Limit order placement optimization with Deep Reinforcement Learning

Learning from patterns in raw historical cryptocurrency market data

by

Marc B. Juchli

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday July 19, 2018 at 09:00 AM.

Student number:	4634845	
Project duration:	November 1, 2017 – July 19, 2018	
Thesis committee:	Prof. dr. M. Loog,	TU Delft, supervisor
	Dr. J. Pouwelse,	TU Delft, co-supervisor
	Prof. dr. ir. M.J.T. Reinders,	TU Delft

This thesis is confidential and cannot be made public until July 31, 2018.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Financial institutions buy or sell assets based on various reasons and such high-level trading strategies oftentimes define the purpose of their business. Regardless of their trading strategy, the invariable outcome is the decision to buy or sell assets.

This work aims to make a step towards answering the non-trivial question on how to optimize a buy or sell of an asset on a stock exchange with the use of reinforcement learning techniques. Particularly:

> How can one design a reinforcement learning environment and construct features, which are derived from a limit order book, in order to optimize on the non-trivial problem of limit order placement, and what are the limitations thereof?

We study event data from a crypto-currency exchange of our choice and build a framework that allows to simulate and understand the outcome of order placement in this market. We then try to overcome the exploitation of other participants in the market by building an intelligent trader that follows a placement strategy which aims to execute orders to a favourable price. Multiple reinforcement learning approaches are developed that consider various types of features, derived from the financial data set. To investigate the performance and analyze the behaviour of these approaches we further develop reinforcement learning environments on top of the aforementioned framework which simulate the processes: *order placement* and *market making* as an extension of the former.

Our findings show...

Preface

Preface…

Marc B. Juchli
Delft, January 2013

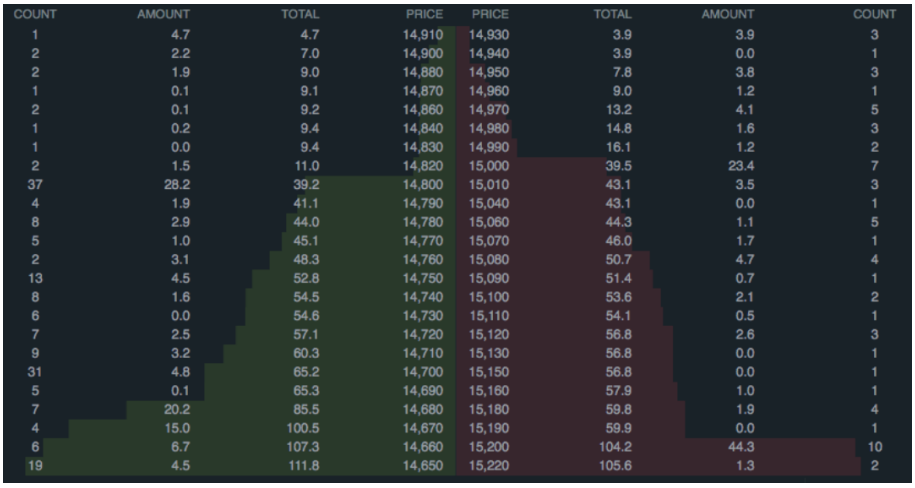
Contents

1 Introduction

Financial institutions make decisions on whether to buy or sell assets based on various reasons, including: customer requests, fundamental analysis[3], technical analysis[13], top-down investing[12], bottom-up investing[1] and many more. The high-level trading strategies oftentimes define how the institution positions itself in the financial markets and, if existent, towards its customers. Regardless of the high-level trading strategy that is being applied, the invariable outcome is the decision to buy or sell assets. This work aims to make a step towards answering the non-trivial question on how one can optimize a buy or sell of an asset on a stock exchange with the use of reinforcement learning techniques. The following sections will elaborate this problem briefly and state the research objectives of this work. We then list the contributions made to the research communities throughout this work, followed by a brief overview of the structure of this report.

1.1. Context and Problem Statement

We are concerned about the way assets, specifically *securities* (exchange traded assets), are traded at stock exchanges. There is little consensus as to when corporate stock was first traded; some argue that the exchange, in the form as we know it today, dates as far back as 1531, when the East Indian Company stock was traded in Antwerp[11]. Modern financial markets such as the London Stock exchange (LSE), the New York Stock Exchange (NYSE) but also the numerous crypto-currency exchanges which appeared suddenly in the last few year, all rely on the same very same principles as back then. They allow participant (so-called traders) to buy or sell a given amount of a security to, respectively for, a certain price. When in the late '90s the regulatories started to let traders reach into the market using electronic communications networks (ECNs), a new era arose [30]. Since then, high frequency trading (HFT) and sophisticated algorithmic trading makes up a substantial and ever increasing part of the participants of the electronic markets. Their servers are oftentimes co-located with the exchanges and specialized computer networks have been constructed to provide millisecond advantage for arbitraging trades between the exchanges. Ever since, traders without such equipment and techniques feel that they are at a disadvantage in such an environment. [30] While anything else than trading trough electronic channels would be unthinkable of today, a certain gap between trading companies and investors without fibre access to the exchanges or supporting algorithms still exists. As a result, traders are forced to take an initial loss into account when buying or selling securities – and might not even be aware of it. In order to understand why that might be the case, we have to grasp a basic understanding of a so-called order book and how securities are bought at an exchange.



COUNT	AMOUNT	TOTAL	PRICE	PRICE	TOTAL	AMOUNT	COUNT
1	4.7	4.7	14,910	14,930	3.9	3.9	3
2	2.2	7.0	14,900	14,940	3.9	0.0	1
2	1.9	9.0	14,880	14,950	7.8	3.8	3
1	0.1	9.1	14,870	14,960	9.0	1.2	1
2	0.1	9.2	14,860	14,970	13.2	4.1	5
1	0.2	9.4	14,840	14,980	14.8	1.6	3
1	0.0	9.4	14,830	14,990	16.1	1.2	2
2	1.5	11.0	14,820	15,000	39.5	23.4	7
37	28.2	39.2	14,800	15,010	43.1	3.5	3
4	1.9	41.1	14,790	15,040	43.1	0.0	1
8	2.9	44.0	14,780	15,060	44.3	1.1	5
5	1.0	45.1	14,770	15,070	46.0	1.7	1
2	3.1	48.3	14,760	15,080	50.7	4.7	4
13	4.5	52.8	14,750	15,090	51.4	0.7	1
8	1.6	54.5	14,740	15,100	53.6	2.1	2
6	0.0	54.6	14,730	15,110	54.1	0.5	1
7	2.5	57.1	14,720	15,120	56.8	2.6	3
9	3.2	60.3	14,710	15,130	56.8	0.0	1
31	4.8	65.2	14,700	15,150	56.8	0.0	1
5	0.1	65.3	14,690	15,160	57.9	1.0	1
7	20.2	85.5	14,680	15,180	59.8	1.9	4
4	15.0	100.5	14,670	15,190	59.9	0.0	1
6	6.7	107.3	14,660	15,200	104.2	44.3	10
19	4.5	111.8	14,650	15,220	105.6	1.3	2

Figure 1.1: Order book snapshot: https://www.bitfinex.com/t/BTC:USD

Figure 1.1 shows a snapshot taken at some time t from the trading pair Bitcoin (BTC) versus US dollar (USD) taken at the Bitfinex[1] cryptocurrency exchange. The order book shows two sides, the parties who are willing to buy on the left and the parties who are willing to sell on the right. The columns indicate the number of buyers and sellers (*count*) who are willing to buy, respectively sell, a certain *amount* for a given *price*. The column *total* is the cumulative sum of the amount, or volume, on each side. The two sides are separated by the *spread*. In this particular case, the current best *bid* price at which someone is willing to buy, is $14,910.00 and the best ask-price at which someone is willing to sell, is $14,930.00. Therefore, the spread is currently $20.00 wide.

Suppose we want to buy 1.0 BTC. Two possible ways to do so are:

1. Buy i shares (1.0) right away for $14,930.00 from a seller. We submit a *market* order.

2. State a price for which we are willing to to buy i shares (1.0) at price p, for example at $14,910.00, and wait until someone is willing to sell for this price. We submit a *limit* order.

