Dissertation Type: enterprise

DEPARTMENT OF COMPUTER SCIENCE

The Deeply Reinforced Trader

Ashwinder Khurana

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Friday 7th August, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Ashwinder Khurana, Friday 7th August, 2020

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Executive Summary

One of the primary goals of artificial intelligence is the solve complex tasks that rely on highly dimensional, unprocessed and noisy input. Namely, recent advances in Deep Reinforcement Learning (DRL) has resulted in the "Deep Q Network [**?**], which is capable of human level performance on several Atari video games - using unprocessed pixels for input. A similar "gamification" of the process of a continuous double auction limit-order-book (LOB), can be an application of DRL.

DRL requires for an agent to interact with an environment and to derive actions from the state of the environment at each timestep. The environment in this case can be a market session, in which several sales traders interact and utilise the LOB to communicate orders to buy or sell an asset, and the agent that interacts with the environment is the proprietary trader, which aims to seek profit for itself rather than on the behalf of a client. Information available from the LOB can be used as combinations of inputs to represent the current state of the 'game', from which the trader can learn which actions to take during the learning process. This is done by using a policy which determines what action to take depending on the current state of the LOB and a reward system which provides rewards depending on the action.
Falcao had used a Policy Gradient agent (PG Agent) in conjunction with Cliff's ZIP to solve the sales trader problem. In comparison this thesis attempts to solely use state of the art DRL techniques to solve the proprietary trader problem, namely via models such as Advantage Actor-Critic, Policy Gradient and DDPG.

Whilst most algorithms are trained and analysed with data from public exchanges collected from sites such as finance.yahoo.com, this thesis focuses on using Cliff's Bristol Stock Exchange [**?**] and Falcao's Bristol Stock Gym [**?**] to simulate market sessions of robot traders to train and compare the effectiveness of the agent against other traditional machine learning based algorithms.

This is a hugely challenging task as the mapping between the current state of the system and the action the agent takes in such an environment is a problem all financial institutions are aiming to tackle in the most sophisticated ways to gain an overall competitive advantage. The results of this thesis are not a success story, but it details the journey of iterating and analysing results to create a profitable trader using the techniques outlined above and the vast challenges that were presented.

> My research hypothesis is that the latest developments in machine learning, more specifically, a trader that uses Deep Reinforcement Learning techniques alone provide a tool set to create a proprietary trader that outperforms humans.

This project explores the hypothesis

- I spent 100+ hours researching prior work on the field and learning about Deep Reinforcement Learning techniques.
- I wrote a total of ? lines of source code, comprising of the merger of capabilities of BSE into Falcao's BristolStockGym, several DRL models, Variational Auto-Encoder and Regression Models.
- Spent 100+ hours training the various models and running experiments on Blue Crystal 4 [**?**]

Supporting Technologies

- All the codebase is written using Python version 3.6.7 and the Python Package Installer PIP tool in its 19.0.3 version.
- Several Python libraries were used extensively throughout the project, namely: PyTorch(1.3.0), NumPy(1.18.1), Matplotlib(3.1.3), Sklearn(0.22)
- I used a Falcao's version of Cliff's Bristol Stock Exchange that is refactored to be compatible for Deep Reinforcement Learning analogous to OpenAI's Gym

Notation and Acronyms

All of the acronyms used in this document are explained on their first usage and no more. However, several acronyms are used across multiple chapters throughout the thesis, they are as follows:

Acronyms

AI	:	Artificial Intelligence
BSE	:	Bristol Stock Exchange
BSG	:	Bristol Stock Gym
LOB	:	Limit Order Book
CDA	:	Continuous Double Auction
MDP	:	Markov Decision Process
DRL	:	Deep Reinforcement Learning
PG	:	Policy Gradient
DDPG	:	Deep Deterministic Policy Gradient
ZI-C	:	Zero Intelligence - Constrained
ZIP	:	Zero Intelligence Plus
DRT	:	Deeply Reinforced Trader
	⋮	
$\mathcal{H}(x)$	:	the Hamming weight of x
$\mathbb{F}_q$	:	a finite field with q elements
x_i	:	the i-th bit of some binary sequence x, st. $x_i \in \{0, 1\}$

Acknowledgements

An optional section, of at most 1 **page**

It is common practice (although totally optional) to acknowledge any third-party advice, contribution or influence you have found useful during your work. Examples include support from friends or family, the input of your Supervisor and/or Advisor, external organisations or persons who have supplied resources of some kind (e.g., funding, advice or time), and so on.

Chapter 1

Contextual Background

1.1 Introduction

The underlying context behind this thesis is the comprehension, analysis and exploitation of the mechanisms that underpin the real-world financial markets, from a Computer Scientist's perspective. The infrastructure and mechanism of interest are major stock exchanges. Stock exchanges are where individuals or institutions come to a centralised market to buy and sell stocks, where the price of these stocks are set by the supply and demand in the market as the buyers and sellers place their orders on the exchange. There are various types of orders that can be used to trade a stock, however, this thesis focuses on a specific type of order, the Limit Order. A limit order is a type of order to purchase or sell a security at a specified price or better. For buy limit orders, the order will be executed only at the limit price or a lower one, while for sell limit orders, the order will be executed only at the limit price or a higher one [**?**]. A collection of orders of this kind formulates the Limit-Order-Book (LOB) , which is a book, maintained by the exchange, that records the outstanding limit orders made by traders.

The role of a sales trader is to trade on behalf of a client, which means that they do not own a stock of their own, but instead buys and sells to secure the best price for their client. A proprietary trader, or a "prop" trader on the other hand buys and sells stocks on their own behalf with their own capital with the aim of seeking profitable returns. Both types of traders have to constantly adapt to the market conditions and react through the prices they quote for their orders.

