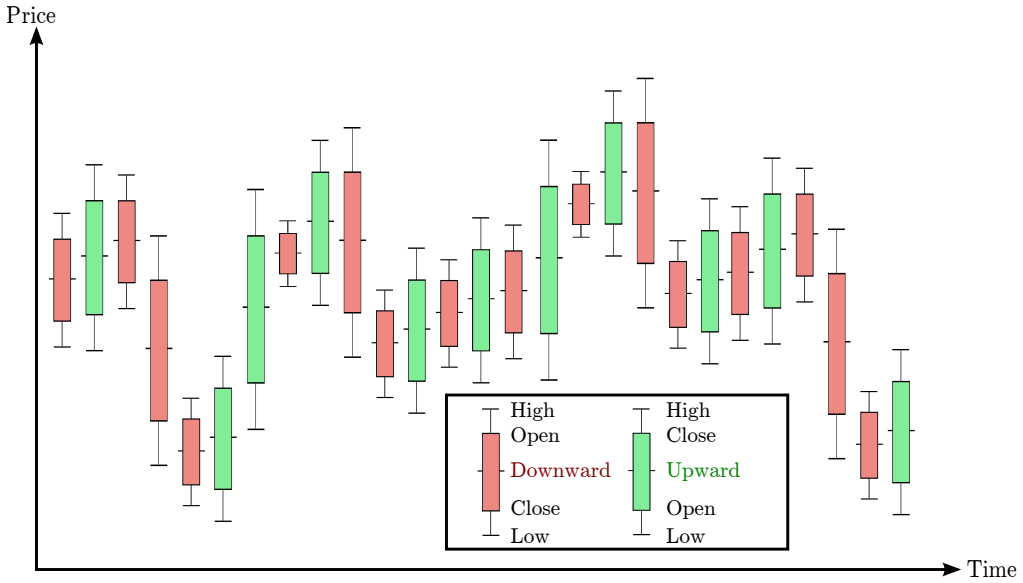

Figure 2.5: A candlestick chart explaining the features of the candlesticks.

Originating from the work of Charles Dow, the inventor of the Dow Jones Industrial Average, *Dow theory* is a form of technical analysis that is used to determine trends in the market. Whilst developed over a century ago, it is still very much relevant to the current

markets and continues to be used today, with the Dow Jones being one of the most popular indicators for economic performance. Dow theory is formed of six key tenets that are used by traders to predict future price movements given the current market and historical data [15]. We outline the tenets briefly below as it provides some reasoning behind the trends one can expect to see in the market.

1. The averages discounts everything - Whilst the markets may not be able to predict all events that will affect it, the market's averages based on the price levels will reflect this and so they do not need to be treated separately.

2. Markets have three major trends - These are based on the time scale with a main movement being defined over the last few years, the medium movement over recent days and weeks and the short term movement over the previous day and hours.

3. Major trends have three phases - The first is the accumulation phase where well informed investors transact against the market. The second is the public participation phase where the public becomes involved in the market and expresses interest. The final phase is distribution where the investors transact with the general public.

4. The averages must confirm each other - Charles Dow used two averages, the industrial average and the rail average, each made up of a weighted average of the major companies in each industry. For a strong market signal to be credible, it must be reflected in each average.

5. Volume confirms the trend - As a price moves, trading volumes should increase in that direction as participants react to the market and engage in transaction. This can be used to confirm the public knowledge of a trend.

6. A trend is in effect until a definite signal that it has been reversed is received - Once a trend is in motion, one can assume it will continue until they receive a signal that opposes that trend.

2.2.2 Development Process and Issues

Typically, algorithms are created based on an initial conjecture about the market that usually comes from practice or intuition. An algorithm is then implemented that trades according to the suggested logic and this is thoroughly backtested across varying market conditions and data sets. Similar to the testing and training of machine learning methods, avoiding over-fitting of the testing data is vital for an algorithm to be successful. Due to this, backtesting is usually followed by simulated trading using real time data but using imaginary funds. If this also proves to be successful then the algorithm will most likely be deployed to trade and will be consistently monitored to evaluate its success. If its performance suffers or there is a change in the market such that it is no longer valid, the system can always be halted, adjusted and then re-deployed allowing the creators to incrementally update and tune the system [3].

There are perceived dangers associated with algorithmic trading, primarily due to its lack of human intervention and reliance on technology. If there is a failure of the systems,

then the algorithm may not be able to recover its opening positions and this could lead to large losses. Additionally, as with any programming task, there may be edge cases that programmers have overlooked which lead to undesirable functionality or failure [3]. Due to this, it is necessary to put in place acceptable loss levels for an algorithm so that even if its logic fails, the damage this causes can be limited to a reasonable level. Furthermore, concerns have been raised regarding the effects of algorithmic trading as a whole. In light of the 2010 Flash Crash that lead to a momentary 7% dip in the Dow Jones index before recovering later that day, many have called for greater regulation and market reform with regards to algorithmic trading. It should be noted that the trading methods deemed to cause the flash crash were high frequency trading methods where participants place a large number of orders in order to manipulate the price and then cancel them so that they can take advantage of the expected price movement. High frequency trading is a form of algorithmic trading but it is far from the only class of algorithms and it is therefore unreasonable to blame the entire domain of algorithmic trading for these kinds of incidents [16].

2.2.3 Related Work

We outline several developments in machine learning algorithmic trading strategies in the sections below to establish an understanding of the existing related work.

Adaptive Reinforcement Learning

An investigation by M. A. H Dempster and V. Leemans introduced the use of Adaptive Reinforcement Learning in forex trading systems [17]. Their original algorithm that they base their ideas on was devised by Moody and Saffell and uses recurrent reinforcement learning techniques to output the preferred position for the algorithm to take based on the changes in the historical time series values [18]. Dempster and Leemans took this work further by introducing a *risk management layer* and *dynamic optimisation layer* on top of the algorithm that allows the user to input their risk preferences and parameters to adjust the behaviour of the machine learning agent as shown in figure 2.6.

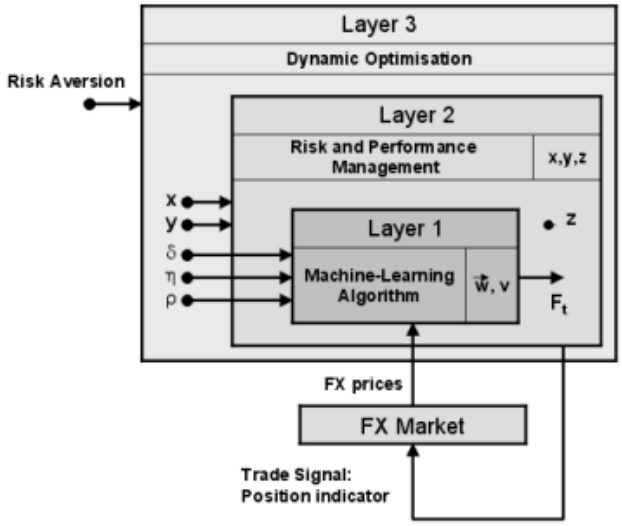

Figure 2.6: The design used for a 3 layer RRL trading system

This novel approach is innovative as it allows the creator to adjust the algorithm to match their preferences as well as to the current market conditions. Whilst allowing the user this ability to adjust the algorithm, it also abstracts them from the parameter details involved which make it simple to use and reason about from a high level.

Recurrent Reinforcement Learning

Xin Du, Jinjian Zhai and Koupin Lv compared the results of classic reinforcement Q-learning methods and compared them to recurrent reinforcement learning techniques, similar to the ones used by Moody and Saffell [19]. Their value functions included measuring the cumulative profit as well as experimenting with the Sharpe ratio and its derivatives. They concluded that the RRL methods produced more stable results than those using Q-learning methods and were also less complex computationally. They also found that a good choice of objective function was vital in order to achieve optimal performance.

Note. The Sharpe ratio refers to a metric used to evaluate the performance of a financial investment with respect to the risk involved. It is calculated using the expected return rate of the investment μ, the risk free rate r and the volatility of this return σ.

$$\text{Sharpe} = \frac{\mu - r}{\sigma}$$

Support Vector Machines

Additionally, an investigation carried out by Shunrong Shen, Haomiao Jiang and Tongda Zhang explored several machine learning algorithms in stock market forecasting, in particular those which used support vector machine (SVM) algorithms [20]. Whilst their in-

vestigation was focused on the stock market rather than forex, the principles they used were based on time series data and could therefore theoretically be applied to the forex market as well. Whilst we will be focusing more on reinforcement learning related techniques, their work demonstrates the variety of machine learning techniques currently being experimented within the algorithmic trading domain. Their results were more focused on prediction accuracy rather than profitability whereas the reinforcement learning methods aimed to optimise an objective value function based on some return performance metric.

2.3 Reinforcement Learning

Reinforcement learning (RL) is an actively researched branch of machine learning that differs from conventional methods as rather than relying on an agent being explicitly told what to do and which actions should be taken, the agent is expected to experience its environment, evaluate what was and what was not successful and then decide for itself what the optimal choices should be. The agent must choose autonomously which actions to take in order to maximise not only the immediate reward, but also the expected subsequent rewards that may be obtained from taking that action. These concepts of exploration, recognising the importance of delayed rewards and learning from both successes and mistakes are the crux of reinforcement learning methods and differentiate them from other machine learning techniques. In 1998, Richard S. Sutton and Andrew G. Barto wrote *Reinforcement Learning: An Introduction* which provides a comprehensive introduction to the theory, history and applications of current RL methods [6]. We will often refer to their material throughout this background section.

Reinforcement learning differs from supervised learning problems as examples of good and bad behaviour are not explicitly provided to the agent. This makes it much more suitable for interactive problems as it can often be difficult to suitably describe optimal behaviour for these types of problems and is far easier to allow the agent to experience the environment. Reinforcement learning represents a general problem that can then be solved through a variety of methods and any such problem is known as a reinforcement learning problem.

2.3.1 The Reinforcement Learning Problem

The reinforcement learning problem defines a framework from which we can construct a variety of problems that can be solved through RL methods. We define the learner to be the *agent* and this will make the decisions of which actions to take. Upon executing an action, the agent alters the state of the world around it, which we refer to as the *environment* and the environment responds by advancing to this new state and generating a reward for the agent in the process. Rewards are real valued numbers and the agent seeks to maximise its expected cumulative rewards over its lifespan [21].

Note. A negative *reward* is equivalent to a literal cost whilst a positive *reward* represents a reward in the traditional sense. Many literature sources define problems with respect to minimising costs rather than maximising rewards but we will stick to aiming to maximise rewards.

The distinction between the *agent* and its *environment* is not always clear, for example if we consider the agent to be an animal, then the environment should include the body of the animal as this will produce pain and pleasure stimuli which would act as rewards and we consider rewards to be external to the agent. The agent in this case would therefore be some abstract decision making engine within the animal that can sense rewards produced by the body and perform actions that alter the state of the body. A general rule that is followed is that all features of the world that cannot be *arbitrarily* changed by the agent should be considered part of the environment [6]. We can formally define the above situation as

follows.

Definition (The Reinforcement Learning Problem)**.** At each discrete time step $t = 0, 1, 2, 3, ...$, the agent receives information about the environment in the form of a state $s_t \in \mathcal{S}$ where $\mathcal{S}$ represents the space of available states. Given this state information and its previous experience, the agent selects an action $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ represents the actions available to the agent given the state s_t. Upon executing this action, the agent finds itself in a new state s_{t+1} and receives a reward $r_{t+1} \in \mathcal{R}$ where $\mathcal{R}$ represents the set of numerical rewards available. We summarise the reinforcement learning problem in figure 2.7.

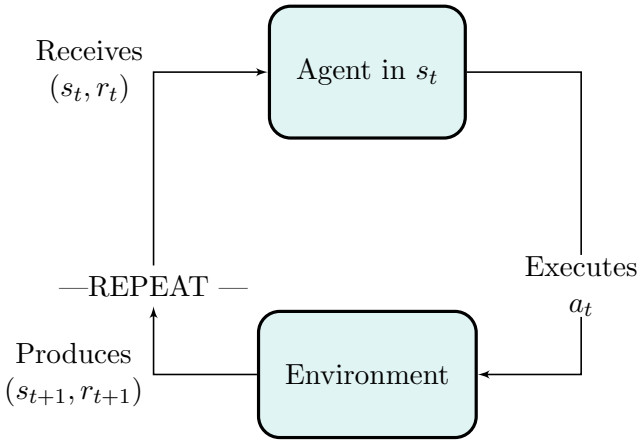

Figure 2.7: The Reinforcement Learning Problem. Consider the clockwise cycle starting from repeat.

In order to make the decision of which action should be take from a particular state, the agent implements a mapping from $(state, action)$ pairs to the probabilities of selecting that action at each time step t. This is referred to as the agent's *policy* and is denoted $\pi_t(s, a)$. The aim of reinforcement learning algorithms is to specify how the agent should evaluate and update its policy given its experiences.

Rewards and Returns

When designing the rewards that an agent will receive, one needs to make sure that they accurately reflect the aims of the overall objective function rather than various sub-goals. An example that Sutton and Barto give is that of an agent in a Chess system, designed with the intention of winning the game. A reasonable way of defining the rewards would be $+1$ for a win, -1 for a loss and 0 for a tie. This will cause the agent to assign value to states that led it to a victory. If one were to also assign a reward of $+1$ for each opponent piece captured, the agent might learn to focus purely on capturing enemy pieces rather than aiming to win the game and therefore acting against its intended behaviour. It is important that rewards are assigned based on *what* should be achieved and not on *how* it should be achieved [6].

As described earlier, an agent seeks to maximise its expected cumulative rewards. We

therefore define the *Return* to be the sum of rewards obtained from a given time period t to some terminal time period T as follows:

$$R_t = \sum_{k=0}^{T} r_{t+k+1} \tag{2.1}$$

This formula is well defined for tasks where there are definite terminal states that an agent is guaranteed to eventually reach, implying that $T < \infty$. We call these tasks *episodic* as they start in a valid starting state, select actions that transition them through a finite number of time epochs and then reach a terminal state. Once the agent enters a terminal state, we assume that it cannot leave and the only valid transition is to itself and this should generate a reward of zero. A complete sequence of actions from a starting state to a terminal state is defined to be an *episode* [6].

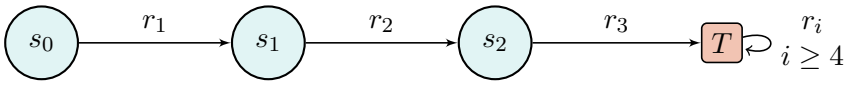

Figure 2.8: An example transition diagram for an episodic task

However, many tasks cannot be constructed as episodic and could potentially continue indefinitely. We refer to these as *continuing* tasks and in this case, $T = \infty$. The sum in (2.1) could in many cases diverge to infinity and provide little value to the agent. To counter this, we introduce the concept of *discounting* which assigns a greater value to immediate rewards than it does to future rewards. Formally, we choose a parameter $\gamma \in [0, 1]$ referred to as the *discount rate* and define the return from time t as follows:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

If $\gamma = 0$ then the agent only takes into account the immediate reward and we consider it to be short sighted or *myopic* whereas as $\gamma \to 1$, the agent becomes more far-sighted and values future rewards and immediate rewards more equally. Crucially, as long as $\gamma < 1$ and the rewards are bounded, i.e. $|r_k| < \infty : \forall k > t$, then this sum converges to a finite value which can be used by the agent to calculate the return. We can formulate (2.1) and (2.2) together in the following:

Definition (Return)**.** Given a terminating time T, a discount factor γ and a sequence of rewards r_i, we define the return from time t to be:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{2.3}$$

The only condition we require is that we cannot have both $T = \infty$ and $\gamma = 1$ used in the sum.