Both types of orders come with their advantages and disadvantages. A market order ensures that we will be able to acquire the stated amount of shares immediately for $14'930.00, provided that no one else is ahead of us or the seller cancels his/her listing. In this case we are automatically willing to pay for the next available best price. However, we do pay a premium compared to the limit order since ask prices are listed higher than bid prices and the more shares one wants to buy, the more sellers we have to contact and buy their offerings to an increased price. With a limit order the exchange guarantees that we will pay $14'910.00 or less. That is, when a seller is willing to sell for the stated price or less, the exchange would match the offerings of both parties. However, this comes with the risk that we will never be able to buy if nobody is going to sell at the mentioned price, which will force us to buy the demanded shares at a later point in time. As the price of a share evolves over time, we might get lucky and be able to buy for a cheaper price than at the time of the initial attempt. The other scenario is that the price did not develop in our favour such that we have to buy for a higher price later on, thus we pay a so-called *opportunity cost*. A third order type, the *cancel* order allows a trader to cancel his/her previously posted limit or market order at any given point in time.

With the brief understanding of how traders can interact with the exchange, we specify the problem of *Order placement* as follows. Order placement determines the price at which a trades places its order. Optimizing order plaement tries to minimize the opportunity cost ideally aims to achieve a more favourable price to pay (respectively receive) than what is currently offered at the market price. Literature (According to Guo et. al. [19]) therefore specifies a time scale of ten to hundred seconds within which a trader has to complete his task to either buy or sell the shares. A time scale less than ten seconds refers to high frequncy trading and above 100 seconds is konwn as the process of *order execution*. Thus, we define order placement opimization: at which price p should one attempt to buy or sell i shares within a time horizon H of 100 seconds? As we shall see, optimizing the placement is not as trivial as one would think, even though the concept of the order

[1] https://www.bitfinex.com

book and the three order types a trader can choose from is admittedly simple. There are various properties that evolve from a limit order book and the participants in the market over time, which can interfere with the intention of buying and selling drastically. Furthermore, since the foundation of electronic trading networks and algorithmic trading, the amount and sophistication of other market participants is ever increasing whereas everyone aims for their advantage over others.

The fact that reinforcement learning learns by maximizing rewards, makes this technique unarguably suitable to work within this context. As a result, a reinforcement learner should be able to foresee on how to place orders according to the given market condition and therefore protect an investor form paying the aforementioned premium to other participants in the market.

1.2. Research objectives

This work extends on the findings of Kearns et. al. [29] who have studied the behaviour of order placement and order execution and further developed a reinforcement learning strategy in order to proceed optimization. Their work has pre-processed and applied features, which were derived from order book data, to a reinforcement learning algorithm which is similar to Q-Learning. Our works aims to do similar research, whereas the crypto-currency domain is chosen, instead of traditional stocks. In addition, rather than constructing features by hand we make an attempt to benefit from hidden patterns in raw market data by using deep reinforcement learning techniques. We therefore investigate the behaviour of order placement in a historical USD/BTC market and try to find properties which may be beneficial for optimization purposes. Given this knowledge we should build a reinforcement learner that can act as an intelligent trader and places limit orders with the incentive to buy or sell asset to a favourable price. We make use of an end-to-end learning process with which the agent improves based on the outcome of the placed orders and ultimately learns a strategy that allows to buy and sell shares at favourable prices. However, in order to simulate and understand the outcome of order placement and more importantly allow interaction with a reinforcement learning agent, we are required to build a framework. The framework should provide capabilities to collect and process market data in order to reproduce a historical order book that serves as a data source. We further demand the framework to provide the functionality of a match engine which emulates the functionality of a stock exchange that can match orders and determine the resulted price paid (respectively received) according to the historical order book. Given the capabilities of the framework, a reinforcement learning environment should be built which allows agents to act as an intelligent traders. The agents will place limit orders with the incentive to learn how to buy or sell asset to a favourable price by choosing limit order prices accordingly. However, the agents demand features which contribute to the learning of how to place its orders accordingly. Therefore, we have to reason about what kind of features can be derived from an order book and if there are any patterns that can be detected, such that the extracted information could contribute to the learning process. In addition, while the previously mentioned work from Kearns et al. had success in using pre-processed market data as features, we believe that raw market data in combination with deep reinforcement learning can be equally successfull. Hence why our ambition is to determine if deep reinforcement learning is perhaps an even more suitable choice in order to deal with unexpected market situations. Finally, the main objective of this research can be formulated in one sentence:

> How should one design a reinforcement learning environment and construct features, which are derived from a limit order book, in order to optimize on the non-trivial problem of limit order placement?

Clearly, included in the study are the limitations of the setting that we considered here.

1.3. Contributions

This thesis makes use of concepts from various research communities in order to work on the above mentioned objectives. The particular contributions made throughout the project are:

- Information retrieval techniques are applied in order to collect and process financial data sets. We analyze the data and find patterns which might be beneficial for the process of limit order placement optimization. Feature sets are then constructed which incorporate the found patterns and allow deep reinforcement learning agents to learn an order placement strategy.
- We study how to translate the problem of limit order placement into a reinforcement learning context. With the use of software engineering techniques we then build an environment which simulates

a rudimentary stock exchange and allows an agent to optimize on the given problem. Therefore we make use of the OpenAI Gym[2] library and contribute our work to the community to proceed further investigations.

- Investigations of reinforcement learning agents applied to the process of *order placement* (and the closely related process known as *market making*) will unveil the capabilities of deep reinforcement learning on a continuously changing state space derived from a multivariate time series. Under these circumstances we determine the possible benefits and limitations of deep reinforcement learning, when applied to noisy environments.

1.4. Document structure

In Chapter 2 we first provide background information to the reader concerning the components of a stock exchange and the fundamentals of the closely related time series. We further make the reader familiar with (Deep) Reinforcement Learning. In Chapter 3 we elaborate on the behaviour of order execution followed by approaches of both statistical and machine learning nature. Chapter 4 explains the process of data collection and its preparation which was done prior its use in the following chapters. Namely, Chapter 5 explains the experimental setup of the Reinforcement Learning environments, the agents and the features being processed and used. In Chapter 6 we then analyze the data and proceed execution placement with various techniques, including the reasoning of our findings. Finally, Chapter 7 formulates a conclusion of our findings and states a future research direction.

[2] https://github.com/openai/gym

2

Preliminaries

In this chapter we provide background information in order to understand the previous work done in this field that is introduced in Chapter 3. We will further rely on the knowledge provided in this chapter when describing the data collection an processing in Chapter 4 as well as when constructing the experimental setup in Chapter 5. We rely on the reader to be patient while reading this chapter as the interplay between the introduced components may not be immediately obvious be will become clear later in the report when the components come to use. At first, the concept of the previously introduced order book is described in greater detail, as this serves as the data structure for the collected historical data. Subsequently, a simplified match engine is described with which we will emulate a local broker that can match orders given the historical order book. An introduction to the properties of a time series is then introduced as the order book underlies its principles. Furthermore, reinforcement learning is introduced, whereas we declare the differences to other machine learning techniques, followed by a detailed explanation of all its components. Finally, deep reinforcement learning is introduced as an extension to the previously described reinforcement learning principles.

2.1. Order Book

Traders post orders in a limit order book in order to state their intention to buy (respectively sell) a given asset, as described in Section 1.1). Orders listed in the limit order book implicitly provide so called *liquidity* to the market as other traders can consume these offerings by posting an order with the equivalent price to sell (respectively buy) the asset.

This section introduces the most popular order types with which a trader can post their offerings into a limit order book. We will learn with which type a trader can ensure to provide liquidity to the market and benefit from lower fees and with which type the trader can state the wish to immediately buy or sell assets and implicitly take liquidity from the market. Furthermore, the characteristics an historical order book that is filled with orders from traders is explained as this will come handy when the match engine is explained in the following section.

2.1.1. Orders

As indicated by the name, an order is an order to buy or sell a stock. There are various types of orders which determine how the order that is placed should be executed by the exchange. In this section we provide information about the two most common types, namely the *limit order* and the *market order*, We define the indication on whether to buy or sell as the *Order Side*,

$$OrderSide = \{Buy, Sell\} \tag{2.1}$$

Before we define the order types in greater detail, we conclude what is said above and define the *Order* as,

$$Order = \{Order_{Limit}, Order_{Market}\} \tag{2.2}$$

Limit order

A limit order implies the attempt to buy or sell a stock at a specific price or better,

$$Order_{Limit} = (side, quantity, price_{Limit}) \tag{2.3}$$

, where $side \in OrderSide$, $quantity \in \mathbb{R}^+$ and $price_{Limit} \in \mathbb{R}^+$.

A buy limit order can only be executed at the limit price or lower, and a sell limit order can only be executed at the limit price or higher [9]. More precisely, in case of a buy order and if the best price on the opposing side of the book becomes lower or equals (respectively higher or equals, in case of a sell order), the broker will match those two orders, resulting in a *trade*. The disadvantage of this order type is that there is no guarantee whether the order gets executed. In case no order on the opposing side appears, the order remains (possibly forever) unexecuted.