Originally, the job of trading was fulfilled by humans who reacted to the market conditions using their intuition and their knowledge of the financial markets, however the job was hit with a wave of automation after 2001, when IBM researchers published results that indicated that two algorithms (ZIP by Prof Dave Cliff and MGD, a modification of GD by Steven Gjerstad & John Dickhaut) consistently outperformed humans in a experimental laboratory version of financial markets [**?**].

Since then, there has been relatively little research done in this field within academia, and subsequent solutions have built upon the traditional heuristic alongside 'old school' machine learning approaches. Considering the explosion of Deep Learning as a field in this last decade, this thesis attempts to explore the application of these state-of-the-art techniques, in particular Deep Reinforcement Learning, to the problem of the adaptive prop trader.

1.2 History of Research

1.2.1 Experimental Economics to Algorithmic Trading

Analysis of the sales trader problem can be seen as early as the 1960's in Vernon Smith's experiments at Purdue University, which led to the birth of the field of Experimental Economics and a Nobel Prize for Smith.

Smith's motivation behind these experiments was to find empirical evidence for the credibility of basic microeconomic theory of Continuous Double Auction's (CDA's) to generate competitive equilibria [**?**] . He formalised this experiment structure as a Double-oral-Auction in which he would a split a group

of humans up into two subgroups: Buyers and Sellers. He gave each human a private limit price, where buyers would not buy above that price and sellers would not sell below that price. Prices quoted by the buyer's are known as the bid prices and the prices quoted by the sellers are known as the ask prices. The buyers and sellers would interact within small time based sessions, otherwise known as trading sessions.

After running repeated experiments of the trading sessions, Smith empirically demonstrated that the prices quoted by the traders demonstrated rapid equilibration, where the prices would converge to the theoretical 'best' price of an asset, despite a small number of traders used.

These sets of experiments were repeated by Gode and Sunder [**?**] in the 1990's. Crucially however, the human traders that were the core of Smith's experiments were replaced with computer based algorithmic traders. Gode and Sunder introduced two types of *zero-intelligence* algorithmic traders for CDA markets: Zero Intelligence - Unconstrained (ZI-U) and Zero Intelligence - Constrained (ZI-C). ZI-U traders are traders that generated a uniformly random bid/ask price, whilst ZI-C traders were traders that also generated a random price, however it was constrained between the private limit price of the trader and the lowest/highest possible price w.r.t to whether it was a buyer or seller - therefore it could never sell at a loss.

The purpose of Gode and Sunder replicating the experiments by Smith with their adaptation from humans to computers was to figure out if the allocative efficiency of a market was down to the intelligence of the traders or the organisation of the market. They ran several experiments with 5 different types of market structures whilst monitoring their allocative efficiency. They concluded that the ZI-U traders were essentially useless in that they did not gravitate to an equilibrium price, however, the surprising result was that the ZI-C traders were surprisingly human-like as the prices quoted by these traders eventually did gravitate to the equilibrium price over the course of each trading session. This allowed them to conclude that most of the 'intelligence' was in the market structure rather than the traders.

This conclusion was critiqued by Cliff in 1996 [**?**] hypothesised that the ZI-C traders would fail to equilibrate under a set of market conditions and them empirically demonstrated that they did indeed fail to equilibrate. He also developed a new algorithmic trader coined as zero-intelligence plus (ZIP), which quoted prices based on a profit margin the algorithm adjusts using a learning rule (Widrow-Huff with momentum). This algorithm succeeded in the market conditions in which ZI-C failed in.

Around the same time, in 1998 Gjerstad and Dickhaut developed the GD algorithm [**?**]. The core of this algorithm is that the traders compute a belief function that an order will be accepted by the opposing trader types (buyer vs seller) based on the information extracted from the market. This belief function seeks to maximise the trader's expected gain over the course of the session.

Prior to 2001, these algorithms had solely been pit against other algorithms, however in 2001, Tesauro and Das from IBM [**?**] ran numerous experiments where they would pit algorithms like ZIP, MGD (a modified GD) against humans as well as agent vs agent trading contests. The results of these experiments were groundbreaking as they demonstrated that ZIP and MGD were dominant in the agent vs agent contents, but more crucially their performance surpassed that of human traders.

Despite the promising results and the wide press received from the IBM publications, relatively little experimentation has been conducted of this type. Meanwhile, Tesauro and Bredin published an improved version GD called GD-Extended (GDX) in 2002, and Vytelingum published the Adaptive-Aggressiveness Algorithm in 2006 [**?**].

1.2.2 Developments in Artificial Intelligence

Despite the the longstanding theory behind Deep Learning, it has been an explosive field in the last decade, with notable achievements including: enabling google translate to become significantly more accurate, furthering NLP research massively and key factor behind the driverless car. It has been labelled the 'Deep Learning Revolution' by Terry Sejnowski [**?**].

Similarly, Reinforcement Learning has been an idea coined for a long time[**?**], however it was applied to very limited and relatively simple problems.

The two ideas silo'd have had their respective achievements, however recently there have been unprescedented achievements in the field of artificial intelligence with the combination of the two fields predictably called Deep Reinforcement Learning. Some notable achievements including Robotics [**?**] that maps raw input pixels to actions from a robot and AlphaZero from Google-owned startup DeepMind which is a Deep Reinforcement Learning successor to their own AlphaGo, who beat the world champion at the board game 'Go'.

1.3 Motivations

The motivations behind this project are three fold.

1.3.1 Finance perspective

The first motivation is in the fact that any project that invents or improves upon current solutions within this domain is solving an extremely complex problem in the world of Finance.

Huge institutions such as banks and hedge funds are at a constant arms race with one another to see who can create the most robust, sophisticated and profitable algorithms to enact their trades, thus an arms race for keeping their clients and staying in business. In doing so, these institutions are hiring the most able Computer scientists, and Quantitative developers to create optimal solutions in house in order to deliver this competitive edge against the rest of the market.