States and the Markov Property

The definition of a state in a reinforcement learning problem is crucial to the success of the agent. 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. On the other hand, it is unrealistic and unhelpful to define a state such that it contains more information than the agent should be expected to know at an expected time point. This is often summarised to say that an agent is not expected to know everything important to it at all time points but it is expected to remember anything important that it has already experienced [6].

Mathematically, this is equivalent to the Markov property which is commonly used in stochastic processes and forms the underlying property of Markov chain theory. Informally, it states that conditional on the present value, the future value of the process is independent of the past values. For the reinforcement learning problem, the reward and state signals that we received at $t+1$ should depend only on the state and action take at time t rather than the complete history of states and actions.

Proposition (Markov Property)**.** *Given any sequence of actions a_i, state signals s_i and reward signals r_i, we say that a reinforcement learning problem satisfies the Markov property if and only if* (2.4) *holds for all possible values of s' and r.*

$$\mathbb{P}(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, ..., r_1, s_0, a_0) = \mathbb{P}(s_{t+1} = s', r_{t+1} = r | s_t, a_t) \tag{2.4}$$

The Markov property is important as it allows us to calculate the dynamics of the transition to the next state purely based on the current state and action. This can be generalised to allowing us to compute the n-step future dynamics based solely on the agents current state and action. Many problems do not fully satisfy the Markov property, however it is often valuable to consider them as Markov as we can still apply some of the RL algorithms to them and obtain meaningful results [6].

Markov Decision Processes

If we define a problem such that it follows the reinforcement learning problem framework and satisfies the Markov property as described above then we refer to it as a *Markov Decision Process (MDP)*. For an MDP defined on a finite state and action space, we define the two following quantities which provide the crucial dynamics.

Definition (Transition Probabilities)**.** Given an initial state s_t, an action a_t and a subsequent state s_{t+1}, the transition probabilities of a finite MDP give the probability of transitioning to the subsequent state given the initial state and action.

$$\mathcal{P}^a_{ss'} = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$$

Definition (Expected Reward)**.** Given an initial state s_t, an action a_t and a subsequent state s_{t+1}, the expected reward is the expectation of the reward signal given the subsequent

state, the initial state and the action that occurred.

$$\mathcal{R}^a_{ss'} = E[r_{t+1}|s_{t+1} = s', s_t = s, a_t = a]$$

Note. Often these transition and reward dynamics are unknown to the user but the state and action space are available. The availability of knowledge for the model often affects the choice of RL technique to use.

Value Functions

To evaluate and improve policies, most reinforcement learning methods use *value functions* which assign values to each state given a specific policy. It is also possible to use value functions which assign values to each state-action pair.

Definition (State Value Function)**.** Given a state s and time t, the value of the state at t under a fixed policy π is the expected return of starting in the given state and then following the policy.

$$V^\pi(s) = E_\pi[R_t|s_t = s] = E_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s]$$

Definition (Action Value Function)**.** Given a state s and action a at time t, the value of a state-action pair at t under a fixed policy π is the expected return of starting in the given state, taking the given action and then following the policy.

$$Q^\pi(s, a) = E_\pi[R_t|s_t = s, a_t = a] = E_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s, a_t = a]$$

Note. $E_\pi[.]$ denotes the expectation given that the agent continues to follow the policy π.

For fairly small scale problems, V^π and Q^π can be directly computed using the transition probabilities and expected reward dynamics if a full specification of the model is provided. Alternatively, they can be estimated based upon experience. Maintaining an average of the returns that followed from each state and action can be used to build up these estimates and as the number of visits and executions of the states and actions respectively approaches infinity, these will converge to the values. This is known as *Monte Carlo Estimation* and uses random sampling of the MDP.

For larger problems, or more importantly continuous state and action space problems, it is unrealistic that each state can be visited enough times to produce a meaningful average and impractical to store the estimates. We therefore rely on *function approximations* of the Q and V functions through parameter updating.

An important result for value functions is that they can be defined recursively in terms of the values of *successor states* as follows.

Proposition (Bellman Equation)**.** *Given a state s, a fixed policy π, a discount factor γ and a value function $V^\pi(x)$, we have the following recursive relationship of the value function.*

$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

For a derivation of this see Sutton and Barto 3.7. This can also be extended to relate $Q(s,a)$ value functions.

This allows us to compute the value of a state as the expected reward plus the discounted value of the state we will transition into upon receiving that reward. This is often represented through *backup diagrams* which illustrates the states and actions used to estimate the value for a particular state.

Optimal Policies and Value Functions

The agent's goal is to choose actions such that it maximises the expected cumulative reward that it will receive. This implies that there exists some policy which maximises this expected cumulative reward i.e. $\exists \pi^*$ s.t. $V^{\pi^*}(s) \geq V^\pi(s) : \forall \pi, s \in \mathcal{S}$ which is known as the *optimal policy*. Computing this optimal policy is central to reinforcement learning methods.

2.3.2 Solving the Problem

In can be shown that for a system with N states and known transition and reward dynamics, the optimal policy can be computed using a system of NxN linear equations with time complexity $O(N^2)$. For small N this may be suitable but for larger systems, this complexity coupled with the *curse of dimensionality* means that this method is often becomes infeasible.

Note. The curse of dimensionality refers to how as the number of variable in a state grows, the number of states required to represent the state space often grows exponentially.

Often, the dynamics of the reinforcement learning problem are not known and there are a large number of possible states. We are therefore unable to use the above approach and must rely on alternative methods as introduced below.

Generalised Policy Iteration

To compute an optimal policy for a given RL problem, most methods use *Generalised Policy Iteration (GPI)* which iteratively performs the two following procedures until sufficient convergence is achieved [6]. This technique is illustrated within figure 2.9.

1. **Policy Evaluation** - We evaluate the value function for all states with respect to the current policy
2. **Policy Improvement** - Using these estimated value functions, we create a policy that is *greedy* with respect to these value functions

Note. A policy is *greedy* with respect to a value function if it deterministically chooses the action that maximises the value function and never chooses any other actions.

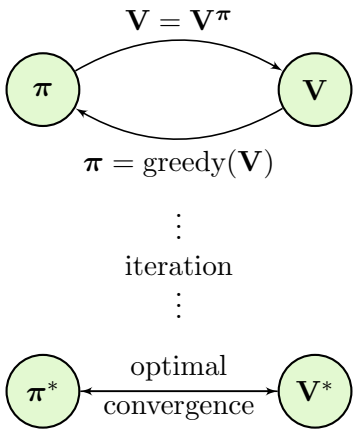

Figure 2.9: The Generalised Policy Iteration process

Once we no longer see changes in the value functions or policy between iterations, it has been shown that the converged policy is in fact the *optimal policy*. Thus, GPI is at the centre of most RL algorithms and provides an effective skeleton process for finding an optimal policy. RL methods then define how they perform policy evaluation and iteration which is what differentiates them. We will briefly discuss dynamic programming, Monte Carlo and temporal difference methods.

Dynamic Programming

If one knows the dynamics of the environment, once can use dynamic programming (DP) methods to determine an optimal policy. We recursively updates the value functions for each state by starting at the terminal state and use the Bellman equation to update the values of all states based on the expectation of their reward and the estimated value of the subsequent state. Using the estimate of the value of the subsequent state to compute the estimate of the current state is a process known as *bootstrapping*. Once we have sufficient convergence in the value functions, we improve our policy by making it greedy with respect to the value functions and then re-evaluate the state values. This converges to the optimal policy as in GPI.

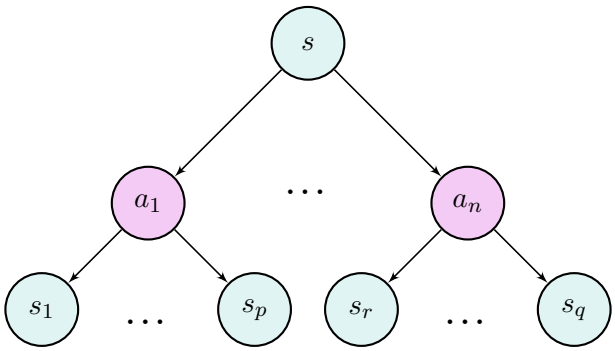

Figure 2.10: An backup diagram that shows the *actions* and *states* involved in computing the value of s using DP policy iteration.

Dynamic programming is not very efficient for large problems and suffers from the major drawback that the entire dynamics of the model must be known.

Monte Carlo Methods

Monte Carlo (MC) methods compute the optimal policy based solely on experience and do not require a full description of the model, as long as we can produce sample runs from the model. The name comes from the fact that the experiences are non deterministic and Monte Carlo is commonly used to refer to methods that rely on randomness. We compute the value function by averaging the return from each state over a number of sample runs, also known as traces, until we reach convergence. As we require a trace run to eventually terminate, MC methods are only valid for episodic tasks. Upon reaching a terminating state in the trace, we go backwards through the preceding states, calculating the return following each state and then average that with previous returns from that state to compute the updated value functions. As in GPI, we then make the policy greedy with respect to these value functions and repeat until we have convergence to the optimal policy.

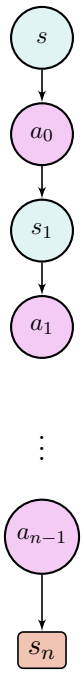

Figure 2.11: An example backup diagram from a single trace used in MC policy iteration, demonstrating the involved states and actions used in updating the value for s.

Temporal Difference Learning

Temporal Difference (TD) Learning methods were originally described by Richard Sutton and combine the best of dynamic programming and Monte Carlo methods [22]. Like in DP, they *bootstrap* by using estimates to update other estimates. They also *sample* like Monte Carlo methods and do not require a full model as they compute their estimates based on experienced runs. The word *temporal* is used because we use the differences of value estimates for states experienced at different time points.

Rather than having to wait for a complete return R_t to be produced upon reaching a terminal state before updating the value of s_t, TD methods can update the value of s_t at the very next time step based on the estimated $V(s_{t+1})$ and the sampled r_{t+1}. This can be critical for tasks with large episodes as TD methods do not need to wait until the end of the episode before updating their value estimates whereas MC methods do. It also means that TD methods can be used for non-episodic tasks as they do not require the episode to complete before they can update.

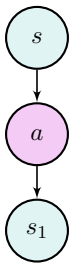

Figure 2.12: An example backup diagram involving a state and its successor as used in TD(0) methods

It has been proved that TD methods converge to V^π if the policy π remains fixed. Whilst mathematically it has not been proven that TD methods converge faster than other methods, in practice they typically do, further justifying their use [6].

One of the most popular TD methods is TD(λ) which rather than assign value just to the previous state as in TD(0), assigns value to all states in the trace leading up to the return according their *eligibility*, or contribution to the current trace. We choose $\lambda \in [0, 1]$ which Boyan refers to as a trade off between *bias* and *variance* [23]. When $\lambda = 0$, TD(λ) reduces to the one step lookahead of TD(0) shown in 2.12 which has a lower random variance being based only on a one step transition but is biased by the estimate of V^π for the subsequent state. For $\lambda = 1$, TD(λ) becomes equivalent to Monte Carlo methods and considers only the whole trace up to the terminating state. This has a larger variance as it depends on a typically much larger sequence of stochastic rewards but is unbiased as it does not use estimates of V^π. For intermediate values, we average the returns for each of the *n-step* traces following the current state and these are weighted by the value of λ. As $\lambda \to 1$, more of the preceding states have their values updated although this change is smaller for states further away from the current state. This is demonstrated in figure fig. 2.13.

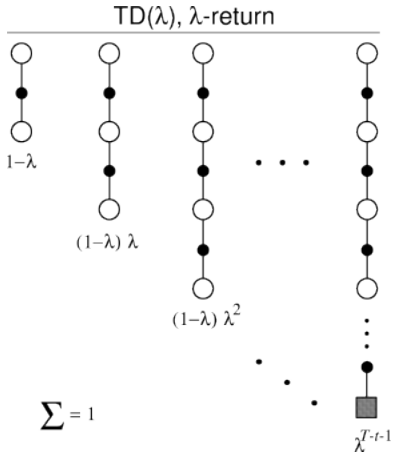

Figure 2.13: The backup diagrams for TD(λ) showing the weight of each of the returns [6].

2.3.3 Considerations

There are some considerations that we need to bear in mind when using the reinforcement learning techniques described above and we explain the key ones below.

Exploration vs Exploitation

In Monte Carlo and temporal difference methods, we need to make sure that we would reach each state and execute each action an infinite number of times if we performed an infinite number of trace runs. This ensures that we evaluate each state and action fairly and do not confine ourselves to a local minimum. However, as we are updating the policy to make it greedy, we will create a policy that could very likely never execute a given action and this would therefore subsequently never be evaluated. At the same time, we want to make sure that we continue to perform actions that we suspect from experience produce significantly high returns. The trade off between these two demands is known as *exploration vs exploitation*; we need to continue to *explore* all actions whilst *exploiting* those we think produce high returns [21]. Two popular solutions exist to this problem, ϵ-*greedy* policies and *softmax* policies.

ϵ-greedy policies work by defining a value ϵ such that whenever the agent needs to choose an action, it chooses a random one with probability ϵ. As $epsilon \to 0$, the agent becomes more exploitative, being completely greedy when $\epsilon = 0$, whilst as $\epsilon \to 1$, the agent will be more inclined to explore, acting randomly at the case $\epsilon = 1$. Choosing ϵ is not an exact science as the influence of ϵ on the time taken for an ϵ-greedy policy to converge will depend significantly on the variance of the rewards in the model.

Alternatively, we have the softmax policy where the agent selects an action with a probability proportional to the exponential of its value.

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}(s)} e^{Q(s,a')/\tau}}$$

The parameter τ is a positive constant known as the *temperature*. The higher the value of τ, the more explorative the agent will be whereas as $\tau \to 0$ the agent becomes more exploitative. Unlike the ϵ-greedy policy, τ is not simply between 0 and 1 and must be chosen by guessing how large the Q values will be, making it less intuitive and simple to use than ϵ-greedy. Its advantage however, is that actions are chosen with a probability that is proportional to their estimated value rather than equally across all suboptimal actions. In practice, both methods can be used and are often chosen based upon the requirements of the task [6].

Expanding to Continuous State and Action Spaces

The methods above describe storing explicit value function estimates for each states which are then used to find the optimal policy. In many situations however, there may be a large

number of states and actions, even a continuous amount, and the methods described above may not be immediately suitable. Fortunately, the key ideas can still be used and combined with numerical techniques such as *function approximation*, these methods can be adapted to suit these conditions which we will see when we look at *Least-Squares Temporal Difference* learning.