Market order

A market order implies the attempt to buy or sell a stock at the current market price, expressing the desire to buy (respectively sell) for the best available price. Therefore,

$$Order_{Market} = (side, quantity) \tag{2.4}$$

, where $side \in OrderSide$ and $quantity \in \mathbb{R}^+$.

The advantage of a market order is that as long as there are willing buyers and sellers, the execution of the order is almost always guaranteed. [10] The disadvantage is the price you pay when your order is executed. Market orders are executed by starting from the best price of the opposing side, then traversing down the book as liquidity is consumed. Hence, market orders tend to become expensive, especially for large orders.

2.1.2. Characteristics

Figure 1.1 shows a real world example of a limit order book; in this case the snapshot was taken from a known crypto-currency exchange. Being precise, this is the *state* of an order book at some time t and shows the current offerings (in form of limit orders, see 2.1.1) from participants at this moment in time (we neglect the possibility that the state might have been changed during the data sending process). Hence, we refer to it as an *order book state (OS)*. We refer to the *order book (OB)* that is used in this project as a recorded history of order book states.

$$OB = OS_1, ...OS_n \tag{2.5}$$

As we can see, every such state holds entries on the buyer and seller side which change in their *price* and *amount*. To each such row, which can be formed by participants who submitted limit orders of some amount at the same price level, we refer to as *order book entry (OE_{s_l})* of the side s at level l.

$$OE_{s_l} = (count, price, amount) \tag{2.6}$$

, whereas $count \in \mathbb{N}$, $price \in \mathbb{R}^+$ and $amount \in \mathbb{R}^+$. As a result, the order book state is a sequence containing order book entries for each *side* (buy and sell) and a time stamp ts of the state,

$$OS = (ts, OE_{b_1}, ..., OE_{b_n}, OE_{s_1}, ..., OE_{s_n}) \tag{2.7}$$

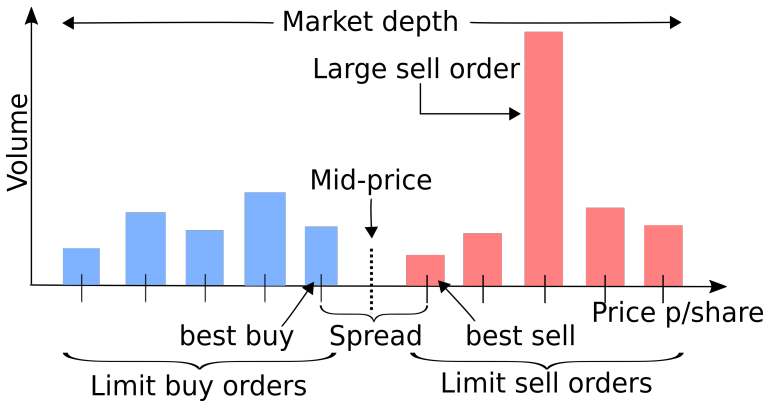


Figure 2.1: Figure taken from [6]. Simplified limit order book and provides understanding of some characteristics.

Figure 2.1 illustrates a simplified order book, from which we can derived definitions. The *limit level* specifies the position of an order book entry within the side of an order book state and the so-called *market depth* corresponds to how deep in the order book buyers and sellers have listed offerings. A deep order book therefore indicates a large range of limit levels. The term *volume* can relate to the total volume traded over a given time horizon, or can indicate the sum of amounts of what is currently offered to a certain price. Considering the sides of the order book, a *bid* refers to a price on the buyer side and the *best bid* represents the highest price for which someone is willing to buy a given asset. The best bid appears as the first order book entry on the buyer side, closest to the spread. Contrarily, an *ask* refers to a price on the seller side and the *best ask* represents the lowest price for which someone is willing to sell a given asset. The best ask appears as the first order book entry on the seller side, closest to the spread. Consequently, the *market price* is the average between the best bid and best ask price and the *spread* indicates the difference between the best bid and best ask.

The most recent price on which a buyer and seller agreed upon to trade a security is known as *quote*. In an *order driven market*, liquidity is a synonym for the ease of trading. *Liquidity* stands for the amount provided by parties of the opposing side and is what effectively enables one to buy and sell securities. That is achieved by submitting limit orders which are not immediately executed. A so called *market maker* provides liquidity to the market by posting limit orders which are not immediately executed. In return, the market maker pays less fees than a market taker, the so-called *maker fee*. Contrarily, the *market taker* takes liquidity out of the market by posting either market orders or limit orders which immediately execute. As this is not beneficial to the exchange, the market taker pays a slightly increased rate of fees, known as *taker fee*.

2.2. Match Engine

The *matching engine* is the component which responsible for the process of matching buy and sell orders at a stock exchange, such as *NASDAQ* or *NYSE* as examples of traditional stock exchanges; or *Bitfinex*, *Bittrex*, *Coinbase* as examples of crypto-currency exchanges. In order to determine the outcome of an order, the trader would typically submit the order to an exchange and either trade on the live market or get access to a test environment, which can be, if existent, still costly. Consequently, the order is processed at the current market and there is no option to process it on a historical data set in order to determine outcome, had the order been posted at some time t in the past. For the aforementioned reasons, a *local* match engine is being developed that allows to evaluate the outcome of order placement on a historical order book data set, free of charge. The local match engine is a key element of the order placement optimization process that ultimately a reinforcement learner will proceed. The outcome of matched orders will directly affect the reward received by an agent with which it will try to improve its capabilities.

This section first give the definition of a *trade* as a result of two matching orders. Subsequently, the time horizon as an addition to the previously introduced order types (Section 2.1.1) is presented with which we can describe the interface of the match engine that will be used throughout the learning process. Finally the rules of the implementation of the local match engine is provided which explain the mechanics of the matching process.

2.2.1. Trade

In order to understand the purpose of the matching process, which is described in more detail below, we first have to define what a *trade* is. A trade results when the orders (Eq. 2.2) from two parties with opposing order side (Eq. 2.1) agree on an amount and price to trade their shares. That is,

$$Trade = (ts, side, type, quantity, price) \tag{2.8}$$

, where ts is the time-stamp when the participants agreed on the exchange of the products, $side \in OrderSide$, $type \in OrderType$, $quantity \in \mathbb{R}^+$ and $price \in \mathbb{R}^+$.

2.2.2. Interface

This match engine enables the simulation and evaluation of order placement without the need to consult an electronic trading network. Alongside the order that is sent to the match engine (directly or via an electronic trading network), the user can specify a *time horizon H* indicating how long the order should stay active. The two most commonly used timing mechanisms are:

Good Till Time (GTT): The order stays in the order book until a specified amount of time is consumed. *(Some implementations declare the same idea as Good Till Date whereas a date and time is provided which specifies until when the order is valid.)*

Good-Til-Canceled (GTC): The order stays in the order book until the user submits a cancellation.

Hence, the match engine exposes an interface that represents a function $match$ which takes any type of an *Order* (Section 2.1.1) and the time horizon H and returns a sequence of *trade* (Eq. 2.8). That is,

$$match : Order \times H \rightarrow Trades \tag{2.9}$$

, whereas $|Trades| \in \mathbb{N}$. The order is *filled* if the sum of the traded quantity is equal to the amount stated in the submitted order, *partially-filled* if the traded quantity is > 0 but not filled and *not filled* otherwise.

The matching process behaves different, depending on the submitted order type, and is explained in the following paragraph.

2.2.3. Rules

The rules presented below are, compared to match engines used in electronic trading networks, are rather primitive, yet correct within the subset of its capability. The rules used by the order matching engine are mainly derived and simplified from [5]:

1. Limit orders (defined in Section 2.1.1) may be partially filled or not filled at all in case of absence of parties on the opposing side.
2. Market orders (defined in Section 2.1.1) will execute immediately when an opposite order exists in the market.
3. Market orders may be partially filled, at different prices, depending on liquidity in the opposing side of the book.
4. Limit orders are attempted to be matched from the given point in time forward, or in case of a Good Till Time (GTT) for as long as specified.

2.2.4. Limitations

Since the match engine used in this project is a rudimentary implementation for the purpose of simulating and analyzing order execution and placement, it features only a subset a conventional match engine used by electronic trading networks. That said, the following limitations have to be taken into consideration:

Participants: first and foremost, the match engine is used locally where no other participants are interacting during its use. In order to be able to approximate the most likely outcome, historical data serves as a simulation of participants acting in the market. While this is valuable real world data, unfortunately

it does not cover 1) the possibility of hidden participants entering or 2) leaving the market upon placing an order from our side. Participants who would enter the market would likely be in our favour as they would act as potential buyers and sellers and therefore provide liquidity. Participants who leave the market would introduce a slight disadvantage as there would be less liquidity. Since both parties are absent, we consider our implementation as a good approximation without a major advantage or disadvantage.