The heavy investment in this jobs and the development of trading algorithms comes from the big Financial institutions relatively late adoption of surging technologies, mainly driven by small FinTech companies eating from the established companies' profits with their innovative tech solutions to an otherwise very traditional industry. MX reports [**?**] that as an example, a breakdown of JP Morgan's revenue indicates that roughly 15% of JP Morgan's interest net income ($9b) stems from their trading of assets and investing securities, which highlights their interest in optimising the entire process of trading.

The aforementioned IBM experiment 1.2.1 also indicated that the ZIP and MGD algorithms outperformed their human counterparts in the sales trader job, albeit in a much more simplified environment, the claim still holds true for real world applications where algorithmic approaches outperform the previously hired traditional human trader.

There are several reasons for this, as computers are able to outperform a traders intuition and can analyse large sets of data very quickly to produce fast and appropriate action. Another aspect is more so to do with human psychology. Research indicates[**?**] that trading and money management is akin to human survival, which directly affects the part of the brain called the amygdala. This is an evolutionary knee-jerk reaction that triggers pain or pleasure. A successful trade creates a huge sense of euphoria, but a bad trade equally creates a sense of pain, regret and frustration. This can lead to inappropriate action taken from the human trader, for example revenge trading. The objective perspective from an individual not involved in trading suddenly becomes hard to translate when there is a stake involved whilst trading when the animalistic sense of survival kicks in, leaving very little room for subjectivity. This aspect of human psychology makes it more desirable for firms to trust algorithms that are not prone to this downfall. This has resulted in 35% of the U.S's equities being managed by automatised processes [**?**].

On the flip side, despite the removal of human error, Torsten Slok, chief international economist at Deutsche Bank, has named it the number 1 risk to markets in 2019[**?**]. This is down to the potential of concentrated sell offs causing a downward spiral of markets. This is a credible argument made and raises questions for the future of AI within finance and its integration within society as whole.

Despite this, the reliance on state of the art technology and research to propel financial services into the fifth industrial revolution is undeniable which presents an opportunity for curious computer scientists and academics to tackle these complex problems.

1.3.2 Computer Science perspective

As mentioned in section **??** DRL has recently scaled to previously unsolvable problems. This is down to the ability of DRL to learn from raw sensors or input images as input, without the intervention of human, in an unsupervised machine learning manner.

AlphaGo Zero surpassed its predecessor AlphaGo in an unprecedented manner because the knowledge of the AI is not bounded by human knowledge[**?**]. This is achieved by an initialised neural network that has no knowledge about the game of Go. The AI then plays games against itself via the combination of its network plus a powerful search algorithm. As more iterations of the games are completed, the network is tuned and better predictions are made. This updates the neural network and combined with the search algorithm to create a more 'powerful' version of itself, a new AlphaGo Zero. This process repeats to continuously improve its performance. AlphaGo in the other hand, used a more traditional approach to learn how to play the game, because it "watched" many different exemplar games and learnt based in a supervised machine learning manner.

AlphaGo Zero differs from its predecessor in many ways. The most apparent difference is that rather than the hand-engineered feature extraction process that AlphaGo used, AlphaGo Zero used the raw images of the black and white stones from the Go board as its input. Also, AlphaGo used a more traditional approach to learn how to play the game, because it "watched" many different exemplar games and learnt based in a supervised machine learning manner, whilst AlphaGo Zero learnt by playing itself.

Whilst AlphaGo Zero as an example is a holy grail within the DRL field, the potential for this process to applied to varying problem is one of the key motivations behind trying to apply the technique to this problem domain.

1.3.3 Furthering previous work

Falcao's thesis [**?**] also alluded to the excitement of applying DRL techniques to this domain, but after proving unsuccessful, opted to direct his attention to combining DRL techniques with existing more traditional techniques.

Furthermore, Falcao built an excellent framework BSG (Bristol Stock Gym) and interface to train DRL within the problem definition as it is a framework that is refactored from Cliff's Bristol Stock Exchange to simulate trade experiments as well training the network whilst iterating through episodes of the 'game' which are analogous to market sessions in BSE.

Whilst Falcao's thesis provides a very good overview of his journey exploring DRL, this project focuses on applying several more techniques within the field to see if they alone can solve the problem of a prop trader

1.4 Central Challenges

The challenges in this project are vast and unsolved for this domain.

One of the main challenges is to extract the key information available from the LOB at every time step. The word 'key' is ambiguous here because that is a challenge within itself, to decide what is useful information for the networks to use as input.

Another challenge is to design a network architecture that is fit to tackle the prop trader problem, including all its hyperparameters and model layer choices. This is a common problem within machine learning, and despite resources being available as well as exemplar material, the choices made w.r.t to

these factors have to be bespoke to the data available from BSG, which would take a considerable amount of time experimenting and training.

Research indicates that one of the most complex aspects of DRL is to create the optimal reward function so that the DRL agent learns the desired behaviour, an example is in the OpenAI gym environment 'Lunar Landing' game, in which the agent has to land the spacecraft safely. A well intentioned reward system may include punishing the agent with negative rewards for landing the spacecraft incorrectly, however, can result in the agent not wanting to land the spacecraft at all, which is not desirable behaviour. This raises in question the reward function itself as well as the choices behind the magnitude of reward for each action.