Chapter 3

A Novel Reinforcement Learning Approach to Algorithmic Forex Trading

We propose a novel algorithmic trading strategy that presents the current market circumstance to an agent, allows it to analyse this situation using its experience and then make a decision as to what the best investment action should be. For example, if the current conditions indicated a *bull market*[1] then the agent might choose to invest in a long position with the expectation that the rising prices will lead to a profit. If at a later time, the market suggested that this upwards trend was coming to an end, the agent may choose to close its long position and realise the profits that it potentially made. If the situation further indicated that a *bear market*[2] was imminent, then the agent could choose to open a short position with the hope of falling prices to make a profit.

We initially outline our proposal and justify its novelty before discussing the data available to us and the relevant reinforcement techniques that we will make use of, mainly the least-squares temporal difference learning algorithms. We will then formally present our approach in the format of a reinforcement learning problem, describing and justifying the design choices that we have made.

3.1 Outline and Novelty

To achieve this, we intend to use the reinforcement learning framework defined earlier as we perceive the above problem to be one that can be solved using RL methods. We consider the state space to contain the necessary information regarding the market and any current investments, the action space to be the possible investment opportunities at a given time and the reward function to reflect the profits that the agent earns. We will use techniques based upon Least-Squares Temporal Difference (LSTD) learning methods and will evaluate

[1] A market where an upwards trend in the price of the involved assets is expected

[2] A market where a downwards trend in the price of assets is expected

their success in solving our problem.

The underlying novelty in this approach compared to supervised or deterministic algorithmic trading methods is that we will expect the agent to learn for itself and allow it to make its own decisions based upon its previous profit rewards and experience. Examples of profitable circumstances and the appropriate actions will not be provided to the agent and it will need to learn to identify these. This could lead it to adapt to changes in market conditions automatically without the need for human intervention, including scenarios that may have never been experienced before. If it detected it was achieving poor performance and was unsure what the current optimal decisions should be, it could suspend from actual trading, simulate the environment to discover the optimal investment choices and then re-engage its trading capabilities.

It will be free to make its own investment decisions which ideally we will not need to intervene in, hence why we have chosen to use words such as *might*, *may* and *could* to describe the potential actions of the agent as we will not know for certain which action the agent deems best to take ourselves. As with most algorithmic trading strategies, acceptable loss levels should be set if used in production that prevent the agent from losing a large amount of money within a set time period and this would be the only example of intervention; with regards to the actual decision making, the agent will be completely autonomous.

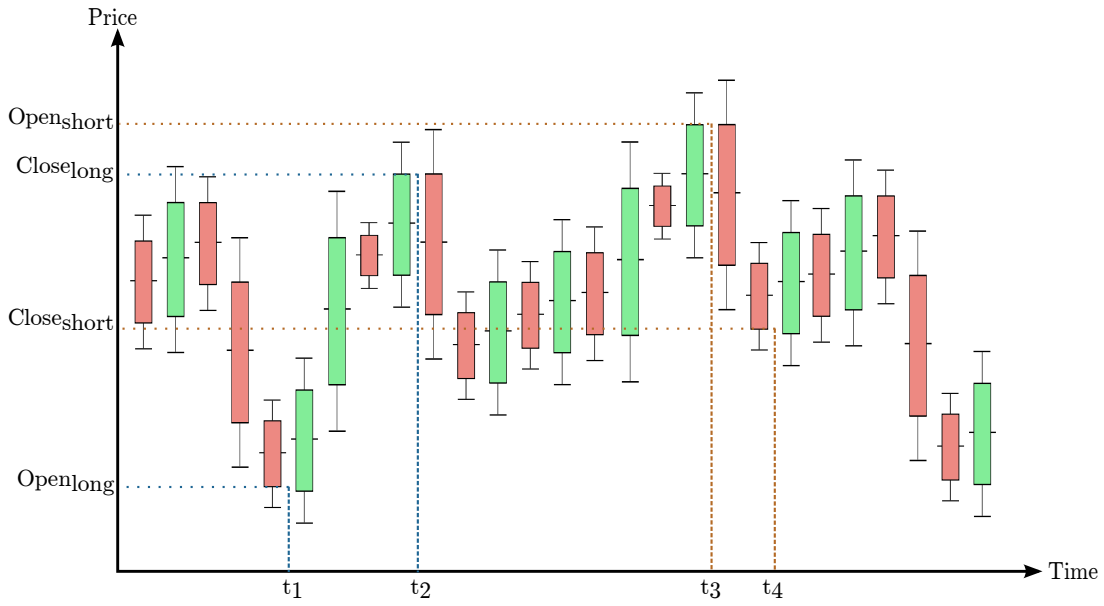

Figure 3.1: An example of the agent's possible positions choices on a candlestick chart. Blue markings demonstrate the details of the long position and orange markings show a short position.

We can adapt the candlestick chart from 2.5 to provide examples of possible positions that the agent could take as shown in 3.1. At time t_1, the agent could recognise that an upward trend was imminent and take a long position starting at the price $\text{open}_{\text{long}}$, say 1. At t_2, the agent could predict that this upward trend was about to end and close the long position at $\text{close}_{\text{long}}$, say 1.1, which would generate a profit of 0.1 across the period. Simi-

larly, the agent could initiate a short position at t_3 in anticipation of a downwards movement which they could then close at t_4 if they thought the market would revert upwards. This would then generate a profit of $\text{open}_{\text{short}}$ minus $\text{close}_{\text{short}}$. These are just two examples of possible positions that the agent could take; ideally, if the agent could predict every price movement perfectly, they would take the corresponding profitable long or short position at each time step to generate the maximum possible profit available. This ability to realise profit across market movements in either direction is why *volatility*[3] is often considered to be a trader's best friend for as long as the market is moving in some direction, there will be opportunities for profit and the greater the movements, the larger the available profits will be.

We outlined in the background section that many reinforcement learning methods require the problem to be both Markov and episodic. LSTD defines such a class of methods that requires the task to be episodic and Markov and we therefore need to justify that our idea satisfies these properties. The approach we have introduced defines a problem that is Markov since all of the historical information that the agent needs to make its decision will be contained within the state that represents the current market condition. The opening value and type of the position will also be stored in the state as required. Furthermore, if we define a day of trading to be an episode, then we ensure that the task is episodic. The state that contains the last candlestick of the day therefore represents a *terminal* state and we will use this to end an episode.

3.2 Data and LSTD Background

A crucial component of any machine learning algorithm is the data that is available for training and testing. Our proposed algorithm will solely use a time series of candlesticks across a given time period as its data source. We have kindly been given a significant volume of candlestick data by the Swiss Finance Corporation with which we can evaluate our algorithm.

3.2.1 Candlesticks

A candlestick representing the movement of a financial asset over a time period, as described in the financial background, can be represented as a vector $d \in \mathbb{R}^4$. We will use the following common convention for ordering the 4 values in a candle; $d = (open, high, low, close)^\intercal$. We can therefore think of our dataset as a time series of vectors in $\mathbb{R}^4$. Each candle stick also has an associated period, determined by the start and end times which we will record in our implementation.

[3]The uncertainty in the price of an asset, often computed as the standard deviation of the returns per unit time.

3.2.2 Available Currencies and Ranges

We have been given 11 sets of candlestick time series data that represent the bid rates of a particular currency pair, made up predominantly of the 4 major currency pairs. Each set covers approximately one year's worth of data with the candlesticks representing 1 minute time periods. The table below details each of these data sets, their time period and the number of candlesticks that they contain. Note that these candlesticks are only available for times when the market is open and there is therefore no data for the times when it is closed, such as on Saturdays. Whilst the data only provides the *bid* prices and not the *ask* prices, we feel that for a proposed approach this will suffice so that we ourselves could potentially factor in both the implicit and explicit transactions costs separately in the future to see how it affects the model.

Currency Pair	Start Date	End Date	#Candles
Euro - Pound	2014-01-02	2014-12-19	358857
US Dollar - Canadian Dollar	2014-01-02	2014-12-19	358855
US Dollar - Swiss Franc	2014-01-02	2014-12-19	358857
US Dollar - Japanese Yen	2014-01-02	2014-12-19	358857
Euro - Canadian Dollar	2014-01-02	2014-12-19	358847
Euro - US Dollar	2014-01-02	2014-12-19	358857
Euro - Australian Dollar	2014-01-02	2014-12-19	358847
Pound - US Dollar	2014-01-02	2014-12-19	358857
Euro - Japanese Yen	2014-01-02	2014-12-19	358857
Australian Dollar - US Dollar	2014-01-02	2014-12-19	358857
Pound - Japanese Yen	2014-01-02	2014-12-19	358857

The table above illustrates that over the 11 currency pairs, we have a significant amount of data to work with for training and testing purposes. Each of the datasets also covers a similar time range and contains a similar number of candlesticks which will allow us to draw reasonable comparisons between the results obtained from each datasets. Additionally, it will allow us to evaluate the performance of our algorithm based upon the average results across all the datasets rather than relying on just one. When investing, it is considered wise to choose a variety of assets with negative correlations between them to achieve a *balanced portfolio* [24]. A robust trading algorithm would therefore choose to operate on a collection of assets and so we are fortunate to have the collection above available to us.

3.2.3 Further Data Sources

Financial data sets, such as the exchange rates of currency pairs, are widely available through numerous APIs, such as the OANDA Exchange Rate service[4], that a user can

[4]OANDA Service - https://www.oanda.com/rates/

request in programming languages such as python. Many services require a subscription fee to obtain real time feeds and are used predominantly by proprietary trading firms to maintain feeds that are updated in real-time. However, for our purposes, the 1 minute candlesticks provided should suffice. Alternatively, Yahoo Finance[5] offers a free API that allows one to access stock and currency data at a higher frequency of 1 per second. This could prove useful for future work if we wish to evaluate the algorithm on data made up of shorter time periods or on stock data rather than foreign exchange.

3.2.4 Least-Squares Temporal Difference Learning

TD(λ) policy evaluation methods that use *function approximation* aim to converge to a vector $\boldsymbol{\beta}$ such that $V^\pi(s) = \boldsymbol{\beta}^\intercal \boldsymbol{\phi}(s)$ where π is the fixed policy to be valued, s is a state and $\boldsymbol{\phi}(s)$ is a *feature vector* used to represent s as a numerical vector. The traditional TD(λ) method requires the choice of stepsize parameters which it then uses to update $\boldsymbol{\beta}$ in a form of *stochastic gradient descent*. Unfortunately, incorrect choices of these stepsize parameters can often lead to poor results and the stochastic gradient descent nature of the algorithm means that it often uses the available data inefficiently [23]. Least-Squares Temporal Difference (LSTD) learning is a temporal difference technique first proposed by Bradtke and Barto [25] that aims to resolve these issues. Rather than using gradient descent, LSTD iteratively builds an explicit matrix $\mathbf{A}$ and vector $\mathbf{b}$ such that $\boldsymbol{\beta}$ can be estimated as $\boldsymbol{\beta} = \mathbf{A}^{-1}\mathbf{b}$.

Note. When computing $\mathbf{A}^{-1}\mathbf{b}$, it may happen that $\mathbf{A}$ is singular and traditional matrix inversion will fail. It is therefore recommended that one uses the *Moore-Penrose Psuedoinverse* and *Singular Value Decomposition* to compute the best fit solution [26].

Bradtke and Barto derived the original LSTD algorithm which effectively only handles the case when $\lambda = 0$. Boyan then extended this work to all $\lambda \in [0, 1]$ and also provided a simpler derivation of the algorithm [23]. For $\lambda > 0$, we require that the task is *episodic* as we will need the agent to receive a terminating reward signal. We clarify some of the additional parameters used before presenting the LSTD(λ) algorithm as described by Boyan.

Symbol	Usage
$K \in \mathbb{N}$	The size of a state's feature vector representation
$\boldsymbol{\phi}(s) \in \mathbb{R}^{\mathrm{K}}$	A feature vector representing state s
$\boldsymbol{\beta} \in \mathbb{R}^{\mathrm{K}}$	A coefficient vector used to estimate V^π
$\mathbf{z_t} \in \mathbb{R}^{\mathrm{K}}$	Vectors used at each time step to store the *eligibility* of each state for reward
$\mathbf{b} \in \mathbb{R}^{\mathrm{K}}$	A vector produced in the algorithm used to compute $\boldsymbol{\beta}$
$\mathbf{A} \in \mathbb{R}^{\mathrm{K}*\mathrm{K}}$	A matrix produced in the algorithm used to compute $\boldsymbol{\beta}$
N	The number of trace episodes to use in policy evaluation

[5] Yahoo Finance Python API - `https://pypi.python.org/pypi/yahoo-finance/1.1.4`

Note. In RL problems, a state often represents some abstract representation of the current situation that may not be immediately quantifiable. Function approximation algorithms rely on the fact that a state can be represented numerically and we are therefore required to define a *feature vector* denoted as $\boldsymbol{\phi}(s) \in \mathbb{R}^{\mathrm{K}}$. Care needs to be taken to make sure that the chosen feature vector design contains enough information to uniquely describe a state and accurately reflect the current situation of the agent and its environment. It is also typical that the complexity of function approximation RL methods depend highly on the value of K and so it should be chosen to be sufficiently large to contain the information required but no larger.

LSTD(λ, π) Policy Evaluation Algorithm

The LSTD(λ, π) algorithm is used to estimate the value of a state under a fixed policy π. To calculate the value of a particular state s, we use the following estimation: $V^{\pi}(s) \approx \boldsymbol{\beta}^{\intercal}\boldsymbol{\phi}(s)$.

A $:= \mathbf{0}_{K,K}$
b $:= \mathbf{0}_{K,1}$
$t = 0$
for $n = 1, 2, \ldots, N$ **do**
  Choose start state $s_t \in \mathcal{S}$
  $z_t = \boldsymbol{\phi}(s_t)$
  while $s_t \neq END$ **do**
    (s_{t+1}, r_{t+1}) = simulate one time-step according to π
    $\mathbf{A} = \mathbf{A} + z_t(\boldsymbol{\phi}(s_t) - \boldsymbol{\phi}(s_{t+1}))^{\intercal}$
    $\mathbf{b} = \mathbf{b} + z_t r_{t+1}$
    $z_{t+1} = \lambda z_t + \boldsymbol{\phi}(s_{t+1})$
    $t = t + 1$
  end while{Set $\boldsymbol{\beta} = \mathbf{A}^{-1}\mathbf{b}$ when required using SVD}
end for
return $\boldsymbol{\beta} = \mathbf{A}^{-1}\mathbf{b}$

The computational complexity of LSTD(λ) is described in terms of K, the size of the feature vector and N, the number of traces. The updates to **A** and **b** have a time complexity of $\mathrm{O}(K^2)$ and the matrix inversion required to obtain $\boldsymbol{\beta}$ has time complexity $\mathrm{O}(K^3)$. We typically only need to re-compute $\boldsymbol{\beta}$ once per trace rather than for each timestep to observe convergence. TD(λ) has a time complexity of only $\mathrm{O}(K)$ when updating coefficients each time step and so LSTD(λ) is typically more expensive computationally with an overall time complexity of $\mathrm{O}(NK^3)$. However, we list the following advantages that LSTD(λ) provides [23].

- LSTD(λ) does not require a stepsize parameter which can badly affect performance if chosen improperly.

- LSTD(λ) does not require an initial estimate for $\boldsymbol{\beta}$ which can also significantly limit TD(λ) if incorrectly chosen.
- LSTD(λ) only requires one parameter, λ, and this does not actually affect the convergence significantly.