Ordering this match engine is restricted to simulate the matching of only one order from one participant at a time. Hence, any type of ordered processing of incoming orders (typically solved with a queuing system) is not supported. However, this functionality is also not required for our purposes.

Timing inaccuracy: occurs when submitting an order with a time horizon (see Section 2.2.2). The fact that we rely on historical data and the time stamps of the orders submitted from participants in the past is a limitation when submitting an order over a certain period of time (GTT). It can occur that at the end of the period the order would have some time t left (e.g. a few seconds) but the next following order book state is nearer to the future than t would allow. We therefore have to abort the matching process early.

2.3. Order execution and placement

Given the understanding of the order book and the match engine, it is obvious that a trader has a variety of options on how to approach a market and fulfill his duties to buy (respectively sell) shares. Conceptually, the process a trader proceeds involves the following two steps: *order execution* and *order placement*, whereas the latter is the main subject of this thesis.

Many useful definitions which highlight the difficulties related to the order execution domain were stated by Lim et al. [22] and Guo et al. [19]. Most importantly, *order execution* concerns about optimally slicing big orders into smaller ones in order to minimize the price impact, that is, moving the price up by executing large buy orders (respectively down for sell orders) at once. By splitting up a big order into smaller pieces and spreading the execution over an enlarged time horizon, the impact cost can be lessened. Typically on a daily or weekly basis. Contrarily, *order placement* concerns about optimally placing orders within ten to hundred seconds. Placing hereby refers to the setting of the limit level for a limit order as described in Section 2.1.1. The aim is to minimize the *opportunity cost* which arises when the price moves against our favour.

Literature[19, 29] suggests to use the *volume weighted average price (VWAP)* as a measure of the *return* of order placement and order execution. That is,

$$p_{vwap} = \frac{\sum v_p * p}{V} \tag{2.10}$$

, whereas p is the price paid for v_p shares and V represents the total volume of shares.

2.4. Time series

According to the efficient market hypothesis[24], the price of an asset reflects all the information available to the market participants at any give time. Given the vast information flow, the natural consequence is that the price of such an asset changes over time. As we have seen in the previous Section **??**, the price of an asset is determined by actions proceeded by traders. Therefore, the financial markets and particularly the order book are best described over time, namely as a *time series*. More precisely, the definition of a time series is an ordered sequence of values of a variable at equally spaced time intervals [4]. The nature of time series data originated the applications generally known as Time Series Analysis and Time Series Forecasting. Both of which play an important role throughout this project, and therefore a brief background is provided in this section.

2.4.1. Time series analysis

The analysis of data observed at different points in time leads to problems in statistical modelling and inference. More specifically, the correlation of adjacent points in time can restrict the applicability of conventional statistical methods which traditionally depend on the assumption that these adjacent observations are independent and identically distributed. A systematic approach by which one attempts to answer the mathematical and statistical questions posed by these time correlations is commonly referred to as time series analysis. Therefore, mathematical models are developed with the primary objective to provide plausible descriptions

for sample data. [32]

Some of the time series behaviours, which will be presented within this body of work, may hint that a sort of regularity exist over time. We refer the notion of regularity using a concept called *stationarity*, as introduced in [32].

A **strictly stationary** time series is one for which the probabilistic behavior of every collection of values $x_{t_1}, x_{t_2}, ..., x_{t_k}$ is identical to that of the time shifted set $x_{t_1+h}, x_{t_2+h}, ..., x_{t_k+h}$ for all time shifts $h = 0, \pm 1, \pm 2, ...$

A **weakly stationary** time series, x_t, is a finite variance process such that

- the mean value function μ_t is constant and does not depend on time t, and
- the autocovariance function $\gamma(s,t)$, depends on s and t only through their difference $|s-t|$.

Whereas μ_t is defined as

$$\mu_t = \mathbb{E}(x_t) = \int_{-\infty}^{\infty} x f_t(x) dx \tag{2.11}$$

with f_t being the *marginal density function* [32]. And $\gamma(s,t)$ is defined as

$$\gamma(s,t) = cov(x_s, x_t) = \mathbb{E}[(x_s - \mu_s)(x_t - \mu_t)] \tag{2.12}$$

for all time points s and t.

Henceforth, we will use the term stationary to mean weakly stationary; if a process is stationary in the strict sense, we will use the term strictly stationary.

2.4.2. Time series forecasting

In statistics, prediction is a part of statistical inference. Providing a means of the transfer of knowledge about a sample of a population to the whole population, and to other related populations is one description of statistics. However, this is not necessarily equivalent to the process of predicting over time. This process, instead, is known as forecasting and describes the transfer of information across, often to very specific point in, time [14]. Hence the problem is defined in [21] as: *forecasting future values X_{t+h} where $h > 0$ of a weakly stationary process X_t from the known values X_s where $s \leq t$*. The integer h is called lead time or forecasting horizon, whereas h stands for horizon.

Forecasting methods can be classified, according to [16], into three types: *Judgemental forecasts* produce projections based on intuition, inside knowledge, and any other relevant information. *Univariate methods* forecast depends on present or past values of the time series on which the forecast is projected. Finally, *multivariate methods* forecast depends on one or more additional time series variables or multivariate models.

Over the course of this work, we make use of univariate- and multivariate methods.

2.5. Reinforcement Learning

This section first aims to describe what Reinforcement Learning is and highlights its differences to other machine learning paradigms. We briefly reason why this particular technique might be an appropriate choice for the task of optimizing order placement. Then, a basic understanding about Markov Decision Processes is provided, after which we explain the interaction between the Reinforcement Learning components, followed by a description of their properties.

2.5.1. Advantages of end-to-end learning

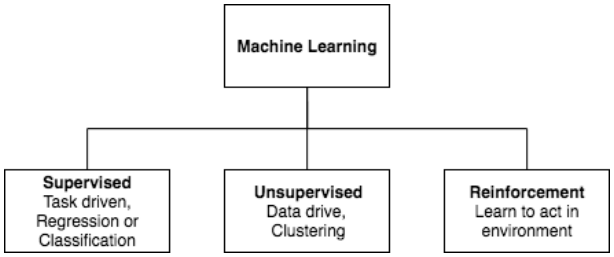


Figure 2.2: Categorization of machine learning techniques

Reinforcement Learning is a specific learning approach in the Machine Learning (see Figure 2.2) field and aims to solve problems which involve *sequential decision making*. Therefore, when a decision made in a system affects the future decisions and eventually its outcome, the aim is to learn the optimal sequence of decisions with reinforcement learning.

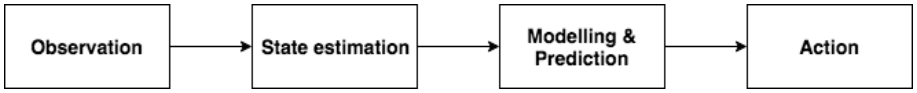


Figure 2.3: Reinforcement learning end-to-end learning pipeline

For optimizing order placement in limit order books, statistical approaches have long been the preferred choice. While statistics emphasizes inference of a process, machine learning on the other hand emphasizes the prediction of the future with respect to some variable. Machine learning paradigms, such as supervised learning, rely on an algorithm that learns by presenting a specific situation provided with the right action to do. From there, the algorithm tries to generalize the model.

Contrarily, in reinforcement learning there is no supervision and instead an agent learns by maximizing rewards. The feedback retrieved while proceeding a task with a sequence of actions might be delayed over several time steps and hence the agent might spend some time exploring until it finally reaches the goal and can updates its strategy accordingly. This process can be regarded as *end-to-end learning* and is illustrated in Figure 2.3. In abstract terms, the agent makes an *observation* of its environment and estimates a *state* for which it *models and predicts* the *action* to be taken. Once the action was executed, the agent receives a *reward* and will take this into consideration during during the prediction phases in the future. The beauty of which is that an arbitrarily complex process can be regarded as a black box as long as it can take an input from the learner to do its job and responds how well the task was executed. In our context this implies that we would model the order placement process pipeline whereas the learner improves upon the outcome of the submitted orders. In addition, for reinforcement learning problems, the data is not independent and identically distributed (I.I.D). The agent might in fact, while exploring, miss out on some important parts to learn the optimal behaviour. Hence, time is crucial as the agent must explore as many parts of the environment to be able to take the appropriate actions. [8]

Example: Since we are working with financial systems, let us assume we want to buy and sell stocks on a stock exchange. In reinforcement learning terms, the trader is represented as an *agent* and the exchange is the *environment*. The details of the environment do not have to be known as it is rather regarded as a black-box. The agents purpose is to observe the state of the environment: say for example the current price of a stock. The agent then makes estimates about the situation of the observed state and decides which action to take next – buy or sell. The action is then send to the environment which determines whether this was a good or bad choice, for example whether we made a profit or a loss.