1.5 Conclusion

The high-level objective of this project is to use state-of-art Deep Reinforcement Learning techniques to develop an adaptive proprietary trader. More specifically, the concrete aims are:

1. To develop a concrete understanding of Deep Reinforcement Learning techniques from scratch as they are not a core part of the university's machine learning curriculum.
2. To develop a feature extraction mechanism to extract the most crucial information available from the public market session at every time step.
3. Experiment and explore different DRL models with a varying set of hyper-parameters and layer choices
4. Most importantly, develop the optimal reward function for the agent to exhibit the most desirable behaviour

Chapter 2

Technical Background

2.1 The Overarching Problem

2.1.1 The Limit Order Book

The Limit Order Book (LOB) is a standardised view of all relevant/key market data within a Continuous Double Auction (CDA), and they serve as the backbone for most modern electronic exchanges within financial markets. It summarises all the live limit orders that are currently outstanding on the exchange with bid orders – offers to buy an asset, and ask orders – offers to sell an asset, both displayed on either side of the LOB. Each side is called a book, ergo the bid book is on one side, and the ask book is on the other.

Figure 2.1: A graphical representation of a Limit Order Book (LOB), reproduced from Cliff (2018)

The left hand side of figure ?? is a mockup of a graphical representation of BSE, whilst in its original form, is in a python dictionary data structure. As seen in the figure, the bid book lists all the bid orders with their prices and the respective quantities available at that price as a key/value pair in descending order to represent all the outstanding orders from the buyers. Conversely, the ask book lists the price-quantity pair for all the ask orders in ascending order. In this order, the highest bid price is at the top of the bid book alongside the lowest ask price: the two prices are called the *best bid* and the *best ask* respectively.

This graphical representation allows for easy feature extraction for some key elements that may not explicitly stated in the figure.

- The *Time Stamp* for the current time in the trading session, shown post-facing the "T" in the top left corner
- The *Bid-Ask Spread* which is the difference between the *best ask* and the *best bid*
- The *Midprice* is the arithmetic mean of the *best ask* and the *best bid*

- The *Tape* shows a record of all the executed trades and cancellations of orders
- The *Microprice* is a cross volume weighted average of the *best ask* and *best bid*, prefaced by the green "M" below the time stamp
- The latest limit order sent to the exchange by a trader

The *midprice* and *microprice* are values that are used to approximate the value of the asset and attempt to summarise the market. In this example snapshot figure ?? the *best ask* is \$1.77 and the *best bid* is \$1.50 so the midprice is (\$1.77 + \$1.50)/2 = \$1.66. The *microprice* is a more intricate calculation because it is a cross volume weighted average:

$$\frac{BestBidQty * BestAskPrice + BestAskQty * BestBidPrice}{BestBidQty + BestAskQty} = Microprice \tag{2.1}$$

$$\frac{5 * \$1.77 + 13 * \$1.50}{5 + 18} = \$1.58 \tag{2.2}$$

The right hand side of the figure represent the supply and demand curves curated from the ask and bid books respectively, where the orange line is the demand curve and the blue line is the supply curve. Following the orange or blue step functions from left to right, at each step, the height represents the price and the width represents the available quantity at the price point. Lastly, the green cross represents the microprice.

A trade occurs when a bid offer's price is greater than the *best ask* or when an ask offer's price is less than the *best bid* on the LOB, this is known as *crossing the spread*. This process is quantity invariant, that is if the latest order price crosses the spread but the requested quantity exceeds the quantity available at that price, all the quantities that are available are sold crossed price, and the unfulfilled quantities are then placed on the LOB as an outstanding order.

As seen in the figure ??, the supply and demand curves do not cross, which therefore represents the fact that no trader is willing to trade at the current prices given in the exchange. The supply and demand curves not crossing means that the LOB is currently at equilibrium, as the best bid is not greater than the best ask.

Figure 2.2: A graphical representation of a Limit Order Book (LOB), reproduced from Cliff (2018)

The figure ?? indicates that a new order has been placed to the exchange. Interestingly, this order crosses the spread because the ask price \$1.48 was less than the *best bid* in figure ?? so therefore a transaction occurs. Incidentally, the quantity requested was less than the quantity available at \$1.50 so the entire order is fulfilled, and the quantity remaining is updated from 10 to 5. For arguments sake, if the quantity requested was 20 rather than 5, all the quantities that are available at \$1.50 (10) would be used to fulfil the new order and then the LOB would have a new *best bid* of \$1.48 with a qty of 10.

2.1.2 Proprietary trader's Objective

The agents that are taking part in the 'gamification' of the trading session are sales traders. The aim of the sales trader is to maximise their consumer surplus relative to their client's order. As mentioned in the previous chapter, the sales trader does not hold a stock of their own, they trade on behalf of their *client*, who provides them with a limit price that they cannot buy or sell above or below respectively. The goal of the sales trader is to maximise the difference between the limit price and the trade price, which is the consumer surplus. In the real world, the sales trader makes commission based on the consumer surplus and the client pockets the rest.

The proprietary trader's objective is that to maximise their own capital buy buying and selling stocks for themselves. Whilst initially it may seem like the objective of the prop trader is different to the sales trader and therefore cant be compared, but the proprietary traders ability to maximise the difference between the price they buy and sell a stock at is analogous to a sales trader maximising the difference between its limit price and its trade price. This rephrasing of the problem allows this project to directly compare the traditional machine learning approaches to the prop trader developed in this project.

2.2 Reinforcement Learning

This project transforms the current Prop trader problem to a problem that is well suited to a DRL paradigm, which will be the main avenue of solutions this project explores, so it is paramount to provide the technical background for reinforcement learning, which is the basis of this project.

2.2.1 What is Reinforcement Learning

Reinforcement Learning is the training of machine learning models(agents) to take actions that interact with a complex, to maximise a a reward. More formally, the agent learns, via interactive with an environment, a policy $\pi(\alpha|\sigma)$ which is the probability to take a particular action α, after observing a state σ that maximises its reward.