Modern reinforcement learning methods are still very much in development and as such, are often heavily reliant on parameter choices. Additionally, to complicate things further, there is little guidance on how parameters should be chosen to best suit a particular problem. Experimentation is often required to correctly choose parameters that are optimal for the given problem and this can severely impact one's progress. Therefore, any method that allows the user to reduce the number of parameters required or minimise the effect that a poor parameter choice would have on the results offers significant advantages over those which do not. With this in mind and the advantages listed above, we feel that LSTD(λ) is more suited to the scope of this project and we will use this as the basis for our reinforcement learning.

The LSTD(λ) method provides a way to evaluate a fixed policy π. To complete the *generalised policy iteration* process, we need to update the policy to be greedy with respect to value function, i.e. the $\boldsymbol{\beta}$ vector. As described in GPI, we repeat this process until we obtain sufficient convergence, determined by the threshold ϵ, in the policy as demonstrated in the algorithm below, inspired by the GPI shown by Lagoudakis and Parr [27].

LSTD(λ) Policy Iteration Algorithm

The LSTD(λ) policy iteration algorithm uses LSTD(λ) to evaluate a policy, then improves the policy to be greedy with respect to the computed value functions and this repeats until convergence is reached.

1: $\pi_0 = \mathbf{0}$
2: $\pi' = \pi_0$
3: **repeat**
4: $\quad \pi = \pi'$
5: $\quad \pi' = LSTD(\lambda, \pi)$
6: **until** $||\pi - \pi'|| < \epsilon$
7: **return** π - This is the *optimal policy*

3.2.5 Least-Squares Policy Iteration

Least-Squares Policy Iteration (LSPI) is an expansion of the LSTD(λ) algorithm proposed by Michail G. Lagoudakis and Ronald Parr [27]. They argue that using LSTD(λ) as the evaluation algorithm in policy iteration could be problematic as it learns the state value functions, $V^{\pi}(s)$, rather than the action value functions, $Q^{\pi}(s, a)$, which means that one has to estimate the expected reward when computing the optimal policy. They introduce LSTDQ which operates very similarly to LSTD but estimates $Q^{\pi}(s, a)$ for a fixed policy π.

Their algorithm follows the same structure as the LSTD(λ) except that rather than using a feature vector to represent a state, $\boldsymbol{\phi}(s)$, they use them to represent state-action pairs, $\boldsymbol{\phi}(s, a)$. This then produces a vector $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$ which can be used to calculate the value of a state-action pair, $Q(s, a) = \mathbf{w}^{\intercal}\boldsymbol{\phi}(s, a)$. As was the case in LSTD(λ), we require that the feature vector accurately represents the required attributes of a particular state-action pair. The GPI process used is then the same as the one demonstrated for LSTD(λ) with the difference of using LSTDQ rather than LSTD. We evaluate both approaches to see if these suggested benefits can be realised within our domain.

3.3 Proposed Reinforcement Learning Model

We propose a reinforcement learning trading algorithm that uses LSTD methods to compute an optimal policy with respect to the provided training data set. We outline the key components of our approach that we will use to define our problem in terms of the reinforcement learning problem framework.

3.3.1 State Representation

We chose to define a state, s, such that it contained the information relevant to the current position held as well as that of the time series data over a fixed number of past periods to act as a *history* which the agent could use as a basis to select the optimal actions.

Current Position

For simplicity, we chose to only handle a single position at once rather than allowing the agent to initiate multiple positions simultaneously. Therefore, if the agent wishes to transition from a long position to a short position, it must first close the long position and then open the short position in the next epoch. The agent can therefore either be in a state with an open long position, an open short position or be *idle* where no position is currently held. Additionally, to define a position, we need to record the value at which the currently open position was opened so that any profits and losses can be calculated as the basis for future rewards. For the long and short position states this is the positive value of the financial asset at the time the position was opened. For the idle state this is not defined as no position is currently open.

Time Series History

To allow the RL agent to recognise the trends in the market that lead to the rewards it receives, we would require that a state maintains a record of a fixed number, n, of past candlesticks.

Definition. For a state's candlestick history of size n, we define this as the following vector $\mathbf{s_h} \in \mathbb{R}^{4n}$ where $\mathbf{d_i}$ represents the i^{th} most recent candlestick with $\mathbf{d_1}$ therefore being the most recent candlestick.

$$\mathbf{s_h} = \begin{pmatrix} \mathbf{d_1} \\ \mathbf{d_2} \\ \vdots \\ \mathbf{d_n} \end{pmatrix} = \begin{pmatrix} open_1 \\ high_1 \\ low_1 \\ close_1 \\ \vdots \\ open_n \\ high_n \\ low_n \\ close_n \end{pmatrix}$$

We experimented with different possible history structures, initially choosing to simply record the n past candles in a linear value. A more complex approach would be to average the candlestick values over given periods to allow the state's history to cover a larger time range whilst maintaining the same number of stored candlesticks values. Storing an excessive number of candlesticks would be expensive in both computational space and time for large time series data sets and we wished to avoid this.

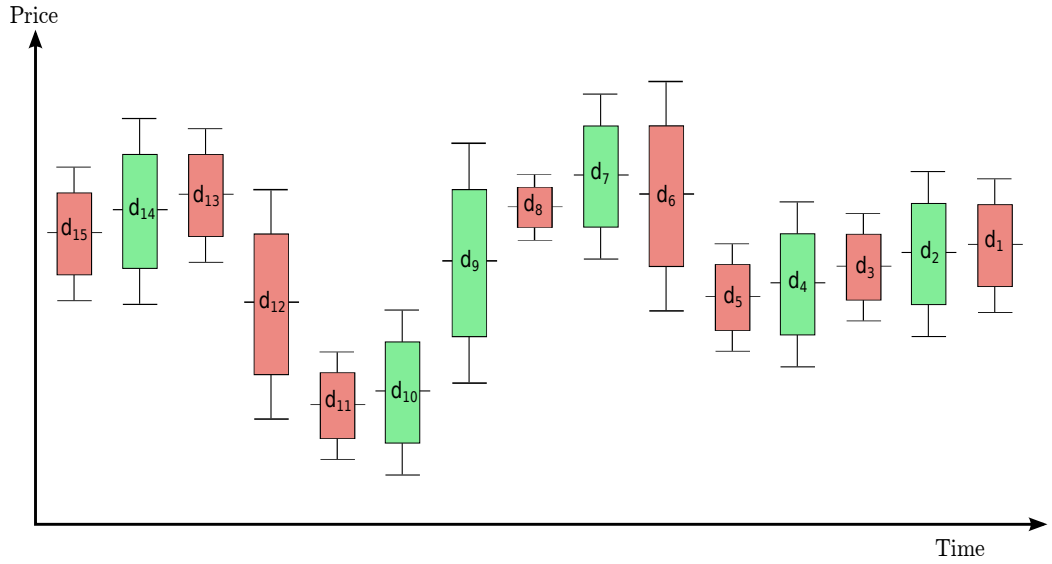

Figure 3.2: An example candlestick chart used to illustrate different history structures.

To illustrate the possible different history structures, we use the example candlestick chart in figure 3.2 containing 15 candles. If we just used a simple history structure where each candle was equally ranked, we would have a history of size 15 weighting each candle equally. An example of an alternative history structure would be $[5, 4, 3, 2, 1]$ which should be read as the number of periods to average the candlesticks over. This would lead to the

following candlestick history of size 5.

$$\mathbf{s_h} = \begin{pmatrix} \mathbf{d_1} \\ \frac{\mathbf{d_2}+\mathbf{d_3}}{2} \\ \frac{\mathbf{d_4}+\mathbf{d_5}+\mathbf{d_6}}{3} \\ \frac{\mathbf{d_7}+\mathbf{d_8}+\mathbf{d_9}+\mathbf{d_{10}}}{4} \\ \frac{\mathbf{d_{11}}+\mathbf{d_{12}}+\mathbf{d_{13}}+\mathbf{d_{14}}+\mathbf{d_{15}}}{5} \end{pmatrix}$$

Similarly, we could use a history with an exponential structure, i.e. taking the form [8,4,2,1] which would produce the following history vector of size 4.

$$\mathbf{s_h} = \begin{pmatrix} \mathbf{d_1} \\ \frac{\mathbf{d_2}+\mathbf{d_3}}{2} \\ \frac{\mathbf{d_4}+\mathbf{d_5}+\mathbf{d_6}+\mathbf{d_7}}{4} \\ \frac{\mathbf{d_8}+\mathbf{d_9}+\mathbf{d_{10}}+\mathbf{d_{11}}+\mathbf{d_{12}}+\mathbf{d_{13}}+\mathbf{d_{14}}+\mathbf{d_{15}}}{8} \end{pmatrix}$$

As mentioned, the advantage of averaging candles over multiple periods is that it allows us to store information over the same period in a much smaller number of candles. In doing so however, we do lose explicit information about the price movements in a particular period and we therefore face the common trade-off of storage space vs information stored. The above structure also implies that for the first n time periods of the day where n is the time length of the history vector, the agent should be unable to execute actions as there is insufficient data in its candlestick history for it to make an informed choice.

Note. Whilst the agent can only find itself in a state holding a long, short or no position; because the time series history represents a vector of real values, our state space is continuous and we will therefore use LSTD methods as they have been shown to work on continuous state spaces.

We complete our definition of a state with an illustration that shows all the values used in a state's representation as well as their types.

State Representation

$$\text{State History} = \begin{pmatrix} \mathbf{d_1} \\ \mathbf{d_2} \\ \vdots \\ \mathbf{d_n} \end{pmatrix} \in \mathbb{R}^{4n} \quad ; \begin{array}{ll} \text{Opening Price} & \in \mathbb{R}^+ \\ \text{Current Price} & \in \mathbb{R}^+ \\ \text{Position Type} & \in \{\text{S}, \text{L}, \text{I}\} \end{array}$$

State Feature Vector

As described earlier, a crucial element of the LSTD algorithm is the feature vector representing a state that provides a numerical vector used to compute the value of the state. We required that this feature vector accurately encapsulated the current position and time series history details described above.

To encapsulate the time series history data, we chose to define m centre histories of the same time length, n, as the history stored in a state that we would use to compare a state's history to. Each of these m histories therefore represents n consecutive candlestick values stored in a vector. We initially sought to establish these centres through *k-means clustering*[6] using initial centres that were equally spaced across the time series history data. We found however that given the size of our data set, this was often very computationally expensive and made testing our approach highly challenging. We therefore chose to use candlestick histories that were equally spaced across the dataset. We would then use these m candlestick centres to define a vector in $\mathbb{R}^{\mathrm{m}}$ that represents the position of a particular history with respect to the centre histories selected.

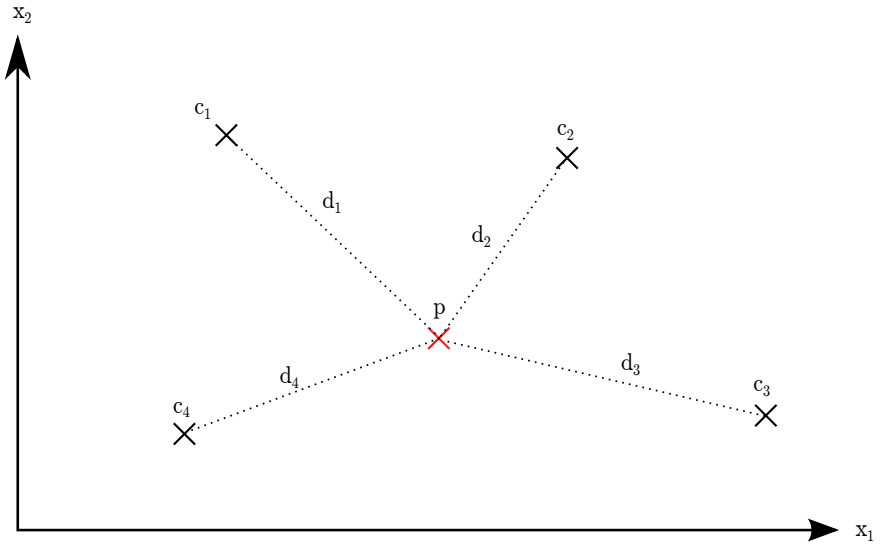

Figure 3.3: An example of using the centre histories to describe the position of a history.

For example, in figure 3.3 we can see that there are 4 centre vectors $c_1, c_2, c_3, c_4 \in \mathbb{R}^2$ represented as black crosses. Similar to the concept of triangulation, we can use the distance from these centres to describe the position of the target vector, p shown as the red cross, relative to the chosen centres. This would produce the following vector to describe the position of p. Whilst this example describes a situation in $\mathbb{R}^2$ so that it can be shown in this report, the concept easily generalises to $\mathbb{R}^{4\mathrm{n}}$ as required by us.

[6] An algorithm that given a data set and an initial estimate of the central points in the data set, iteratively assigns the data points and re-calculates the centres until it converges to the central points.

$$\mathbf{p_{pos}} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}$$

Radial basis Functions (RBFs) are a family of real valued functions whose values depend only on the distance of a point $\mathbf{x}$ to another point $\mathbf{c}$ which is often referred to as the centre. We chose to use Gaussian radial basis functions which take the following form.

$$\phi_{gauss}(\mathbf{x}, \mathbf{c}) = e^{-(\epsilon\|\mathbf{x}-\mathbf{c}\|^2)}$$

Where ϵ is some positive constant used to control the width of the Gaussian produced by the RBF. For values of ϵ close to 0, the resulting Gaussian is quite wide and is not as sensitive to small changes in the distance for $\mathbf{x}$ values close to the centre. For larger values, such as $\epsilon > 1$, the Gaussian is much thinner and is very sensitive to changes in the distance from the centre for $\mathbf{x}$ values near the centre. One can therefore choose a value of ϵ that reflects how localised they want their RBFs to be. Figure 3.4 shows a Gaussian RBF for different values of ϵ for a value $x \in \mathbb{R}$ with a centre of 0. The justification for using Gaussian RBFs is that their responses are only significant for values close to their centres and therefore [28], a history that was particularly close to one of the centre histories would be easily recognisable in the feature vector representation.

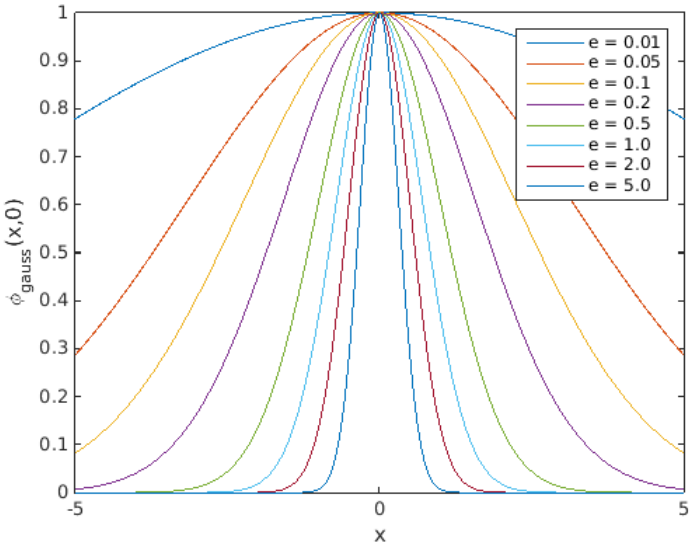

Figure 3.4: A demonstration of how various ϵ alter the shape of a Gaussian RBF.