2.5.2. Markov Decision Process (MDP)

A process such as the one sketched above, can be formalized as a Markov Decision Process. An MDP is a 5-tuple (S, A, P, R, γ) where:

1. S is the finite set of possible states $s_t \in S$ at some time step.

2. $A(s_t)$ is the set of actions available in the state at time step t, that is $a_t \in A(s_t)$, whereas $A = \bigcup_{s_t \in S} A(s_t)$
3. $p(s_{t+1}|s_t, a_t)$ is the state transition model that describes how the environment state changes, depending on the action a and the current state s_t.
4. $p(r_{t+1}|s_t, a_t)$ is the reward model that describes the immediate reward value that the agent receives from the environment after performing an action in the current state s_t.
5. $\gamma \in [0, 1]$ is the discount factor which determines the importance of the future rewards.

2.5.3. Interaction

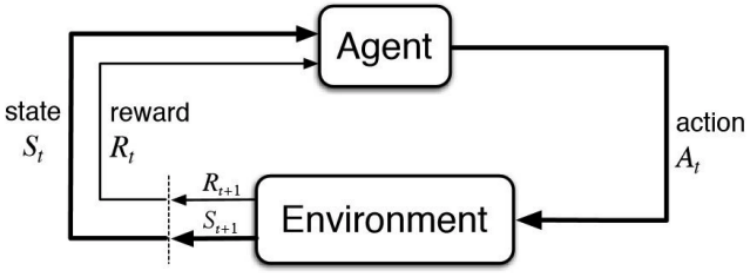


Figure 2.4: Figure taken from [8]: interaction between a reinforcement learning agent and environment. Illustrating the action taken by the client being in some state which results in some reward and a new state.

A reinforcement learning problem is commonly defined with the help of two main components: *Environment* and *Agent*.

With the interfaces provided above (Section 2.5.2), we can define an interaction process between an agent and environment by assuming discrete time steps: $t = 0, 1, 2, ...$

1. The agent observes a state $s_t \in S$
2. and produces an action at time step t: $a_t \in A(s_t)$
3. which leads to a reward $r_{t+1} \in R$ and the next state s_{t+1}

During this process, and as the agent aims to maximize its future reward, the agent consults a *policy*, which dictates which action to take given a particular state.

Policy

A policy is a function that can be either deterministic or stochastic. The distribution $\pi(a|s)$ is used for a stochastic policy and a mapping function $\pi(s) : S \rightarrow A$ is used for a deterministic policy, whereas S is the set of possible states and A is the set of possible actions.

The stochastic *policy* at time step t: π_t is a mapping from state to action probabilities as a result of the agents experience, and therefore, $\pi_t(a|s)$ is the probability that $a_t = a$ when $s_t = s$.

Reward

The goal is that the agent learns how to select actions such that it maximizes its future reward when submitting them to the environment. We rely on the standard assumption that future rewards are discounted by a factor of γ per time-step in the sense that the total discounted reward accounts to $r_1 + \gamma * r_2 + \gamma^2 * r_3 + \gamma^3 * r_4 + ...$) Hence we define the future discounted *return* at time t as

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} * r_i \tag{2.13}$$

, where T is the length of the episode (which can be infinity if there is no maximum length for the episode). The discounting factor has two obligations: it prevents the total reward from going to infinity (since $0 \leq \gamma \leq 1$), and it allows to control the preference of the agent between immediate rewards and potentially received reward in the future. [7]

Value Functions

When the transition function of an MPD is not available, model-free reinforcement learning allows the agent to simply rely on some trial-and-error experience for action selection in order to learn an optimal policy. Therefore, the value of a state s indicates how good or bad a state is for the agent to be in, measured by the expected total reward for an agent starting from this state. Hence we introduce the **value function**, which depends on the policy the agent chooses its actions from:

$$V^{\pi}(s) = \mathbb{E}[R_t] = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \ \forall s \in S \tag{2.14}$$

Among all value functions there is an **optimal value function** which has higher values for all states

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \ \forall s \in S \tag{2.15}$$

Furthermore, the **optimal policy** π^{*} can be derived as

$$\pi^{*} = \arg\max_{\pi} V^{\pi}(s) \ \forall s \in S \tag{2.16}$$

In addition to the value of a state with respect to the expected total reward to be achieved, we might also be interested in a value which determines the value of being an a certain state s and taking a certain action a. To get there we first introduce the **Q function**, which takes a state-action pair and returns a real value:

$$Q : S \times A \rightarrow \mathbb{R} \tag{2.17}$$

Finally, the **optimal action-value function** (or **optimal Q function**) $Q^{*}(s,a)$ as the maximum expected return achievable after seeing some state s and then taking some action a. That is,

$$Q^{*}(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{2.18}$$

with the policy π mapping the states to either actions or distributions over actions.

The relationship between the *optimal value function* and the *optimal action-value function* is, as already hinted by their names, easily obtained as

$$V^{*}(s) = \max_{a} Q^{*}(s,a) \ \forall s \in S \tag{2.19}$$

and thus the *optimal policy* for state s can be derived by choosing the action a that gives maximum value

$$\pi^{*}(s) = \arg\max_{a} Q^{*}(s,a) \ \forall s \in S \tag{2.20}$$

2.5.4. Environment

There are two types of environments:

Deterministic environment: implies that both the sate transition model and reward model are deterministic functions. In this setup, if the agent in a given state s_t repeats a given action a, the result will always be the same next state s_{t+1} and reward r_t.

Stochastic environment: implies that there is an uncertainty about the outcome of taking an action a in state s_t as the next state s_{t+1} and received reward r_t might not be the same for each time.

Deterministic environments are, in general, easier to solve as the agent learns to improve the policy without uncertainties in the MDP.

2.5.5. Agent

The goal of the agent is to solve the MDP by finding the optimal policy, which means finding the sequence of action that lead to maximize the total received reward. However, there are various approaches to so, which are commonly categorized (see [7]) as follows.

A *value based agent* starts off with a random value function and then finds a new (improved) value function in an iterative process, until reaching the optimal value function (Eq. 2.15). As shown in Eq. 2.14 one can easily derive the optimal policy from the optimal value function. A *policy based agent* starts off with a random policy, then finds the value function of that policy and derives a new (improved) policy based on the previous value function, until finding the optimal policy (Eq. 2.20). Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). As stated in Eq. 2.16, given a policy, one can derive the value function. The *actor-critic agent* is a combination of a value-based and policy-based agent. Both, the policy and the reward from each state will be stored. *Model-based agents* attempt to approximate the environment using a model. It then suggests the best possible behaviour.

2.5.6. Deep Reinforcement Learning

From [2] goes: *"Reinforcement learning can be naturally integrated with artificial neural networks to obtain high-quality generalization"*. The term *generalization* refers to the action-value function (Eq. 2.18) and the fact that this value is estimated for each state separately–which becomes totally impractical in for large state spaces that can occur in real world scenarios. Deep reinforcement learning generally stands for approximating the value function, the policy, or the model of reinforcement learning via a neural network. As is preferred in reinforcement learning, neural network approximates a function as a non-linear function. Therefore, the estimate of the approximation is a local optimum, which is not always desirable. In our particular case, we use deep reinforcement learning in order to approximate the action-value function (Eq. 2.18). Therefore we represent the action-value function with weights ω as,

$$Q(s,a;\omega) \approx Q^*(s,a) \tag{2.21}$$

Given a state s, the neural network outputs n linear output units (corresponding to n actions), as shown in Figure 2.5. The agent will then choose the action with the maximum q-value.