Markov Decision Process

Reinforcement Learning requires a fully observable environment in which Markov Decision Processes (MDP's) are a formalisation of this environment and the problem definition as a whole, where almost all RL problems can be formalised as.

A MDP is a 5-tuple which has components that are defined as follows:

- $\Sigma \rightarrow$ The set of all states, where a state σ_t is defined as an observation from an environment at time stamp t
- A $\rightarrow$ The set of all actions possible, where an action α is any action available to the the agent in state σ_t at time stamp t
- $\Phi \rightarrow$ The set of all transition probabilities, where ϕ_{α_t} is the probability that action α_t in state σ_t at time stamp t will lead to state σ' in time stamp $t+1$. $\Phi(\sigma_t, \sigma') = Pr(\sigma(t+1) = \sigma'|\sigma_t, \alpha_t)$. This is a stochastic process because the same state-action transition may not lead to the same state', which may due to random factors in the environment or interactions with the other agents, or an inaccurate interpretation of the current state from the agent, which is suitable for this projects' environment.
- $\rho_{\alpha_t}(\sigma_t) \rightarrow$ The immediate reward after the transition from state σ_t to an arbitrary state σ'
- $\gamma \in [0, 1] \rightarrow$ is a discount factor representing the difference in importance between present and future rewards. Depending on the value of γ, it will encode the idea that a reward that is received in the present is more desirable/valuable than the same reward received at any point in the future .

Making MDP a 5-tuple:$(\Sigma, A, \Phi(\cdot,\cdot), \rho, \gamma)$

The goal of a MDP is to optimise a policy $\pi(\alpha|\sigma)$, which defines the probability that the agent should take an action α given an observed state σ, maximising a cumulative reward function of random rewards, i.e the expected discounted sum of rewards over an infinite time horizon. So, an agent during every time step in its existence, tries to maximise the expected return:

$$U(t) = \sum_{t=0}^{\infty} \gamma^t \rho(\sigma_t) \tag{2.3}$$

The equation 2.3 establishes a measure of value for a given *sequence of states*, however, this measure needs to be used to define the value for a *specific state*. This measure ideally represents both the immediate reward, and a measure of of the agent 'heading in the right direction', which in this context means to make a profitable trade, to accumulate future rewards.

To summarise so far, the *utility* of a state is the expected reward received in the current state, plus the rewards we accumulate on the journey through future states, following a given policy. This definition is framed as an *expectation* because the environment that is specified in this domain is stochastic, so future states are not deterministic. This notion of *utility* is encapsulated in what is called the value function.

$$V_\pi(\sigma) = E_\pi[U(t)|\sigma_t = \sigma] \tag{2.4}$$

This equation can be rephrased with the derivation[**?**] to give :

$$V_\pi(\sigma) = E_\pi[\rho(\sigma_t) + \gamma(V(\sigma_{t+1}))|\sigma_t = \sigma] \tag{2.5}$$

This now defines a function for calculating the utility of a *specific state* as input, and the function returns a value for entering the given state.

Alongside this, the Action-Value function, an estimate of the value given an action is as follows:

$$Q_\pi(\sigma, \alpha) = E_\pi[U(t)|\sigma_t = \sigma, A_t, = \alpha] \tag{2.6}$$

Considering the fact that the agent is interested in the optimal policy, and the optimal policy is the sequence of actions, that maximise the utility until the 'game' is finished, the value function alone cannot be used, as it only provides the expected return from a specific state, but information regarding which action to take is not provided. If the value function is to be used alone, the agent would have to simulate all possible actions to determine which action takes the agent to the state with the highest utility, which impossible in this problem domain. The Action-Value function connects the expected returns with actions, which provides the agent with information on what action to take to maximise its utility.

The derivation in [**?**] allows the equation to be rephrased in such a way that matches the format of a Bellman equation, which then allows the action-value equation 2.6 to be rephrased as a Bellman Expectation Equation, subject to a policy π:

$$V_\pi(s) = E_\pi[R_t + \gamma V_\pi(S_{t+1}|S_t = s] \tag{2.7}$$

Which allows the Value 2.5 , in finite terms, to be expressed as:

$$V_\pi(s) = \sum_{\alpha \in A} \pi(\alpha|\sigma) \sum_{\sigma'} \sum_{r} p(\sigma', r|\sigma, a)\{r + \gamma V_\pi(\sigma')\} \tag{2.8}$$

As well as the Action-Value function 2.6, expressed as:

$$Q_\pi(s) = \sum_{\alpha \in A} \pi(\alpha|\sigma) \sum_{\sigma'} \sum_{r} p(\sigma', r|\sigma, a)\{r + \gamma Q_\pi(\sigma')\} \tag{2.9}$$

2.2.2 Optimality

In an ideal scenario, the agent wants to find the optimal solution to the MDP problem. So, the definition of 'optimal' is as follows:

Optimal Value Function. The optimal state-value function $V_*(\sigma)$ is the maximum value function over all policies: $V_*(\sigma) = \max_\pi V_\pi(\sigma)$. In essence, there are all kinds of policies that the agent can follow in the Markov chain, but this means that the agent wants to traverse the problem in such a way to maximise the expectation of rewards from the system.

Optimal Action-Value Function. The optimal state-value function $Q_*(\sigma)$ is the maximum action-value function over all policies $Q_*(\sigma) = \max_\pi Q_\pi(\sigma)$. This means that if the agent commits to a particular action, this provides the maximum possible returns out of every possible traversal thereafter. Crucially, if the agent knows $Q_*(\sigma)$, then it has found the best possible traversal, so it has 'solved' the entire problem. So informally, an MDP is solved if $Q_*(\sigma)$ is solved.