Using the Gaussian RBFs as described above, we were able to define the following

representation of a state's candlestick history position within the data set.

Definition. The position vector of a state's candlestick history $\mathbf{s_h}$ relative to the m predefined centre histories $\mathbf{c_1} \dots \mathbf{c_m}$ is the following vector.

$$\boldsymbol{\phi}_{pos}(s) = \begin{pmatrix} \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \end{pmatrix}$$

To represent the state's position details, we identified two possible solutions. The first leads to a larger feature vector but is more explicitly different for each of the position types whilst the second is almost a third of the size smaller but only differentiates each position type by one value in the feature vector representation as shown below. Additionally, rather than storing the opening value of a position directly in the feature vector, we chose to store the unrealised profit and loss (PnL) of the position, calculated as below for each position type ((L)ong, (S)hort and (I)dle positions). See section 2.1.3 for further details on computing these values.

Note. $\text{value}_{\text{current}}$ refers to the most recent price value that the agent is aware of. The agent stores this in each state implicitly as the closing value of the most recent candle and we therefore have that $\text{value}_{\text{current}} = s.d_1.\text{close}$ where d_1 is the most recent candle.

$$\begin{aligned} L_{pnl} &= \text{value}_{\text{current}} - \text{value}_{\text{opening}} \\ S_{pnl} &= \text{value}_{\text{opening}} - \text{value}_{\text{current}} \\ I_{pnl} &= 0 \end{aligned}$$

The first feature vector representation is a vector, $\boldsymbol{\phi}^1(s) \in \mathbb{R}^{3\text{m}+3}$, where a short position is stored in the $1 \dots m+1$ indices, no position is stored in the $m+2 \dots 2m+2$ indices and a long position is stored in the $2m+3 \dots 3m+3$ indices with the remaining $2m+2$ indices being zero in each possible state. This is shown below where $\mathbf{0}^{m+1}$ is a column vector of zeroes of size $m+1$. This approach of storing each possible state as a particular block in the feature vector with the remaining blocks being zero vectors was also used in the LSPI examples such as the RL problem of balancing a bicycle [27]. As the value of a state is computed as the inner product of $\boldsymbol{\beta}$ and $\boldsymbol{\phi}(s)$, only the non zero values in the feature vector will be used to calculate the value as required.

Note. For simplicity, we describe the feature vector in terms of $\boldsymbol{\phi}_S(s)$, $\boldsymbol{\phi}_I(s)$ and $\boldsymbol{\phi}_L(s)$, i.e. a separate feature vector for each position type. The overall feature vector would then be denoted as follows:

$$\boldsymbol{\phi}(s) = \begin{cases} \boldsymbol{\phi}_S(s) & \text{if } s \text{ is a short state} \\ \boldsymbol{\phi}_I(s) & \text{if } s \text{ is an idle state} \\ \boldsymbol{\phi}_L(s) & \text{if } s \text{ is a long state} \end{cases}$$

First Feature Vector Representation

$$\boldsymbol{\phi}^1_S(s) = \begin{pmatrix} \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \\ S_{pnl} \\ \mathbf{0}^{m+1} \\ \mathbf{0}^{m+1} \end{pmatrix} \quad \boldsymbol{\phi}^1_I(s) = \begin{pmatrix} \mathbf{0}^{m+1} \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \\ I_{pnl} \\ \mathbf{0}^{m+1} \end{pmatrix} \quad \boldsymbol{\phi}^1_L(s) = \begin{pmatrix} \mathbf{0}^{m+1} \\ \mathbf{0}^{m+1} \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \\ L_{pnl} \end{pmatrix}$$

The second feature vector representation is a vector, $\boldsymbol{\phi}^2(s) \in \mathbb{R}^{m+4}$. All types of state use the first m indices to store the values of the Gaussian RBFs for their candlestick history, the $m+1$ index to store the current PnL of the position and the final 3 indices to indicate which position type they hold as shown below.

Second Feature Vector Representation

$$\boldsymbol{\phi}^2_S(s) = \begin{pmatrix} \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \\ S_{pnl} \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{\phi}^2_I(s) = \begin{pmatrix} \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \\ I_{pnl} \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \boldsymbol{\phi}^2_L(s) = \begin{pmatrix} \phi_{gauss}(\mathbf{s_h}, \mathbf{c_1}) \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_2}) \\ \vdots \\ \phi_{gauss}(\mathbf{s_h}, \mathbf{c_m}) \\ L_{pnl} \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Upon experimenting with these two representations, we found that the smaller size of the second choice did not significantly impact the performance of the algorithm. We felt that since the value of a state was computed as a linear combination of the feature vector and β,

then explicitly assigning 0 values for all the indices not related to the current state would lead to more effective results and this was supported by the use of a similar technique in the examples for the LSPI paper [27]. We therefore chose to focus on using the first feature representation only within our evaluation.

3.3.2 Available Actions

Whilst our state space is continuous, our action space is not. We assume that the agent can only invest in one unit of the particular financial asset so it can therefore only buy or sell one unit at its current value when appropriate. This could be adapted to invest an arbitrary amount but we chose to restrict ourselves to 1 unit as it suffices for our proof of concept model. Therefore, all opening and closing actions represent doing so for one unit of the financial asset.

If the agent is currently in an *idle state* where no position is open, it has 3 possible actions; open a short position at the current price, open a long position at the current price or stay idle and wait for the next price value. Opening a position takes it to the respective state for that position type with an updated candlestick history.

In a state where the agent has a *long position* currently open, it can either close it at the current price, taking it to an idle state where it has no current position, or it can keep the long position open and wait for the next price value to be received. Similarly, in a state where the agent currently has a *short position* open, it can either close the position or stay and wait for the next candle.

Note. Any action that the agent takes, regardless of whether it moves it from one position type to another or not, also advances the agent's candlestick history to include the current candle so that it can be used in the decision making process for future states.

If we classify each of the states as either representing a long, short or idle position, we get the following action space.

$$\begin{aligned}\mathcal{A}(\text{long}) &= \{\text{close}, \text{stay}\} \\ \mathcal{A}(\text{short}) &= \{\text{close}, \text{stay}\} \\ \mathcal{A}(\text{idle}) &= \{\text{open}_{\text{long}}, \text{open}_{\text{short}}, \text{stay}\}\end{aligned}$$

Even though the *close* and *stay* actions can be used in different states, and even produce different rewards in the case of *close*, we consider them to be the same action.

3.3.3 Transition Probabilities

Within our reinforcement learning model, we will assume that all valid actions are guaranteed to succeed if the agent selects them. In reality, it may be possible that a transaction could fail due to technical issues or a lack of funds that make the investment action impossible. To simplify our model however, we will assume this does not occur and an action will always move the agent one state into the intended next state. Of course, the agent is not

aware of what the next price will be when it transitions into the next state but it is certain which type of state it will transition into.

3.3.4 Reward Structure and Objective Function

The reinforcement learning model trains itself by assigning the contributions of the rewards that the agent receives to the correct states and actions that led it there via value functions. To support this, we need to provide appropriate rewards for each possible action. We also need to ensure that our rewards best reflect the overall objective function rather than immediate sub-goals that could be pursued by the agent instead of the primary objective as noted in the RL background.

Our algorithm will be interested in pure profitability over a given set of data. We therefore define the objective function of the agent to be to maximise the cumulative profit and loss (PnL) of all its positions across the data set; see section 2.1.3 for details of a position's PnL.

We therefore define a reward as follows; if the agent opens a new position or continues to stay in its current position or idle, we generate a reward signal of 0 for it. Upon closing a position we generate a reward signal equal to the realised PnL of that position as defined in section 2.1.3.

3.3.5 Markov Decision Process

We can formalise the above descriptions into a complete specification of our Markov Decision Process. As we are on an infinite state space, we will classify each state as either being idle (I), long (L) or short (S) and will differentiate them at different time points by using a subscript for the time epoch.

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
I_t	I_{t+1}	`stay`	1	0
I_t	L_{t+1}	`open long`	1	0
I_t	S_{t+1}	`open short`	1	0
L_t	L_{t+1}	`stay`	1	0
L_t	I_{t+1}	`close`	1	$\text{value}_{\text{closing}} - \text{value}_{\text{opening}}$
S_t	S_{t+1}	`stay`	1	0
S_t	I_{t+1}	`close`	1	$\text{value}_{\text{opening}} - \text{value}_{\text{closing}}$

We can represent this table more clearly in the form of a state transition diagram between the classes of states as shown in figure 3.5.

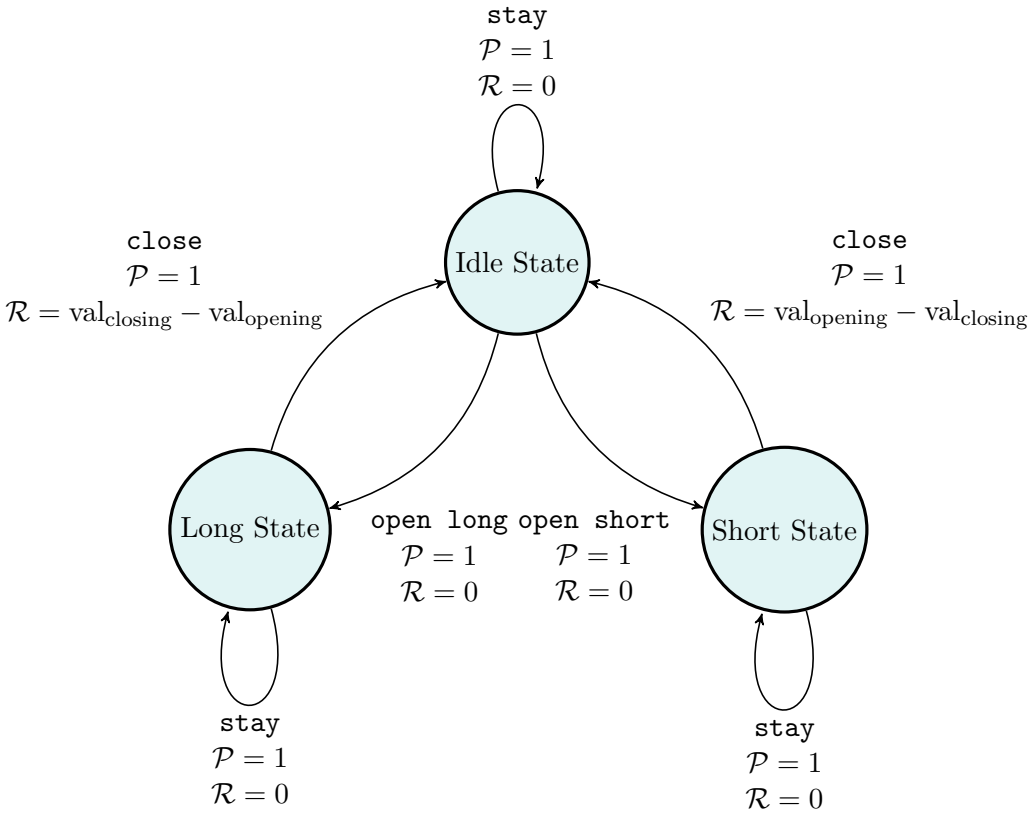

Figure 3.5: A Markov Decision Process transition diagram for the trading RL problem.

3.3.6 Parameter Selection

The model above is defined with respect to a set of parameters whose values will need to be chosen through intuition and experimentation. We list the keys parameters below alongside their possible range of values. We use ϵ to refer to both the width of the Gaussian RBFs and the exploration rate but we will make it clear which usage is intended when they are used.

Parameter	Values
Number of Gaussian RBF centres	$m \in \mathbb{N}$
Width of Gaussian RBFs	$\epsilon \in \mathbb{R}^+$
Reward discount factor	$\gamma \in (0, 1)$
History structure	List of number of days to average over in history in the form $[l_n, l_{n-1}, \ldots, l_1]$
Lambda	$\lambda \in [0, 1]$
Exploration epsilon	$\epsilon \in (0, 1)$

Chapter 4

An Evaluation of our Approach using Least-Squares Temporal Difference Learning

We now present an evaluation of our approach, first detailing our design choices and implementation strategy. We then discuss the results that we obtained for our reinforcement learning approach using least-squares temporal difference methods before addressing the challenges that we encountered during the project.

4.1 Design and Implementation

4.1.1 Language Choices

We used *Python* as our language of choice in the implementation of our strategies. The general purpose nature of python that allow it to support object-oriented design, dynamic typing and scripting techniques make it ideal for algorithmic trading development, allowing the user to easily create and modify their designs. Python also comes with a variety of packages, notably the *Numpy* and *Scipy* packages which we used extensively to manipulate vectors and matrices in LSTD. Python is a popular algorithmic trading language choice as it is not very verbose, making it easier to debug, and also has a wide range of finance libraries, such as *Pandas*[1], in development due to its active community [29]. It is also widely used in the machine learning community for similar reasons as above, particularly with regards to its support of large numerical data sets through libraries and its versatility.

[1]Pandas Python Data Analysis Library - `http://pandas.pydata.org/`

4.1.2 A Backtesting system

A crucial component of evaluating any algorithmic trading strategy is *backtesting*. Using past data, one runs their algorithm as if the data was being received in real-time. Performance metrics such as profitability and risk levels are then analysed to provide an indication as to whether the algorithm is worth pursuing. However, results obtained through backtesting alone are often not enough and testing on current data is usually required as confirmation. The increasing popularity of algorithmic trading strategies has led to the development of numerous platforms which can be used for back testing. *Quantopian*[2] for example is one such platform that allows its users to code their algorithms online in a python environment and backtest them on a large series of historical data. One drawback however is that it currently lacks support for foreign exchange trading and would therefore not be suitable for our needs. A python library called *PyAlgoTrade*[3] also provides support for back testing as well as trade execution techniques. Additionally, it provides support for handling twitter events which could also prove useful for future work that uses the details of economic events in its decision making.

We felt however that developing our own tailored system would provide us with greater flexibility when introducing the reinforcement learning techniques as well as further our understanding of algorithmic trading and the steps involved in creating a strategy. We therefore chose to create our own backtesting system, the implementation of which we discuss below.

Design

We aimed to design our backtesting system to be as general as possible so that it could potentially be used for a variety of strategies. We used abstract classes to provide a general testing framework that the models could then implement to interact with the testing system. The testing system is primarily event driven and we require that all models implement a method describing what to do when they receive a new candlestick and this method is then called by the tester for each candlestick in the time ordered data series. We outline the key classes used below in.

[2]Quantopian online backtesting system - `https://www.quantopian.com/`
[3]PyAlgoTrade Library - `https://gbeced.github.io/pyalgotrade/`

Class Name	Description
`model.py`	An abstract class that an algorithmic trading strategy will implement to allow it to be tested. Requires the definition of a `receive_candle(c)` method. It also provides methods that allow models to open and close long and short positions through the model testing interface.
`modeltester.py`	Takes a model and tests it on a given set of candlestick data before reporting the results. Maintains records of the positions opened and the PnL from them.
`candlestick.py`	Stores the data required for a single financial candlestick as well as provides a method to transform it into a vector.
`timeseries.py`	Provides a mechanism to access, modify and query a time-ordered collection of financial candlesticks.
`results.py`	A structure which maintains the results of a backtest. Updated when a position is opened and closed. Usually, we have daily results objects and an overall results object that aggregates the daily results.
`position.py`	An abstract class that represents a financial position, implemented by either a long, short or idle position. Maintains the details of the opening value as well as providing methods to calculate the PnL.