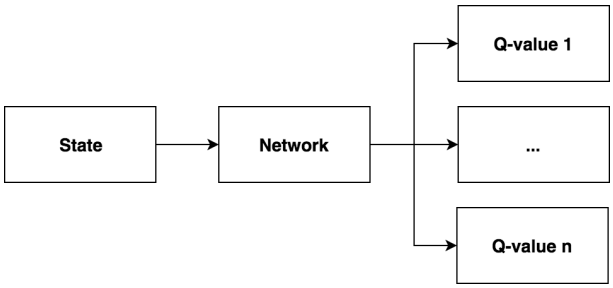


Figure 2.5: Neural network outputs q-values

In terms of the previously described reinforcement end-to-end learning pipeline, the use of a function approximator simplifies this process. We can omit the state estimation step and instead rely on raw features [26], as illustrated in Figure 2.6:

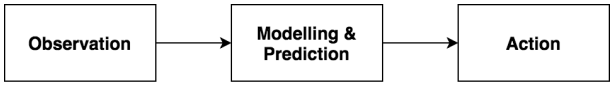


Figure 2.6: Deep Reinforcement learning end-to-end learning pipeline

3

Related Work

The literature for the order placement problem is, relative to the execution problem, sparse (confirmed by Guo et al. [19]). In this Chapter we provide an overview of the related work, upon which this project is built on or insight was taken from. We first give insight into an empirical study about the general behavior of order placements, which serves as a conceptual basis for this project. Subsequently, a statistical approach is presented to provide contrast to the following overview of previous machine learning approaches. The latter serve as a guideline on how to model the reinforcement learning process of this thesis.

3.1. Execution/Placement behaviour

Kearns et al. [28] determine which limit order price results in the most advantageous execution price. At first, the *expected execution price* is investigated with respect to the placement of the limit order. Based on this analysis the standard deviation of the resulted prices will uncover the *risk* that comes along with limit order placement. Finally, by combining the previous two results, an *efficient pricing frontier* can be drawn which highlights the trade-off between risk and returns.

Regarding the definition stated in Section 2.3, their research is to be categorized in between order execution and placement. No splitting of orders is being done, however, a time horizon of several hours was chosen, resulting in an evaluation of order placement with an extended time horizon.

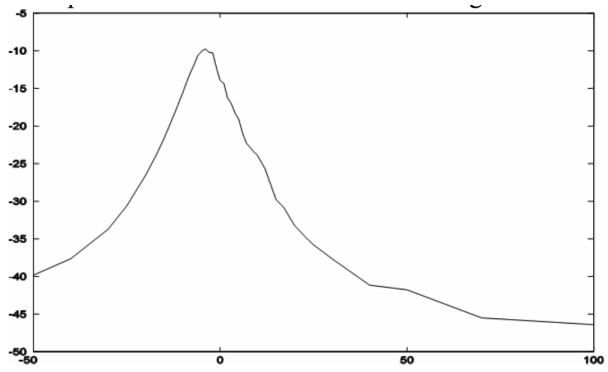


Figure 3.1: Taken from [28] and illustrates the pricing strategy that produces the most favorable expected execution price.

Figure 3.1 shows on the y-axis the return as the weighted average price paid of the expected execution price while acquiring 10,000 shares of MSFT within one hour. The x-axis represents the limit level reaching from -$50 to +$100. As it is evident from the figure, the most favorable expected execution price occurs when setting the limit price close to the price of the spread, yet on the buyer side with a price approximately $10 lower than what is currently offered. The return becomes worse when placing orders more deep in the order book (meaning offering a lower price) as the orders then to not get filled within an hours and instead the inventory has to be bought with a market order at the end of the period. Likewise, the return can be expected to be lower when placing the order higher in the order book (e.g. deeper in the opposing side of the book,

meaning one is willing to pay more). This is due to the fact that the order is being filled instantly by paying a premium.

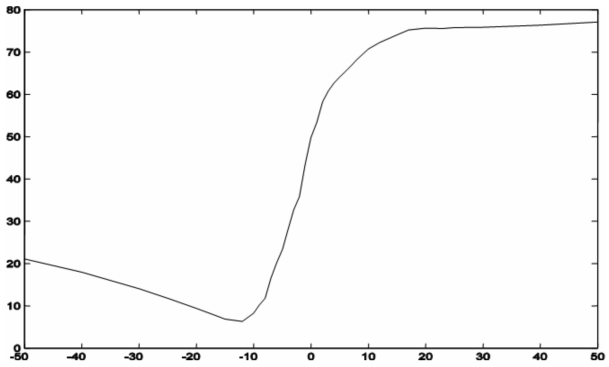


Figure 3.2: Taken from [28] and illustrates the uncertainty of the expected execution price.

Risk is being defined as the standard deviation of the returns and is illustrated on the y-axis in Figure 3.2. This is an important aspect to be considered throughout our project as it unveils to danger that comes along with placing limit orders on less favourable limit levels. Evidently, orders which are placed deep in either of the sides of the book are less likely to be executed and come with a higher uncertainty around the final price.

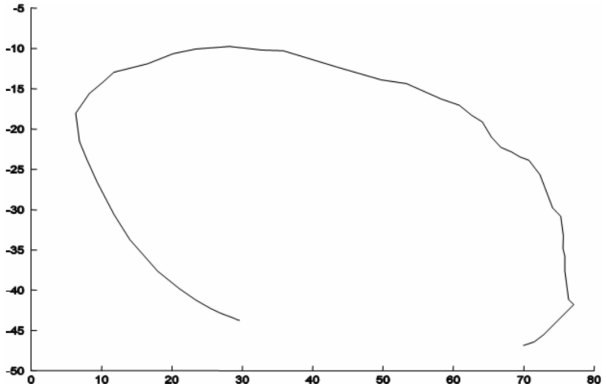


Figure 3.3: Taken from [28] and illustrates the trade-off between risk and return indicated with the efficient pricing frontier.

Lastly, both techniques were combined and result in an efficient pricing frontier (based on the *efficient frontier* initially formulated by Harry Markowitz in 1952 [25]). Therefore, Figure 3.3 shows the trade-off between the risk (x-axis) and return (y-axis). In this example, the point of minimum risk is at $(8,18)$ and the point of maximum returns at $(29,9)$. With this technique a trader, or in our case a reinforcement learning agent, can decide upon an execution strategy by choosing how much risk and return he is willing to take, and then translate the coordinates back into the corresponding limit level.

3.2. Statistical approach

Profound work in a statistical context has been done by Chaiyakorn Yingsaeree in his dissertation [35]. A framework is proposed for making order placement decisions based on the trade-off between the profit gained from favorable execution prices and the risk of non-execution. An execution probability model was developed which estimates the expected payoff (e.g. return) and its variance ($\Longrightarrow$ risk) while placing orders at a certain limit level, followed by the application of *mean variance optimization* to balance the trade-off. The framework was not able to beat the best static strategy in all evaluated cases, however, the improvement gained when it could beat the best static strategy was very significant. This gives us hope that where the statistical approach has its limitations, the reinforcement learning approach presented in this work may be able to understand the market data to a greater extent and avoid the shortcomings of the former.

We provide an overview of the framework without specific application, as this would exceed the scope of this

overview.

The strategy is to buy x shares in time T, whereas the trader is left with the following options:

1. Do nothing.
2. Submit market order at $t = 0$ for price p_0^M
3. Submit market order at $t = T$ for price p_T^M
4. Submit limit order for price p^L. If the order is not filled either a market order follows or no action is taken (depending on the use case).

A function $U_E(p)$ defines the payoff in case of an execution at a price p, and a function $U_{NE}(p)$ defines the cost if the order is not executed at the end of the period with market price p. Consequently, the payoff the trader will receive from submitting a limit buy order at price level L is defined as,

$$U(p^L) = \begin{cases} U_E(p), & \text{if order is executed.} \\ U_{NE}(p_T^M), & \text{if not executed.} \end{cases} \tag{3.1}$$

The expected price is compounded of the probability that the limit order at price p^L will be executed before the end of the period together with the distribution of the asset price at the end of the period,

$$\mathbb{E}[U(p^L)] = P_E(p^L)\ U(p^L) + [1 - P_E(p^L)] \int_{-\infty}^{\infty} U_{NE}(p) f_{p_M^T|p^L}(p) dp \tag{3.2}$$

, whereas $P_E(p^L)$ is the probability that the limit order at price p^L will be executed before the end of the period, and $f_{p_M^T|p^L}(.)$ is the probability density function of the asset price at the end of the period.
Similarly, the variance was drawn as $V[U(p^L)]$ followed by a mean variance optimization step which introduced the utility function,

$$U_O(p^L) = \mathbb{E}[U(p^L)] - \lambda V[U(p^L)] \tag{3.3}$$

, whereas λ serves as a risk factor. That is, when $\lambda = 0$ the trader is concerned about the profit only, and when $\lambda = 1$ the trader is equally concerned about profit and risk and missed opportunities. As a result, the trade-off between profit and risk is defined as,

$$\hat{p} = \arg\max_{p^L} U_O(p^L) \tag{3.4}$$

3.3. Supervised Learning approach

Fletcher et al. [17] investigates order books to find patterns which can be exploited with the aim of forecasting movement of bid and ask prices at time $t + \Delta t$. Although this is not directly applied to optimize order placement, the prediction can certainly be used as the limit price to be set while placing an order.