Optimal Policy. The optimal policy, is to best possible way for an agent to behave in an MDP. The previous two definitions define *what* the reward is, but this defines *how* to achieve the reward. To define what makes one policy better than another, a partial ordering is created over policy space. $\pi \geq \pi' if V_\pi(\sigma) \geq V_{\pi'}(\sigma), \forall \sigma$.

The theorem [**?**] summarises that the optimal policy achieves the optimal value function as well as the optimal state-value function.

2.2.3 Challenges to Find Optimal Solutions

The Bellman Optimality Equation is typically used to solve for trivial MDP problems however there are key challenges that are prevent it to be a solution in practice.

- Bellman Optimality Equation is non-linear - as the Expectations are intractable
- No closed form solution (in general)

To deal with this challenges, there are very clever solutions that will be covered in the next section.

2.3 Solving Reinforcement Learning

Model-Free

There are numerous methods designed, developed and tested to solve the MDP problem, such as dynamic programming. However, solutions that are similar in nature require a well defined model and environment. In most practices, the MDP agent does not know the environment and its intricacies, which must be figured out by the agent itself. *Model-Free* solutions give up the assumption that the model is well defined for the agent, which is more ideal in practice.

Model Free Prediction

This is an 'evaluation' task, where given a policy π the agent evaluates its Value function $V_\pi(\sigma)$ as mentioned in 2.8, which can be done recursively through via the bellman equation until convergence.

Model Free Control

Control is about the method used to improve on a policy π to find the optimal policy π_* as outlined in 2.2.2. This is done via a method called *Policy Iteration*, where unlike the prediction problem which iterates over the Value Function, *Policy Iteration* iterates over the Action-Value function 2.9. This is done via iterating over every $Q_\pi(\sigma, \alpha)$ value and greedily taking the action with the maximum expected return. However, this solution in particular ignores the concept of 'delayed gratification' in which an individual state may not give the agent the maximum expected return, but the states following it yield greater reward, which would be ignored with this greedy solution.

In general, solving the Bellman Equation is to recursively solve for 2.8 and 2.9 is very inefficient as mentioned in below 2.6, so other solutions are explained below.

2.3.1 Model Free Prediction: Monte Carlo

The Monte Carlo (MC) method is one of the simpler and intuitive solutions to the reinforcement learning problems

- MC methods learn directly from experience
- MC methods have no prior knowledge of MDP transitions
- Uses the simplest possible idea: Using sample means from episodic experience

Goal: Given a policy π, learn V_π from episodes of experience: $\sigma_1, \alpha_1, r_1, ..., \sigma_k$ Recall that the value function uses the expected return, however, Monte Carlo uses the empirical sample means as an approximation for the expected return:

$$V_\pi(\sigma) = E_\pi[U(t)|\sigma_t = \sigma] \approx \frac{1}{N}\sum_{i=1}^{N} U_{i,\sigma} \tag{2.10}$$

2.3.2 Model Free Control: Monte Carlo

To solve the control problem, a greedy policy function is used w.r.t to the current value function.

$$\pi_*(\sigma) = arg\max_{\alpha} Q_\pi(\sigma, \alpha) \tag{2.11}$$

The major problem with this approach in this project's problem domain is the fact that because the state space in BSG is very complex, and because the Monte Carlo solution requires a 'reset' back to a specific state σ_t to effectively approximate the expected returns, the solution is not a good fit for the problem at hand.

2.3.3 Model Free: Temporal Difference Learning - TD(0)

Temporal Difference Learning (TDL) is an *online*-policy method in which the agent learns updates it's prediction at every time step throughout an episode, this differs to the MC solution as MC relies on episodic experiences as a whole to learn, rather than learning during the actual episode.

In its mathematical notation, this means that the value function $V_\pi(\sigma)$ is updated toward the *estimated* return $\rho(\sigma_t) + \gamma V_\pi(\sigma_t)$:

$$V_\pi(\sigma) \leftarrow V_\pi(\sigma) + \beta(\rho(\sigma_t) + \gamma V_\pi(\sigma_t) - V_\pi(\sigma_{t+1})) \tag{2.12}$$

Where the estimated return is analogous to the Bellman equation 2.8, in which $\rho(\sigma_t)$ is the immediate reward and $\gamma V_\pi(\sigma_t)$ is the discounted future reward for the next step and where β is the generalised *learning rate*. We substitute this *estimated* return(otherwise known as the *TD Target* in for the *real* return unlike in MC methods, which results in the *TD error* of:

$$\delta_t = \rho(\sigma_t) + \gamma V_\pi(\sigma_t) - V_\pi(\sigma_{t+1}) \tag{2.13}$$

Intuitively this differs with MC methods because, in an example where the agent was driving a car and **almost** crashes, but then drives off safely: the MC method would not update the value function in a negative sense because the end goal was not negative, whilst with TD learning, the *online* learning policy allows the agent to update the value function for the previous time steps leading up to the incident to change future behaviour. This is desirable behaviour for the prop trader problem as it means that the agent could potentially learn the states that lead up to a a sudden drop or rise in price of a stock, which the trader could have bided or asked for, which may have led to a loss. Whilst in MC the trader would have had to commit the to a bad trade to have learnt from the experience.

To solve the control problem for TD Learning , there are various options that can applied to this kind of learning that follows the 'Generalised Policy Iteration' structure, with the exploration vs exploitation problem trade-off being achieved with the $\epsilon - greedy$ method. They are: *SARSA, SARSA, Q-Learning, Expected SARSA*. The following equations have been obtained from [**?**]

SARSA

This method uses every element of the quintuple of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, inspiring the name "SARSA".

$$Q(\sigma_t, \alpha_t) \leftarrow Q(\sigma_t, \alpha_t) + \alpha[\rho(\sigma_t) + \gamma Q(\sigma_{t+1}, \alpha_{t+1}) - Q(\sigma_t, \alpha_t)] \tag{2.14}$$

This is an *on-policy* method which means that the agent will learn by using the experience sampled by the same policy.