The UML diagram shown in figure 4.1 illustrates the relationships between these classes and how they come together to form the backtesting framework.

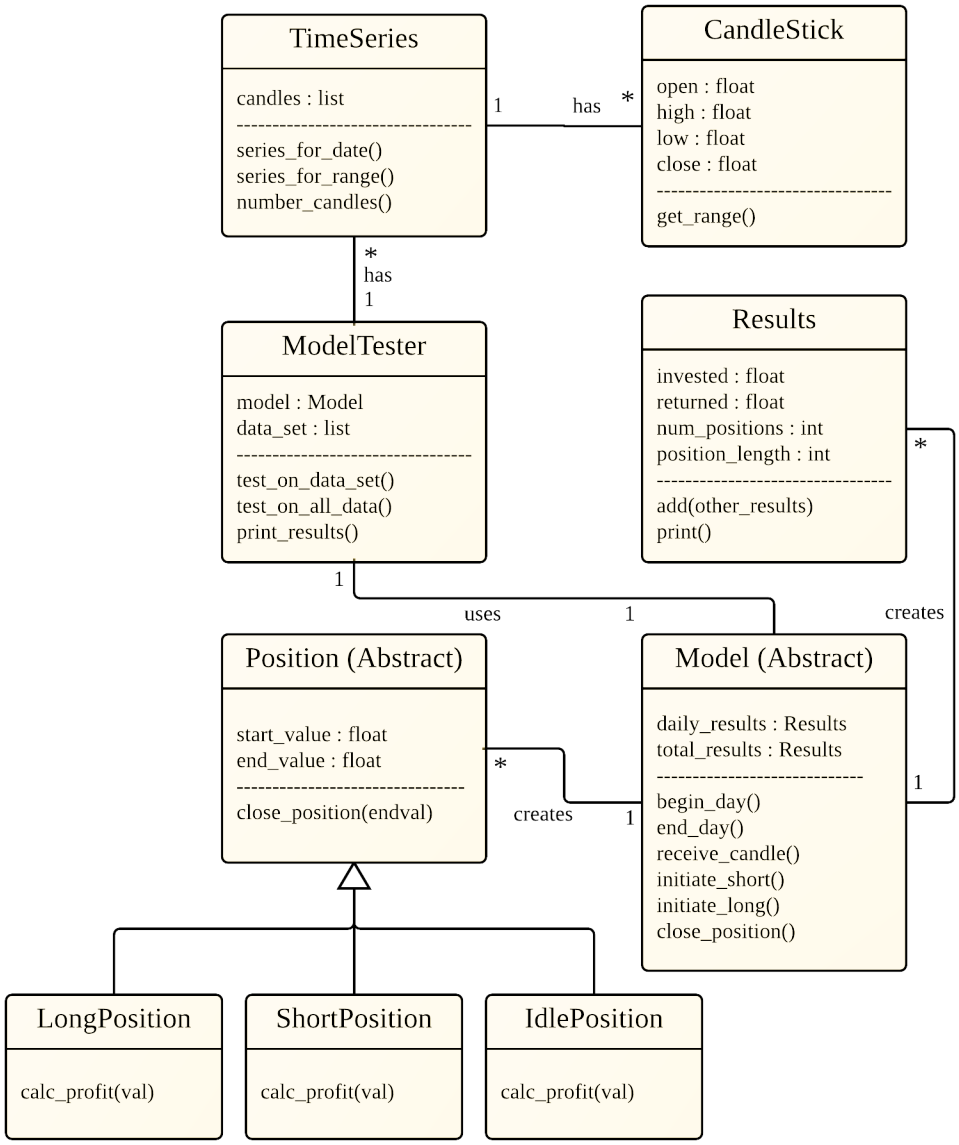

Figure 4.1: A UML diagram showing the design of the back testing system.

As each set of time series data can be tested separately from the others, we experimented with concurrently running the testing system on multiple threads to improve its computational performance. We found this led to much more efficient testing when using multiple datasets and the time taken to complete each test reduced substantially, particularly when using the LSTD algorithms as we could parallelise the training as well. If a model is defined such that it treats each day independently and does not carry any positions forward between days, then this could be extended to allow concurrent testing of individual days which could further improve the performance of the system.

4.1.3 Handling the Results

The *results.py* provides the mechanism for the backtesting system to record, analyse and display the results. Typically, we would use one result object for each trading day, providing details about the positions and results for that particular day, as well as one for the entire duration of the backtest. The table below describes the metrics that are reported by the system.

Metric	Description
Amount Invested	The total amount invested into the asset over the time period.
Amount Returned	The total amount returned from the asset investments over the time period.
Profit and Loss	The amount returned minus the amount invested over the time period.
Rate of Return	Calculated using the following formula $\frac{\text{Profit and Loss}}{\text{Amount Invested}}$.
Annualised Rate of Return	The rate of return multiplied by the ratio of the length of the period to the length of a year to estimate the expected annual return.
Number of Positions	The total number of positions opened and closed during the period. Also records the number of long and short positions independently.
Average Position Length	The average length in candlestick periods of a position during the period.

To compute the results across the complete backtest, we implemented an addition method as part of `results.py` class that adds two result objects together. Care had to be taken when doing this as adding results obtained across different period lengths could produce erroneous reports. For example, adding the un-annualised rate of return for two different period lengths is not a valid metric.

Note. When reporting the results, we needed to choose which currency units we would use, or which *numéraire* to reckon with. We chose to simply use the currency that the price was indicated it within each dataset. This did mean that we could not compare profits and losses directly between datasets due to the different units being used but we could still compare the rates of return which are the most relevant metrics anyway. Alternatively, if one wanted to reckon all in one currency, they would have to use the exchange rate between the currency the prices were quoted in and the currency they wished to report in. This extra conversion means that ones profits are now dependent on two varying exchange rates, affecting the results of the backtesting system.

The backtesting system above provided us with a suitable means of evaluating the performance of our algorithms, reasoning about why they might have failed in certain situations and improving them based on this.

4.1.4 Volatility Breakout Model

To improve our understanding of current models that are used and to test our backtesting system, we initially implemented a financial model that is based upon the volatility of past days. Commonly referred to as a *volatility breakout model*, it operates on the idea that, based upon the previous closing value and range of the previous periods, if an upward or downward trend reaches a certain point it will most likely continue further. Mr Kamran Usmani from the Swiss Finance Corporation provided us with a specification of one of the models that he has developed and we then implemented it ourselves and tested it using different parameters.

By examining the ranges[4] of the previous days and producing a weighted average for this, we calculate the levels at which we should set buy and sell orders as well as take profit and stop loss levels. The parameters involved in this process include the number of days to look back at and the weighting to apply to each of the days to calculate a weighted average range. We also specify parameters that dictate how much trend movement we want before we reach a buy or sell level as well as how much extra movement we want before we reach the take profit level.

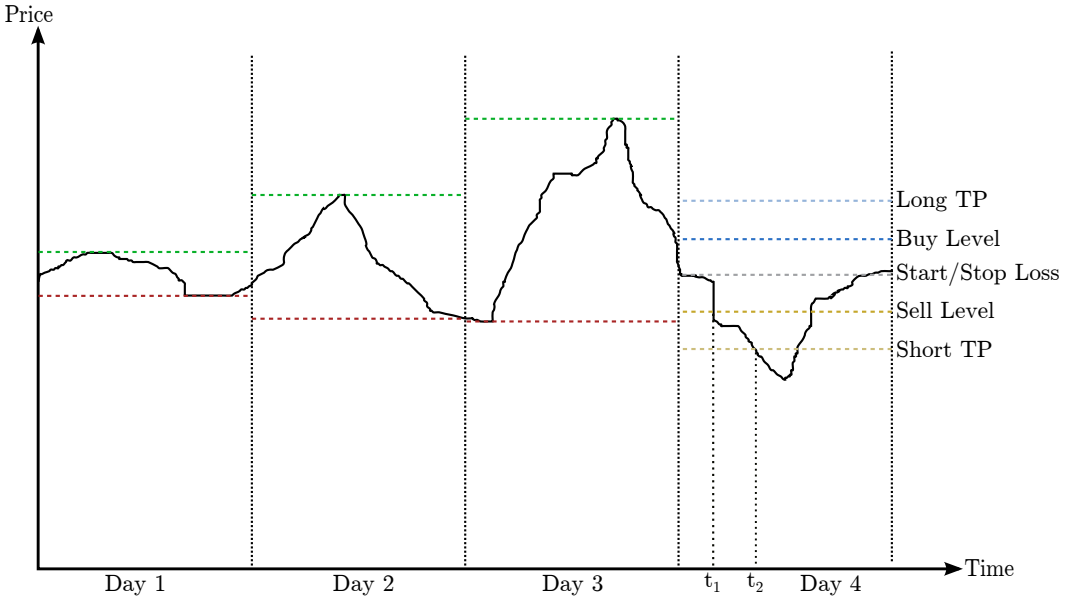

Figure 4.2: An example of the appropriate levels set by the volatility breakout model.

The example situation in 4.2 allows us to outline this approach. On day 4, the model looks back at the high-low range of each day over the past 3 days in this example, we indicate high levels by a green line and low levels as a red line. It may then choose to assign a weighting system such as weighting the range of the most recent past day 3 times as much as the others. To calculate the buy level, shown as the dark blue line, we add some multiple of this range onto the previous close and similarly for the sell level, the dark orange line, we subtract the same multiple. If either of these levels is reached, we set a take profit level as

[4]The difference between the high and low price

some other multiple of the range added to or subtracted from the previous close indicated by the lighter blue/orange lines. We also set a stop loss level at the previous close shown as the grey line, effectively placing a one cancels the other order. We can see that at t_1, we hit the sell level and so we entered a short position. Later, at t_2, we reach the short take profit level and so we secure our profit.

Our RL approach took inspiration from the idea of looking back over a given previous number of periods and weighting them differently based on how recent they are. We used this idea in our candlestick history as a way of summarising the price movements over the past set of periods. However, rather than working across whole days like the volatility breakout mode, the reinforcement learning algorithm that we proposed operates on much shorter periods, typically measured in minutes and is designed to capture smaller price movements.

4.1.5 LSTD Model

To produce the LSTD models, we implemented the descriptions of the algorithms described in the approach in python. We separated out the model and the LSTD training algorithm so that we could experiment with the reinforcement learning system without having to use the full model. We represented a state as an abstract class that required the following methods to be provided.

- `feature_state_vector(...)` - Returns a Numpy feature vector to represent the given state. Provides the necessary parameters, such as the current price, that are needed to compute this.
- `feature_action_vector(action, ...)` - Returns a Numpy feature vector to represent the given state-action pair. Also provides any parameters required to compute this. Adding this method allowed us to easily extend the basic LSTD framework to LSPI.
- `available_actions()` - Provides a list of possible actions for the current state which the agent can then choose between.
- `advance(action, ...)` - Given an action and a list of required parameters, advances the current state by executing the action, returning the new state and reward signal generated.

Note. To represent the actions, we simply used a Python enumeration object as it allowed us to compare actions for equality easily which is all that we required.

Using the above definition of a state, we created the position states that we required, specifically the idle, long position and short position states which contained all of necessary details of the position and data history. Using this, we created the `LSTDTrainer` class which provides the mechanism to perform policy iteration. This class is where the core LSTD algorithms described above were implemented. We then used the generic model class that we defined when creating the backtesting strategy to execute the trading systems using the optimal policy as trained by the LSTD system.

The `LSTDModel` class would take as an input parameter, a trained `LSTDTrainer` system for that data set which would allow it to use the optimal policy. Upon the method `receive_candle(c)` being called, the `LSTDModel` would choose the optimal action based upon the current state that it was in and the current price given by the most recent candlestick. Based upon this optimal action, the model would initiate or close any positions as required and then update its current state to reflect the position changes and candlestick history to include the received candlestick. This process continues as long as candlesticks are being received for the current day. Once the day ends, the model resets its current positions ready for the new trading day, meaning that any open positions are closed, the profit or loss calculated and provided to the agent as a reward signal. This means that until a sufficient number of candles for the new day have been received so that the model can build up the required candlestick history, the model does not trade. We stored the current state using the same state framework as the one defined for the LSTD training. This model could then be used in the backtesting system to report the results. To ensure that we were not over-fitting our data, we split each data set equally in to two consecutive halves. We then trained the `LSTDTrainer` on the first half and then obtained results for the data set by testing the `LSTDModel` using the second half. We demonstrate the relevant classes used and the order in which they are accessed using a sequence diagram in figure 4.3.

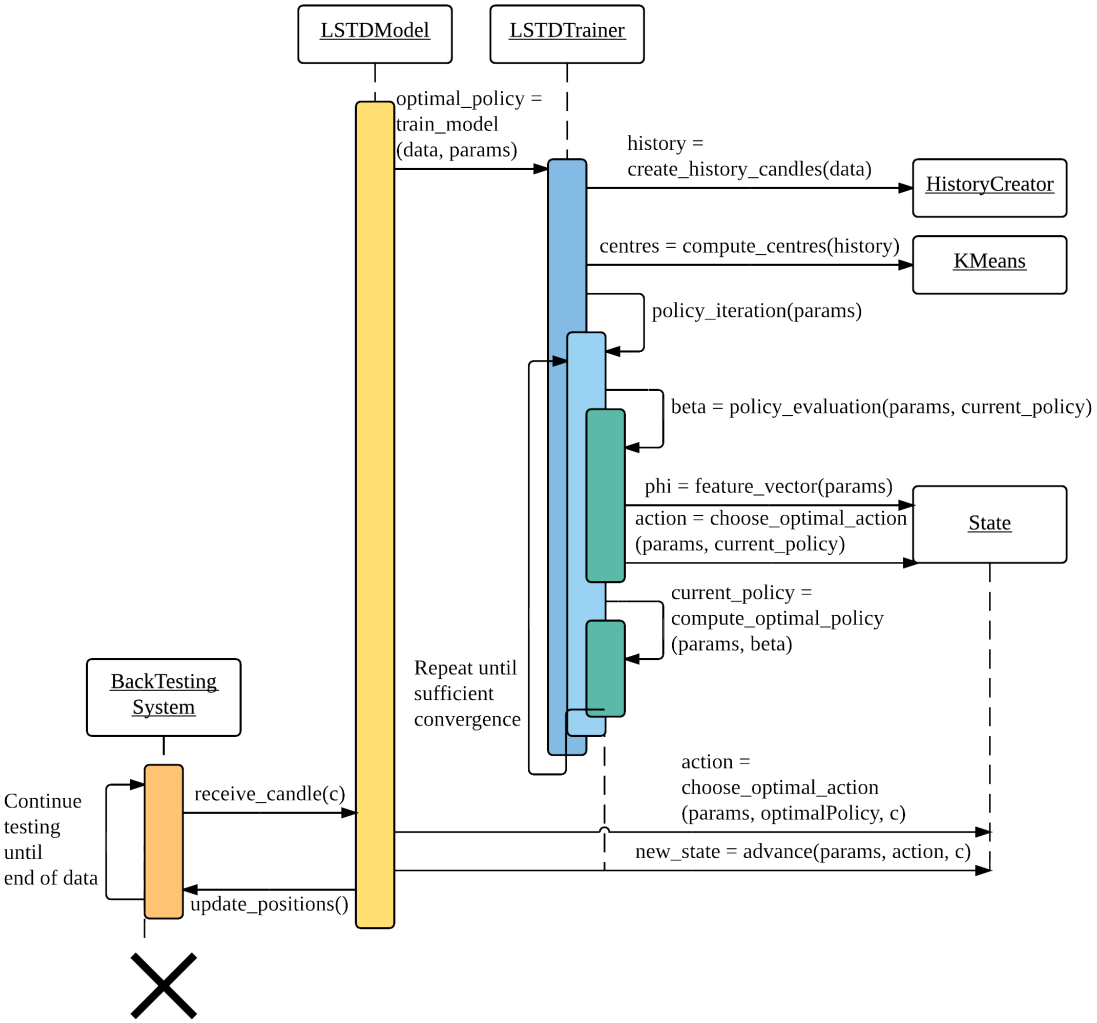

Figure 4.3: A sequence diagram for the LSTD model.