SVM classification techniques with different kernels along with two Multiple Kernel Learning (MKL) techniques, SimpleMKL, were used. It is a multi-class setup with three labels, A: $P_{t+\Delta t}^{Bid} > P_t^{Ask}$, B: $P_{t+\Delta t}^{Ask} < P_t^{Bid}$ and C: $P_{t+\Delta t}^{Bid} < P_t^{Ask}, P_{t+\Delta t}^{Ask} > P_t^{Bid}$. The feature used is the volume at time t at each of the price levels of the order book on both sides as a vector V_t. A set of features was constructed that contains volumes from the current time t and previous time step $t-1$.

With a time delta (Δt) of 100 seconds, an accuracy of 51% was achieved. A shorter time delta results in significantly better performance, however, this is mostly due to the fact of accurate zero movement prediction. An increased time delta results in significantly worse prediction accuracy.

3.4. Reinforcement Learning approach

A large-scale empirical application of reinforcement learning to optimize trade execution is presented by Kearns et al. [29]. Although the title of their research suggests otherwise, according to our definition in Section 2.3, their work is related to order placement with a larger time horizon H (2 minutes and 8 minutes). Hence, their research objective is defined as:

> The goal is to sell (respectively, buy) V shares of a given stock within a fixed time period (or horizon) H, in a manner that maximizes the revenue received (respectively, minimizes the capital spent).

They built a reinforcement learning based on 1.5 years of millisecond time-scale limit order data from NASDAQ. The investigation included three stocks: AMZN, NVDA, and QCOM; each with an inventory I of 5000 shares. The achieved relative improvement over a submit-and-leave strategy ranges from 27.16% to 35.50%. An additional improvement of 12.85% was achieved by considering the following market variables: Spread, Immediate Cost, Signed Volume.

The architecture developed is as follows. *States* are represented by a vector $x \in X$ and correspond to the observation state, whereas the partially observable environment is then treated to be fully observable. *Actions* ($a \in A$) represent the limit price relative to the current ask price, $ask - a$. That is, action $a = 0$ is the ask price, $a < 0$ is a limit price deep in the book and $a > 0$ is a limit order on the opposing side of the book. The *reward* represent the VWAP (Eq. 2.10) of the executed order relative to the bid-ask mid price ($(ask + bid)/2$). In case the order was not filled completely at the end of the time horizon H, then a market order follows. The chosen *algorithm* is a slightly adapted version of the Q-Learning algorithm was developed by the authors which explores the state space inductively. Starting from $t = T...0$ the algorithm explores the inventories $i = 0...I$. For each such step, all possible actions in this state are evaluated, leading to the most rewarding strategy for $t = 0$.

4

Data curation

In the crypto-currency domain, real-time price data is, for most exchanges, freely available. Historical data is oftentimes limited with respect to how far back into past the data is being provided. However, continuously recording real-time data results in a complete historical data set from the time when recording was started and provides the desired data set for this research. A historical limit order book consisting of states which store every bid and ask posted by traders is commonly referred to as a *complete order book*. The process to accumulate the data and build the order book is illustrated in a high level pipeline in Figure 4.1 below. Raw event data is collected from an exchange (Bittrex in our case) and processed in order to form a historical limit order book.

Figure 4.1: Data collection pipeline

In this chapter we explain the details of the collection process process and subsequently how a historical order book can be formed thereof, such that it can serve as the historical data source for the match engine. A sample period of the collected data set is then being investigated in order to find find and visualize properties of the market situation and behaviour of the market participants. The goal of the investigation is to find hypotheses which state why certain occurrences might be beneficial to consider for the purpose of limit order placement. Thus, the findings serve as the basis for the feature engineering process which determines the input for the learner. Therefore, as a last section of this chapter, we engineer features which cover the stated hypotheses. As a result, the features serve as the observed state that is to be evaluated by the reinforcement learning agents described in Chapter 5.

4.1. Collection

There are various ways in order to build a copy of a historical limit order book. However, the only way to rebuild a complete order book is by processing market *events*, that is, every market update for a given ticker (trading pair, in our case USD/BTC). There are three common types of events, all of which are initiated by a market participant (trader): *order created*, *order cancelled* or *order filled* in case a market order crosses the spread and initiates a trade.

Our exchange of choice for collecting data is Bittrex[1] as the exchange provides a *SignalR*[2] (a library that abstracts *HTTP* and *WebSocket*) interface from which one can extract all status updates (events) from the market. More specifically, an event in this either a buy- or sell order, or a fill (e.g. trade). Thus, we subscribe to `https://socket.bittrex.com/signalr` and filter the data field `M` for `updateExchangeState`. The data type of the message contains the name of the trading pair and a nonce to identify the unique status update in up-counting order. That is,

$$StatusUpdate = \{name, nonce, buy_1, ...buy_n, sell_1, ...sell_n, fill_1, ...fill_n\} \tag{4.1}$$

[1] https://bittrex.com/

[2] https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

, whereas $buy \in Order_{Limit}$, $sell \in Order_{Limit}$ (see Eq. 2.3) and $fill \in Trade$ (see Eq. 2.8). With that said, the orders hold an additional field $type \in \{0,1,2\}$ which specifies whether it was a *create*, *cancel* or *change* in the order.

As is evident, multiple events can be sent within one status update message. We segment the status update into separate events with the same nonce, whereas each event expresses either a limit order of side bid or ask or a filled order resulting in a trade,

$$Event = \{name, nonce, type, isTrade, trade, isBid, order\} \tag{4.2}$$

, whereas $isTrade \in \{0,1\}$ and $isBid \in \{0,1\}$ indicating whether the update contains an order or a trade. Consequently, $order \in Order_{Limit}$ and $trade \in Trade$.

4.2. Order book generation

The next step is to transform the collected events into an order book structure. By chronologically iterating over the processed events (Eq. 4.2), we create a new order book state (Eq. 2.7) for each such event that has a consecutive time stamp. During this iterative process we follow the rules below, which ensure that the correct order book entries remain in future order book states. Given the observed event, we act according to the type of the event:

Order created: order book entry is added to the current state.

Order cancelled: the amount of shares of the cancelled order is subtracted from the entry in the current state with the corresponding price level.

Order filled: the amount of the shares which were traded is subtracted from the entry in the current state with the corresponding price level.

As a result, a *list* of order book states is formed which represent a historical order book (Eq. 2.5). *We acknowledge that a list representation is by no means the most performing implementation of an order book, but for our purposes it is sufficient.*

4.3. Understanding the data set

We take a random period of the recorded data set, representing ~10 minutes worth of event data, and try to extract essential information from either raw events or the generated order book. We first obtain an insight into the market situation with regards to some of the properties mentioned in Section 2.1.2 and understand how market participants place orders. Our aim is then to find patterns of how market participants behave and how this may affect the market. Subsequently, we formulate *hypotheses* which concern why certain properties might be beneficial for the order placement process. Being aware that the following observations are taken from a random sample of a historical data set, the intention is to make a suggestion which properties might be worth to consider in the feature engineering process. This is by no means a guarantee that the same observations are true for any order book.

Figure 4.2 below shows the price movement of the sample period, indicating a movement from $10'100 to $10'030 and back within 10 minutes.