SARSAmax/Q-Learning

This is a *off-policy* TD algorithm. This means that the learned action-value function, Q, directly approximates Q_*, the optimal action-value function. This occurs independent of the policy that is being followed.

$$Q(\sigma_t, \alpha_t) \leftarrow Q(\sigma_t, \alpha_t) + \alpha[\rho(\sigma_t) + \gamma \max_{\alpha} Q(\sigma_{t+1}, A) - Q(\sigma_t, \alpha_t)] \tag{2.15}$$

Expected SARSA

This is an *on-policy* TD control algorithm. It is similar to Q-learning, with a slight variance. Instead of the maximum over next state–action pairs, the algorithm uses the expected value, taking into account how likely each action is under the policy being used. Given the next state, σ_{t+1}, this algorithm moves deterministically in the same direction as SARSA moves in expectation.

$$Q(\sigma_t, \alpha_t) \leftarrow Q(\sigma_t, \alpha_t) + \alpha[\rho(\sigma_t) + \gamma E_\pi[Q(\sigma_{t+1}, \alpha_{t+1})|\sigma_{t+1}] - Q(\sigma_t, \alpha_t)] \tag{2.16}$$

All three control solutions for TD learning follow the same *Policy iteration* procedures as outlined earlier in this section.

2.3.4 Value Function Approximation

In an ideal scenario, the discussed methods so far (MC and TD) would work for all reinforcement learning problems, however there is a critical hurdle that these solutions to not account for in real world problems. The solutions discussed thus far solve the reinforcement learning problems using *tabular methods* i.e treating $V_\pi(\sigma)$ and $Q_\pi(\sigma, \alpha)$ as dictionaries, which is simply not a feasible solution for problems with a large state space. As examples, Backgammon has 10^{20} states, and the game AlphaGo Zero solved, 'Go', has 10^{70} states. Backgammon is considered a small game, which provides some added perspective as to the scale of problems that reinforcement learning can solve.

This problem is tackled using *Value function approximation*:

$$\hat{V}(\sigma, \mathbf{w}) \approx V_\pi(\sigma) \tag{2.17}$$

$$\hat{Q}(\sigma, \alpha, \mathbf{w}) \approx Q_\pi(\sigma, \mathbf{w}) \tag{2.18}$$

These functions are parametric function 'approximators', where $\mathbf{w}$ are the weights. So now, the parametric regression allows the approximator to generalise the previously denoted dictionaries as a general fitted function, which allows the agent to query the function for unseen/infinite state spaces.

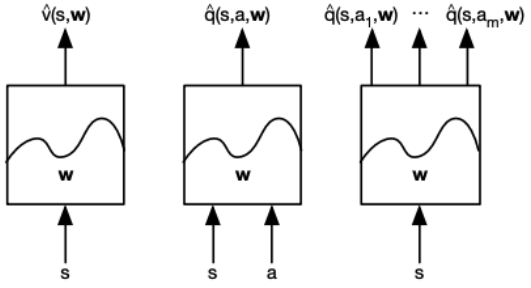

Figure 2.3: Types of value approximators, reproduced from Silver [**?**]

The figure on the left hand side shows the architectures of the varying types of approximators. The leftmost architecture represents a *Value function* approximator, where a 'black-box' function spits out the 'value' of being in state s. The middle architecture showcases the *action-value* approximator, where again, a 'black box' function takes in the state the agent is in, and the action it is considering, and it spits out how good that action is. Lastly, the rightmost architecture represents an architecture that will give the agent the action-value for all possible actions in the given state s.

The natural step to take at this point is to choose the type of 'black-box', and this project explores the usage of neural networks.

The methodology to optimising these weights is Stochastic Gradient Descent(SGD). To perform SGD we need a differentiable function with input parameter $\mathbf{w}$, J($\mathbf{w}$) , where the gradient of J($\mathbf{w}$) is defined as:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_N} \end{pmatrix} \tag{2.19}$$

In a supervised learning example, the goal is to find parameter vector $\mathbf{w}$ that minises the mean-squared error (MSE) between the approximated value function $\hat{V}(s, \mathbf{w})$ and true value $V_\pi(\sigma)$ and then use SGD to find local minima as defined in [**?**]:

$$J(\mathbf{w}) = E_\pi[(V_\pi(\sigma) - \hat{V(\sigma, \mathbf{w})})^2 \tag{2.20}$$

However, considering the context is reinforcement learning, there is no objective truth for $V_\pi(\sigma)$, there are only rewards to indicate to the agent whether or not the actions are good or bad, which are acquired through experience. Luckily, the previous subsections covered precise this. The objective function can be defined with the help of the TD/MC method defined earlier as:

- For MC the 'objective truth' becomes the return U_t: $\Delta\mathbf{w} = \beta(U_t - \hat{V}(\sigma_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{V}(\sigma_t, \mathbf{w}$
- For TD the 'objective truth' becomes the TD target U_t: $\Delta\mathbf{w} = \beta(\rho(\sigma_{t+1}) + \gamma\hat{V}(\sigma_{t+1}, \mathbf{w}) - \hat{V}(\sigma_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{V}(\sigma_t, \mathbf{w})$

2.3.5 Policy Gradient

The last subsection explained the use of value approximation functions 2.17 and 2.18, to pick actions greedily with $\epsilon - greedy$ methods to acquire the max Q_π during policy evaluation - summed up as policy iteration.