Reinforcement Learning Libraries

The above implementation describes a self contained reinforcement learning system that we built from scratch ourselves. Reinforcement learning libraries do exist in python, a popular example being *PyBrain*[5] which provides implementations of several machine learning methods. However, we felt that it could be challenging to modify a purpose built library to fit within our algorithmic trading model structure. We therefore chose to create our own reinforcement learning system as documented above that provides implementations of LSTD and LSTDQ policy iteration methods, complete with the methods required by algorithmic trading models to query them. We also feel that whilst the framework that we have created may not be suited for all reinforcement learning problems, for those within the algorithmic

[5]PyBrain Machine Learning Library - `http://pybrain.org/`

trading domain it is well suited. For example, if one wanted to add new data to a state, such as upcoming economic events that could lead to changes in the market's volatility, and then handle this within the LSTD model, they could easily do so using our system.

4.2 Performance of LSTD Methods

We now present the results of our trading algorithm as well as the justification behind our choice of parameters for both of the methods used.

4.2.1 LSTD(λ) Results

Currency Pair	Annualised Return Rate (%)
Euro - Pound	0.00250
US Dollar - Canadian Dollar	0.000715
US Dollar - Swiss Franc	0.00296
US Dollar - Japanese Yen	0.00335
Euro - Canadian Dollar	0.0281
Euro - US Dollar	0.0164
Euro - Australian Dollar	0.0228
Pound - US Dollar	0.00363
Euro - Japanese Yen	0.00229
Australian Dollar - US Dollar	0.0118
Pound - Japanese Yen	−0.00230
Total	0.00839

4.2.2 LSPI Results

Currency Pair	Annualised Return Rate (%)
Euro - Pound	0.0000316
US Dollar - Canadian Dollar	0.0639
US Dollar - Swiss Franc	−0.00122
US Dollar - Japanese Yen	−0.000307
Euro - Canadian Dollar	−0.000122
Euro - US Dollar	0.000564
Euro - Australian Dollar	0.0000658
Pound - US Dollar	−0.00000851
Euro - Japanese Yen	0.000144
Australian Dollar - US Dollar	0.0493
Pound - Japanese Yen	0.000533
Total	0.0103

The results above were taken as an average over a series of tests for each of the least-squares temporal difference learning algorithms. We used the following parameters when testing.

Parameter	Values
Number of Gaussian RBF centres	8
Width of Gaussian RBFs	5
Reward discount factor	1
History structure	[3,2,1,1,1]
Lambda	0
Exploration epsilon	0.05

For each of the algorithms, we were able to generate a profit and although this is relatively small, through the use of margin it could potentially be magnified. We can see that the LSPI algorithm marginally outperformed the LSTD(λ) algorithm, potentially justifying the advantage of LSTDQ over LSTD policy iteration. Whilst the fact that we have generated a profit is a positive sign, its magnitude is not quite as high as we would have hoped for. Following the justification of parameter choices below, we analyse some of the issues we face that may have influenced the results included.

4.2.3 Parameter Selection

Referring back to the parameters that we identified within the approach, we now justify the choices we made in terms of their values.

Number of Gaussian RBF centres

Increasing this value significantly increased the length of the feature vector but provided us with more central candlestick histories to compare a particular history to. We reasoned that there should be sufficient candlesticks to represent the possible trends that could be present in the candlestick history. We therefore chose this value to be at least as large as the number of candles within the history and to keep the size of the feature vector small, we typically selected it to be equal.

Width of Gaussian RBFs

We wanted to define the Gaussian RBF widths such that each candlestick could be strongly associated to one of the centres. We therefore did not want the value to be too small as otherwise a candlestick history would associate largely with each centre. Alternatively, if the value was too large then it is likely that a candlestick history would not associate with any of the centres. By examining the associations of individual histories with defined centres,

we found that for the majority of our datasets, a width of between $5 - 10$ suitably assigned the histories to specific centres, which would allow the agent to distinguish patterns in the history.

Reward discount factor

As we defined the task to be episodic, we were able to simply take the discount factor to be equal to 1 and value all rewards equally regardless of their position in time. We did experiment with using values smaller than one to see if the agent performed better if it was short-sighted. We had to make sure that the discount factor was large enough so that the reward received when closing a position would be correctly attributed to the action that opened it, especially for large position lengths of the order of 100 time epochs.

History Structure

When deciding the structure to impose upon the candlestick history, we wanted to ensure that we covered a large enough range such that sufficient price movements were captured whilst also allowing the agent to spot the short term trends that we desired. We experimented with different weightings that the agent could apply to the candlesticks and taking inspiration from the volatility breakout model, we attached larger weightings to more recent days. We typically looked back between 7 and 15 candles in the past as any less than this would not provide enough information to the agent and storing more than this would make it difficult for the agent to recognise individual short term trends.

Lambda

When experimenting with the LSTD(λ) proposed by Boyan, we examined how different choices of λ affected the agent's performance in the hope that it would encourage the agent to assign appropriate value to all states leading up to reward. However, we could not identify any significant changes in the policies selected or the computational performance. Additionally, Lagoudakis and Parr suggest that the use of $\lambda > 0$ in the LSPI algorithm will lead the agent to produce inaccurate estimates for actions other than the ones that it takes. For the results above, we therefore chose to restrict ourselves to the case of $\lambda = 0$ [27].

Exploration Epsilon

We found setting the exploration rate to be a challenging aspect of our evaluation. We needed to trade off between experimenting with different position types in varying situations whilst making sure that the agent maintained a position long enough to realise the rewards that it should produce. We identified some solutions to this problem and we present them in greater detail below.

4.3 Issued Encountered

As our results demonstrate, our tests have produced slight profitability through the use of our model. However, we still feel that our novel approach presents a valid proof of concept that could be adapted for future use and improved upon to overcome some of its limitations. Throughout this investigation, we encountered numerous challenges that we endeavoured to overcome. Many of these challenges were associated with our approach and we were able to experiment with alternative solutions to resolve some of these, however, we also faced issues that were a direct consequence of the methods chosen and we outline these below.

4.3.1 Reinforcement Learning Methods

A potential drawback of using the reinforcement learning methods that we chose is that being a fairly modern research topic, the methods are not quite as refined as other machine learning techniques and this made it more challenging to choose the correct parameters. Many of the techniques do not have concrete recipes on how to select the parameters and often a significant amount of trial and error is involved which can be expensive due to the curse of dimensionality. For example, choosing feature vectors that accurately represent a state is a major component of the algorithms and therefore if ours were chosen incorrectly, it could have negatively impacted the results. Lagoudakis and Parr indicate that the choice of feature vectors is vital for effective performance and state that the LSTD algorithms do not recover well from a poor choice for these. Choosing the appropriate feature vectors for a specific domain is still very much an open problem that would provide great advancements to these methods if a solution was presented [27].

Additionally, there may be RL problems for which certain methods are not as effective which have not yet been identified and our problem may perhaps have been an example of this. To complement this, we also found that it is often difficult to diagnose the errors associated with the LSTD policy iteration. Given the output β vector representing the policy, it was hard to attribute the poor performance we found from the policy to a particular aspect of the training due to the significant amount of data that was used during the policy iteration process. Additionally, given that the training algorithm is non-deterministic due to the ϵ-greedy policy and the random choice of start state for each trace, it was difficult to reproduce policy iterations to see what differences occurred when parameters were changed.

4.3.2 Choosing an Exploration ϵ

We also often found that given the initial exploration choice, the policy would become very biased towards its initial action choices. In many of our test runs, we found that the agent would continually take a long or short position throughout the test, indicating that the policy had become biased towards a particular position type. We experimented with varying the value of epsilon to larger values in the hope that the agent explores more and learns the rewards associated with both long and short positions.

However, by doing this, it led to the agent closing positions after a short amount of

time due to the large epsilon allowing it to take sub-optimal actions more often. The agent was therefore not holding a position long enough to produce a significant profit during the training process which we attributed to the choice of ϵ, the exploration rate. For a given ϵ, the length of time t such that the agent has an equal probability of choosing all optimal actions vs choosing one or more sub-optimal actions in that time period is given by (4.1).

$$(1-\epsilon)^t = 0.5 \Rightarrow t = \frac{\log(0.5)}{\log(1-\epsilon)} \tag{4.1}$$

For example, if we set $\epsilon = 0.1$ then after only 7 time epochs, there is more than a 50% chance that the agent will have explored by doing something sub optimal. This may be acceptable for reinforcement learning problems where reward signals are produced fairly often and there are a lot of states that need to be explored. In our situation however, there are only 3 classes of states and rewards are produced after holding a position for what may be a relatively long time. If the agent is currently in a position state and they execute a sub optimal closing action, then they will have sacrificed the potentially larger reward that they could have received in the future.

Initially, we suggested introducing some reward for holding a position to encourage the agent to continue maintaining the investment until it was really optimal to close it. However, as discussed with introducing the reinforcement learning problem, choosing the correct reward signals is vital for the agent to be successful. If we rewarded maintaining a position, then the agent might choose to do this indefinitely, only closing the position at the end of the day when required, which is far from the intended behaviour. We therefore felt that it was wrong to interfere with the dynamics of the model to solve this problem and examined alternatives.

An alternative solution was to re-arrange and solve (4.1) to find an ϵ given a length of time that we considered to be optimal before the agent would have a 50% chance of having executed an explorative sub-optimal action. An equation to compute this is given in (4.2).

$$\epsilon_{avg} = 1 - \sqrt[t]{0.5} \tag{4.2}$$

For example, if we felt that an optimal length of time for the agent to aim to hold a position for before closing it was 100, we could solve the above equation to get $\epsilon = 0.0069$ as the optimal epsilon. However, a problem with this approach is we have to suggest what an optimal position length could be. This could be estimated by looking at historical data to see how long the average short term trend lasted for on our time scale and using this as an approximation. Alternatively, different optimal position lengths could be tested.

Another possible method would be to use a *decaying epsilon*; an epsilon value which decays overtime making the agent more explorative in the beginning before becoming exploitative as time passes. This could be done by adapting a form of exponential decay functions to fit our domain as shown in (4.3).

$$\epsilon(x) = e^{-x/\tau}(\epsilon_{max} - \epsilon_{min}) + \epsilon_{min} \tag{4.3}$$

Given constant values for the maximum and minimum exploration rates, one could then choose a positive value τ that scales the decay function so that it produces the desired average epsilon. The average m of a function $f(x)$ over the range $[a, b]$ is given by.

$$m = \frac{1}{b-a} \int_a^b f(x)\, dx \tag{4.4}$$

Given that the dataset is made up of 1 minute candlesticks, the average trading day has 1440 candlesticks which correspond to time points. Given a desired length of time t before the agent had an equal chance of executing a sub-optimal action, one could use (4.2) to compute the average epsilon required. Solving (4.4) would find the value τ such that the agent used the desired average epsilon, as shown in (4.5).

$$\epsilon_{avg} = \frac{1}{1440} \int_0^{1440} (e^{-x/\tau}(\epsilon_{max} - \epsilon_{min}) + \epsilon_{min})\, dx \tag{4.5}$$

This equation cannot be solved using simple methods and would need to be computed numerically. This would then provide the agent with a decaying epsilon that had the average value required, determined by the desired average position length.

For example, if we set $\epsilon_{max} = 0.1$ and $\epsilon_{min} = 0.001$ and wanted an expected position length of 144, 10% of the length of the trading day, we would first calculate the required average epsilon giving $\epsilon_{avg} = 0.0048$. Solving (4.5) using the value for ϵ_{avg} gives a value of $\tau = 55.3$. We can plot this decaying epsilon graph for these parameters using Matlab to show how ϵ varies over time as shown in figure 4.4.

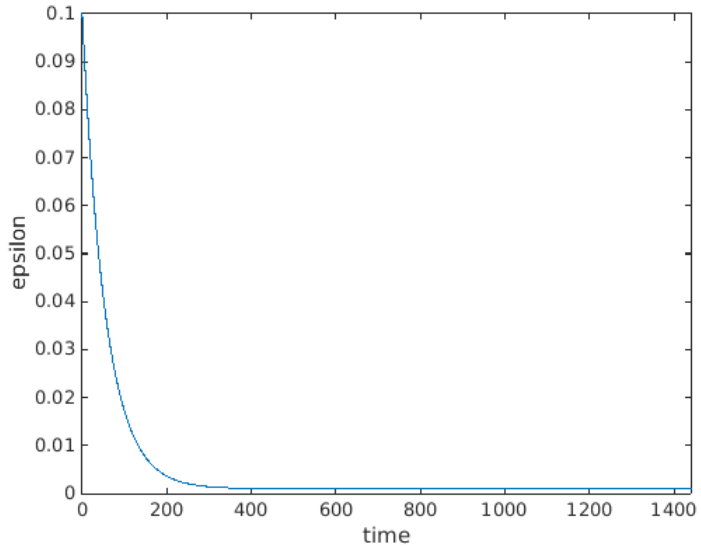

Figure 4.4: An example of the epsilon value decaying over time.

Another alternative that could also be pursued as future work would be to use adaptive control methods with a dynamic ϵ value that are based upon the value functions at each time point for a particular state. Tokic and Palm proposed value-difference based exploration combined with softmax action selection that only explores when it has insufficient knowledge regarding the environment [30]. Their results show greater efficiency compared to both ϵ-greedy and softmax policy selection methods. A drawback of this solution however is that it requires the introduction of two additional parameters, although the authors provide instructions on appropriate values for these when used in a general situation.

4.3.3 Convergence of the Optimal Policy

A significant problem that we faced was the inability of LSTD algorithms to converge to an optimal policy within a reasonable time. As outlined in the approach, each policy evaluation phase in the LSTD methods can be classified as $O(NLK^2)$ with a single $O(K^3)$ to perform the matrix inversion. Here, N is the number of traces to run, L is the expected length of each trace and K is the size of the feature vector. Given the size of our dataset, N and L were quite large and policy evaluation was therefore quite expensive for us. Also, when testing with the LSPI method, our feature vector was 4 times as large as with the LSTD(λ) method which had a significant impact on the time take to converge. Additionally, as there was a large number of possible traces, we saw large variations within our policy vector between iterations which subsequently greatly affected the value functions. This led to our policy iteration not converging and we had to increase the iteration threshold to end this process, producing a potentially sub-optimal policy.

Peter et al. suggest that when using greedy or ϵ-greedy policies with function approximation methods, small changes/errors in the value functions can lead to very large alterations to the current policy which then lead to large changes in the value function during the next iteration [31]. This can lead to oscillation within the algorithm meaning that the policy never converges to an optimal one and instead continuously varies between sub-optimal policies. This meant that often, if our policy did not converge, the training process would continue indefinitely and we would have to halt it manually, creating a huge impact on our performance.

We expect that this occurred in our problem because the length of an episode is quite long at 1440 time epochs and there are also a large number of possible episodes, one for each trading day. Therefore, when the algorithm selects a random date to start its training episode in, there is a high chance it will select one it has not used before. This will provide new data that continuously alters the value functions and the associated policy. As this continues and a sequence of traces is rarely repeated, the optimal policy oscillates. A possible solution to this would be to make the candlestick histories relative rather than exact. For example, if the past 4 price values were $(1, 1.15, 1.08, 1.2)$, we could represent this relatively as $(0, 0.15, 0.08, 0.2)$. This could possibly make the agent associate changes in the candlesticks as a relative difference even if the underlying values had different magnitudes and this could result in less changes to the feature vector within iterations.

Deisenroth and Rasmussen also suggested that reinforcement learning methods are often very inefficient when it comes to using their data during training [32]. Whilst LSTDQ claims

to use data in a more efficient manner than LSTD, it could be that neither are making good use of the provided data. They suggest that even simple problems will require an excessively large number of trials before, often of the order of 10^5 or larger, before policy iteration convergence is achieved. Given the complexity of our method, reaching this many trials was infeasible in terms of performance and we therefore had to settle for sub-optimal convergence in our policy by increasing the threshold limit used when checking convergence. They propose a novel policy search method PILCO as a solution which we discuss in the future work section.

Chapter 5

Discussion

We conclude our investigations by evaluating our success in terms of the objectives stated in the introduction and discuss the impact that these have on the fields of both reinforcement learning and algorithmic trading. We also suggest some possible ideas for extension of our approach in the form of future work.

5.1 Achievements

We have conducted a thorough investigation into the effectiveness of a reinforcement learning approach and least-squares temporal difference learning methods within algorithmic trading. Whilst the results of our trading system were not quite as profitable as we would have hoped for, we feel that there are still opportunities to be explored within this approach. We present some possibilities for extension in the following future work section. With regards to the objectives that we defined in the introduction, we have the following corresponding achievements.

- We introduced and documented a novel reinforcement learning approach to be used as the basis for an algorithmic strategy, defining the states, actions and reward signals as required for the reinforcement learning problem. As mentioned, although it did not achieve highly profitable results, we believe that it provides a viable platform and concepts which can be used by those who wish to take our work further.
- We produced a generic backtesting system for algorithmic trading models in python that allows the user to implement a simple abstract class to define a model. It can then be automatically tested in a concurrent environment across as many data sets that the user wishes to provide. Results are available for each day, for each data set and as a total return rate across all of the datasets.
- We implemented the least-squares temporal difference policy iteration algorithms in python. This includes a general reinforcement learning framework for the agent to use which, whilst currently tailored to work within the algorithmic trading domain, could be easily adapted in the future to work for a general reinforcement learning task.

5.2 Future Work

Within the evaluation, we outlined some possible solutions to some of the specific problems that we faced during the investigation, such as decaying epsilon values. Expanding upon this, we now propose some possible open future works that extend the ideas that we have discussed within this investigation and present the advantages and contributions that they could bring to our approach.

Stock Market Data

We chose to use the forex world as our financial domain as it has an in depth history of technical analysis methods being applied to it and we felt that this ability to establish clear trends would make it highly suitable for machine learning methods. However, there are plenty of algorithmic trading methods that operate using stock market data and it would therefore be natural to extend our framework to cover this. Additionally, as discovered during this investigation, many of the online data APIs or backtesting systems provide much more opportunities for stock data usage. This would make it easier to test the system and verify its performance using multiple sources. Within forex, approximately 95% of all speculative trading takes place across 7 major currency pairs and in total, only about 18 pairs are actively traded [33]. The New York Stock Exchange on the other hand has, as of June 2015, $1,867$ listings making it much broader than the forex market, even though the overall trading volumes are smaller. Additionally, when trading the stock market, many of the sectors have plenty of assets that are negatively correlated with one another. This would allow one to easily construct a balanced portfolio for the algorithm to operate on which reduces the overall risk of the system whilst producing similar return rates, a highly desirable outcome for any trading system.

Analysing Financial Events

We suggested during the evaluation that our implementation of the reinforcement learning problem allowed the information contained within a state to easily be extended. One possible extension that we alluded to was the inclusion of details of upcoming financial events that could affect not only the price of the asset but also the volatility. For example, if there was an upcoming announcement relating to the possible rises in interest rates by a country's central bank, then demand for that currency would increase, producing larger trading volumes and associated volatility. The agent would then learn how to best act given the knowledge of this upcoming information. That may involve avoiding trading due to the higher risks associating with larger volatility or it may lead to the agent taking more positions based on not only the current data signals, but also the anticipation of the effects of the events.

The representations of these events would have to be chosen carefully in order for the agent to interpret them accurately. For example, when including information regarding interest rate changes, it could be useful to provide the expected value change or even a probability distribution over the range of possible values for the model to use. Several

online sources, such as *Daily FX*[1] provide access to calendars of these events, along with indications of what values could be affected, the importance of the event and the forecasted changes. A preliminary investigation conducted by a quantitative research firm, *Deltix Lab*, concluded that using macroeconomic events within algorithmic trading strategies shows promise [34], reinforcing our proposal of adopting it into our system.

Alternatively, rather than incorporating the forecasts of financial events, one could use trending search data as proposed by Curme et al. [35]. They proposed a relationship between current popular search terms and movements within the stock market. Using historical search and stock data, they found a correlation between searches that referenced political or economic material and stock market moves that followed; downward movements were identified to occur more often than upward trends. One could therefore add popular search topics, over the past day for example, and then allow the agent to factor these into its investment choices. For a particular financial asset, the ranking of searches for its details amongst all current search terms could provide the agent with an indication of the current interest in the asset and it could learn to associate this with subsequent market movements.

Incorporating Risk Levels into the Model Dynamics

When discussing the impact of ϵ values during the evaluation, we initially proposed introducing a reward signal when a position is maintained in order to encourage the agent to hold an investment. Whilst we concluded that doing so to fix the issue of short position lengths would be the wrong approach, it could prove to be a novel way of introducing risk parameters into the model. Similar to the layered architecture proposed by Dempster and Leemans [17], the operator could provide a level of risk to the algorithm which would be incorporated into the dynamics of the model. For example, if the operator was risk averse, the model could provide a negative reward signal to the agent upon opening a position. This reward signal could be a decreasing function of the current volatility of the asset to reflect the increased risk associated with trading in the current conditions. On the other hand, if the operator was more inclined to take risks, a positive reward signal could be produced for the agent upon taking a position. This reward signal could also be influenced by other factors such as the current performance of the agent or the level of funds available to it.

One could also experiment with using different objective functions, such as maximizing the Sharpe ratio, and changing the reward signals to coincide with this. Finally, when choosing the optimal action to take, rather than simply being greedy and selecting the action with the largest expected value amongst the current possible actions, the agent could analyse the actual magnitude of the expected value and then based on the current risk level, it could then decide whether it is high enough to warrant actually executing that action. For example, if the agent was almost indifferent between the larger value of opening a position and the slightly smaller value of staying idle but the risk level of the algorithm was set to be very risk averse, then the agent could choose to stay idle anyway rather than taking the risk of opening a position based on an expected small return.

[1] Daily FX's upcoming economic event calendar - http://www.dailyfx.com/calendar

Alternative Reinforcement Learning Techniques

Actor-Critic techniques are a class of methods within the reinforcement learning that use temporal difference learning methods to update the value functions, known as the *critic* which predicts, and use stochastic gradient descent to update the policy, known as the *actor* which controls [36]. These two components are brought together to produce policy iteration. When the method was first introduced, it had limited success and often failed to converge, however, a shift from using normal policy gradients to *natural* gradients, introduced by Peters et al. as natural actor-critic, led to far more efficient results [31]. The advantage that these gradient based methods have over the value function policy iteration methods that we have used is that they are not as sensitive to small changes in the policy or value functions and therefore have stronger convergence guarantees. This could help alleviate some of the issues that we faced with regards to convergence difficulties during policy iteration, making the agent more effective at learning the market trends.

Finally, PILCO is a very recent development within the reinforcement learning domain that describes a model based policy search method introduced in 2011 by Deisenroth and Rasmussen [32]. PILCO is a model based method, meaning that it aims to learn the dynamics of the environment's model, contrasting model-free methods such as Q learning which simply seek to learn the optimal policy. Deisenroth and Rasmussen claim that many reinforcement learning methods require an unacceptably large number of trials to be used successfully, even for simple tasks and PILCO aims to fix this by using the training data much more efficiently. Their results show that even on tasks with high dimensionality, PILCO proved to be a very efficient method and was able to learn the dynamics with a fewer number of trials. This approach could help alleviate the convergence issues that we found as well as reducing the number of trials needed to converge, considering that performance had a major impact on our approach and was a major factor in the results that this provided us with.

Bibliography

[1] Investopedia. (2010). The forex market tutorial, [Online]. Available: http://i.investopedia.com/inv/pdf/tutorials/ForexMarket.pdf (visited on 09/06/2015).

[2] A. Chaboud, B. Chiquoine, E. Hjalmarsson, and C. Vega, "Rise of the machines: algorithmic trading in the foreign exchange market. international finance discussion papers", *The Journal of Finance*, vol. 69, pp. 2045–2084, 2009.

[3] J.-M. Orlando. (2007). Algorithmic presentation to european central bank - BNP Paribas, [Online]. Available: https://www.ecb.europa.eu/paym/groups/pdf/Algo_trading_presentation.pdf (visited on 11/06/2015).

[4] BBC. (2005). Probe into japan share sale error, [Online]. Available: http://news.bbc.co.uk/1/hi/business/4512962.stm (visited on 03/06/2015).

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets", *Neural Computation*, vol. 18, 2006.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998, http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html.

[7] M. Levinson, *The Economist Guide To Financial Markets*. Economist Books, 2014.

[8] M. Goldstein. (2004). China and the renminbi exchange rate, [Online]. Available: http://www.piie.com/publications/chapters_preview/382/9iie3780.pdf (visited on 02/06/2015).

[9] The Economist. (2015). Warning: too much finance is bad for the economy, [Online]. Available: http://www.economist.com/blogs/buttonwood/2015/02/finance-sector-and-growth (visited on 10/06/2015).

[10] Investopedia. (2013). Why it's important to regulate foreign exchange, [Online]. Available: http://www.investopedia.com/articles/forex/041613/why-its-important-regulate-foreign-exchange.asp (visited on 08/06/2015).

[11] P. Rosenstreich, *Forex Revolution: An Insider's Guide to the Real World of Foreign Exchange Trading*. FT Press, 2005.

[12] Business Insider Australia. (2012). It's the last day for open-outcry trading in the intercontinental exchange trading pits, [Online]. Available: http://www.businessinsider.com.au/no-more-open-outcry-trading-in-ice-pits-2012-10 (visited on 17/06/2015).

[13] Geeksoft Technologies. (2013). Why we shouldn't ban algorithmic trading?, [Online]. Available: https://greeksofttechnologies.wordpress.com/2013/05/02/why-we-shouldnt-ban-algorithmic-trading/ (visited on 17/06/2015).

[14] K. V. Nesbitt and S. Barrass, "Finding trading patterns in stock market data", *Computer Graphics and Applications, IEEE*, vol. 24, pp. 45–55, 2004.

[15] M. Scott, "Charles Dow's six secrets to market success", *Alchemist*, vol. 30, pp. 17–18, 2003.

[16] M. Lewis, *Flash Boys: A Wall Street Revolt*. W. W. Norton & Company, 2014.

[17] M. A. H. Dempster and V. Leemans, "An automated fx trading system using adaptive reinforcement learning", *Expert Systems with Applications: Special Issue on Financial Engineering*, vol. 30, pp. 534–552, 2006.

[18] J. Moody and M. Saffell, "Minimising downside risk via stochastic dynamic programming", *International Conference Computational Finance*, vol. 6, pp. 403–415, 1999.

[19] X. Du, J. Zhai, and K. Lv, *Algorithm trading using q-learning and recurrent reinforcement learning*, 2009.

[20] S. Shen, H. Jiang, and T. Zhang, *Stock market forecasting using machine learning algorithms*, 2012.

[21] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[22] R. S. Sutton, "Learning to predict by the methods of temporal difference", *Machine Learning*, vol. 3, pp. 9–44, 1988.

[23] J. A. Boyan, "Least-squares temporal difference learning", *Machine Learning*, vol. 49, pp. 233–246, 2002.

[24] H. Markowitz, "Portfolio selection", *The Journal of Finance*, vol. 7, pp. 77–91, 1952.

[25] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning", *Machine Learning*, vol. 22, pp. 33–57, 1996.

[26] R. Penrose, "A generalized inverse for matrices", *Proceedings of the Cambridge Philosophical Society*, vol. 51, pp. 406–413, 1955.

[27] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration", *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[28] M. J. L. Orr, *Introduction to radial basis function networks*, http://www.anc.ed.ac.uk/rbf/rbf.html, 1996.

[29] M. Halls-Moore. (2013). Best programming language for algorithmic trading systems?, [Online]. Available: https://www.quantstart.com/articles/Best-Programming-Language-for-Algorithmic-Trading-Systems (visited on 01/06/2015).

[30] M. Tokic and G. Palm, "Value-difference based exploration: adaptive control between epsilon-greedy and softmax", in *KI 2011: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, vol. 7006, Springer Berlin Heidelberg, 2011, pp. 335–346, ISBN: 978-3-642-24454-4. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24455-1_33.

[31] J. Peters, S. Vijayakumar, and S. Schaal, *Natural Actor-Critic*. 2005, pp. 280–291. DOI: 10.1007/11564096_29.

[32] M. P. Deisenroth and C. E. Rasmussen, "Pilco: a model-based and data-efficient approach to policy search", *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[33] Investopedia. (2010). Most traded pairs, [Online]. Available: http://www.investopedia.com/walkthrough/forex/getting-started/pairs.aspx (visited on 14/06/2015).

[34] I. Gorelik. (2014). Using macroeconomic events in an automated fx trading strategy, [Online]. Available: https://www.youtube.com/watch?v=prtNdzAf9lw (visited on 14/06/2015).

[35] C. Curme, T. Preis, E. Stanley, and H. S. Moat, "Qauntifying the semantics of search behaviour before stock market moves", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 111, pp. 11 600–11 605, 2012.

[36] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Incremental Natural Actor-Critic Algorithms", in *Neural Information Processing Systems*, 2007.