A more natural and direct approach may be to directly parameterise the *policy* rather than the value

functions - This is known as Policy-Based Reinforcement Learning. So now the policy will be manipulated directly, by controlling and learning the parameters to affect the distribution by which actions are picked: so the policy can be directly modelled as:

$$\pi_\theta(\sigma, \alpha) = P[\alpha|\sigma, \theta] \tag{2.21}$$

The goal for policy based reinforcement learning is that for a given policy $\pi_\theta(\sigma, \alpha)$ with paramters θ, find the best set of paramteres θ. However, the notion of 'best' is ambiguous, and which is why an objective function needs to be defined. There are numerous kinds of functions depending on the type of environment, however, considering that a market session is an episodic environment, which has the notion of a *start-state*, the most suitable objective function is as follows:

$$J_1(\theta) = V^{\pi_\theta}(\sigma_1) = E_{\pi_\theta}[v_1] \tag{2.22}$$

This objective function encodes the notion that if the agent always starts in some start state σ_1 or if the agent has a distribution over σ_1, how does the agent maximise the total end reward.

To find the θ that maximises the objective function $J(\theta)$ to find the optimal policy π_*, the project considers gradient *ascent*.

Policy gradient algorithms search for a local maximum for the objective function $J(\theta)$ by ascending the gradient of the policy, w.r.t to the parameters θ

$$\Delta\theta = \beta\nabla_\theta J(\theta) \tag{2.23}$$

where $\nabla_\theta J(\theta)$ is the **Policy Gradient** and β is the learning rate

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_N} \end{pmatrix} \tag{2.24}$$

Monte Carlo Policy Gradient

To compute the policy gradient analytically by exploiting the likelihood ratio trick:

$$\begin{aligned} \nabla_\theta \pi_\theta(\sigma, \alpha) &= \pi_\theta(\sigma, \alpha) \frac{\nabla_\theta \pi_\theta(\sigma, \alpha}{\pi_\theta(\sigma, \alpha)} \\ &= \pi_\theta(\sigma, \alpha) \nabla_\theta \log_{\pi_\theta}(\sigma, \alpha) \end{aligned} \tag{2.25}$$

Which allows the equation to represent a familiar term in statistics and machine learning, the score function [**?**] which allows the gradient ascent to maximise the log likelihood to tell it how to adjust the gradients.

For **discrete** actions, a *Softmax* Policy is used, where the actions are weighted using a linear combination of features $\phi(\sigma, \alpha)^T\theta$, and then to convert this linear combination to a probability function, these are exponentiated:

$$\pi_\theta(\sigma, \alpha) \propto e^{\phi(\sigma,\alpha)^T\theta} \tag{2.26}$$

To make the score function:

$$\nabla_\theta \log_{\pi_\theta}(\sigma, \alpha) = \phi(\sigma, \alpha) - E_{\pi_\theta}[\phi(\sigma, \cdot)] \tag{2.27}$$

Intuitively this means that the feature for the action that the agent actually took, minus the average feature for all the actions the agent might have taken. So, the score function summarises how much more of the given feature it has more than usual, and if it gets a more reward, then adjust the policy to do it more.

Actor Critic Policy Gradient

An alternative method is also explored in this project. In the MC Policy Gradient method, the notion of the action-value functions defined earlier in this chapter are completely ignored, and focuses on the policy itself. In addition, the MC policy gradient method also has a high amount of variance. Contrastingly, the Actor-Critic method reintroduced the notion of the action-value function via value function approximation outline in subsection 2.3.4, whilst also aiming to reduce the variance.

The main component in this method is that a *Critic* is used to explicitly estimate the action-value function, rather than using the returns of the episode experience.

$$Q_w(\sigma, \alpha) \approx Q^{\pi_\theta}(\sigma, \alpha) \tag{2.28}$$

The Actor-Critic algorithms maintain two sets of parameters:

- Critic - Updates the action-value function paramters: w
- Actor - Updates policy parameters θ, in the direction suggested by the critic

The actor is the entity that actually makes the decision in the environment. The critic merely observes the actor, however, it evaluates the actor's decisions. The critic element is introduced in order to reduce the high variance experienced from using MC Policy Gradient methods.This methodology is unique in the sense that it combines the previously explained Value-based methods, as well as the newly defined Policy-based methods. Unsurprisingly, this method follows the overarching structure of policy gradient methods, however, they are defined as an *approximate* policy gradient:

$$\begin{aligned} \nabla_\theta J(\theta) &\approx E_{\pi_\theta}[\nabla_\theta log_{\pi_\theta}(\sigma, \alpha) Q_w(\sigma, \alpha)] \\ \Delta\theta &= \beta \nabla_\theta log_{\pi_\theta}(\sigma, \alpha) Q_w(\sigma, \alpha) \end{aligned} \tag{2.29}$$

The main idea here is that the actor adjusts the policy parameters in the direction that, according to the critic, will get more reward. So, the true action-value function is replaced by the critic's approximation of this function using a neural network.

The obvious question is how the critic is estimating the action value function, luckily this was covered in the previous sections, because this is a policy evaluation problem. subsections 2.3.1 and 2.3.3

2.3.6 Discrete vs Continuous Action Spaces

2.4 Summary

This chapter aimed to summarise the actual technical challenges of the prop trader problem as well as the plethora of techniques that are traditionally used to solve reinforcement problems. The chapter started with providing the necessary context to understand how reinforcement learning is structured as a problem, and then the consequent sections built up the necessary knowledge to understand the main solutions outlined in the latter sections of the chapter. The project will aim to implement these techniques.

Chapter 3

Project Execution

Chapter 4

Critical Evaluation

Chapter 5

Conclusion

Appendix A

An Example Appendix

Content which is not central to, but may enhance the dissertation can be included in one or more appendices; examples include, but are not limited to

- lengthy mathematical proofs, numerical or graphical results which are summarised in the main body,
- sample or example calculations, and
- results of user studies or questionnaires.

Note that in line with most research conferences, the marking panel is not obliged to read such appendices.