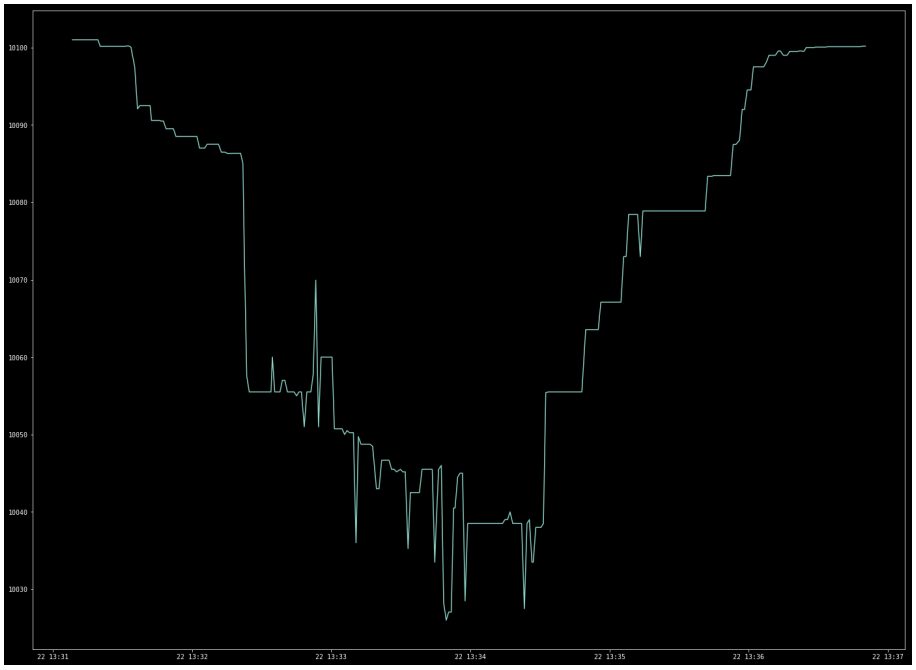
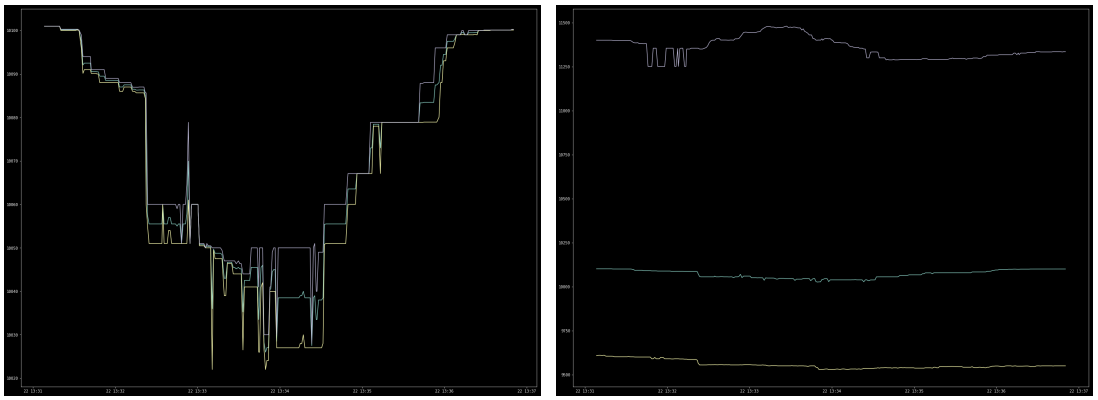


Figure 4.2: Price movement of sample data set

4.3.1. Importance of order prices



(a) Best bid and best ask

(b) Deepest bid and deepest ask

Figure 4.3: USD/BTC price and bid/ask positions

Next we show the same price movement (cyan) but with the best bid (yellow) and ask (violet) (Figure 4.3a) and the deepest level of bid and ask (Figure 4.3b). It is evident that the best bid and ask are close to the price before and after the price dip, meaning the spread is narrow. During the dip, the spread widens and is at times as large as $25.

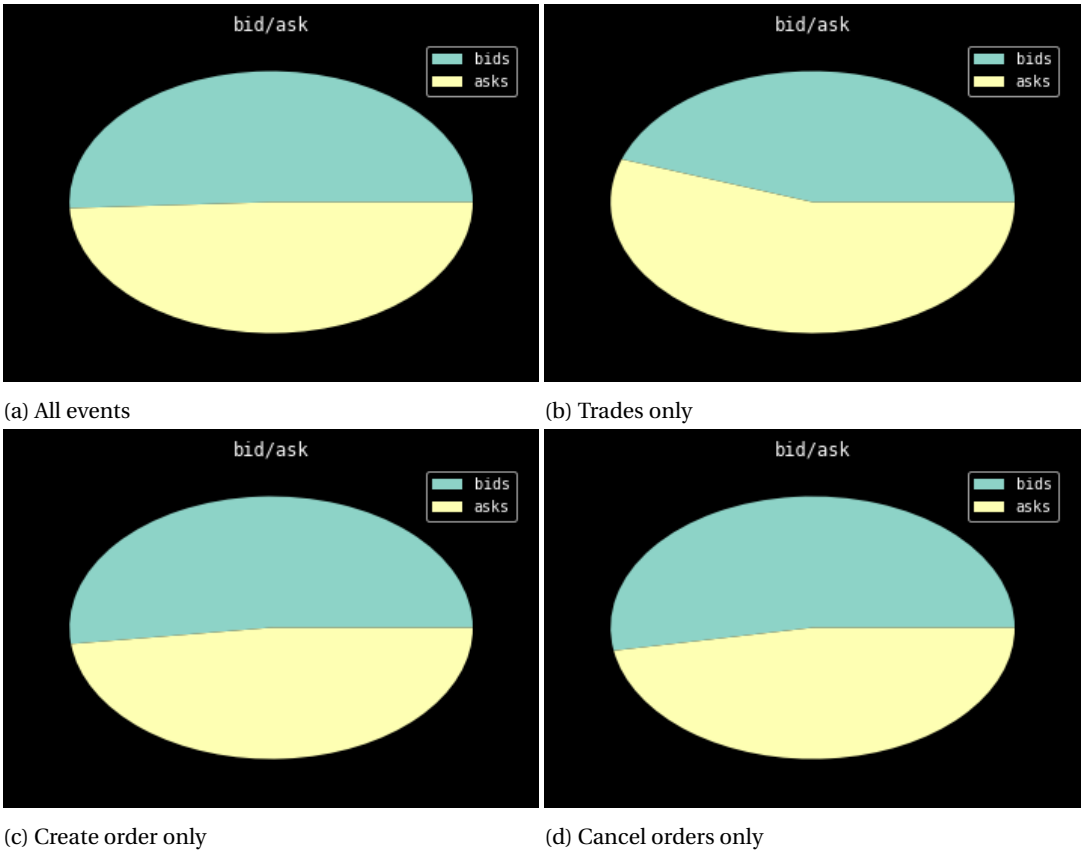
Hypothesis: *participants post offerings close to the spread when the price is stable and start offering with a certain threshold from the spread during a price fall or rise.*

The orders placed on the deepest level on the buyer and seller side undergo a very interesting change. Immediately before the the price dip, ask prices start to fluctuate as some participants cancel their listings and possibly others (trader cannot be associated as this is hidden information). On the contrary, bid prices remain much more stable. Likewise, during the fall and rise of the price, the ask price starts to increase on the deepest level of the seller side. This phenomenon is at first at a first glimpse unexpected as one would expect the sell offers to be lowered as the price falls. Knowing that the price rises shortly after, it becomes more evident that some sellers ambition was to allure buyers by threatening with higher sell listings. The ask prices return to the level before the dip as soon as the price starts rising again.

Hypothesis: *sellers allure buyers with higher listings during a price fall.*

4.3.2. Importance of order volume

It was shown that market participants position their offerings at different price levels as the asset price is moving due to trading. The second variable in posting orders is the volume and we aim to determine whether or not this is a factor which is affected during the previously introduced price movement.



(a) All events

(b) Trades only

(c) Create order only

(d) Cancel orders only

Figure 4.4: Bid / Ask volume imbalance

Figure 4.4 shows the (im-)balance of the volumes of bid and ask orders segmented as follows. Figure 4.4a shows the ratio of bids and asks of all events, including trades, creations and cancellations. Figure 4.4b show the ratio of trades being initiated by either a bid or ask order. Figure 4.4c demonstrates the ratio of created

orders and Figure 4.4d cancellation orders, again distinguishing between bid and ask. It is evident that, even though the price moved significantly within the recorded time range, the entirety of the orders is well balanced between trades or orders on the bid side and the ask side. For trades however, clearly more shares were sold than bought. It is then evident that the market participants reacted on the sale of the asset by not only creating but also cancelling more buy than sell orders. We take the last statement as an indication that the market participants may have responded to the price dip by cancelling their current buy orders and posting them deeper in the book (out of fear that the price might fall even further). Interestingly, even though the price rose after the dip, not as much volume was used on creations and cancellations of orders on the seller side.

Hypothesis: *imbalance of bids and asks of event types indicates future behaviour of participants.*

4.3.3. Volume of orders and trades over time on the market price

So far, the volume was investigated as a sum of events over time. In order to understand the behaviour of the participants in greater detail, a *volume map* shall provide better insight into the single events occurred over time.

Figure 4.5 shows the volume map of the orders which were created over the time span of the data set. Hence, the x-axis represents the time stamp and the y-axis is the volume of the placed order. For visibility reasons and since the majority of the orders have a small volume and fewer have large volume, the y-axis follows a log-scale. Participants cannot be assigned to such an order, as the trader id is non-public information. However, as we will see, one can determine some participants according to their behaviour.

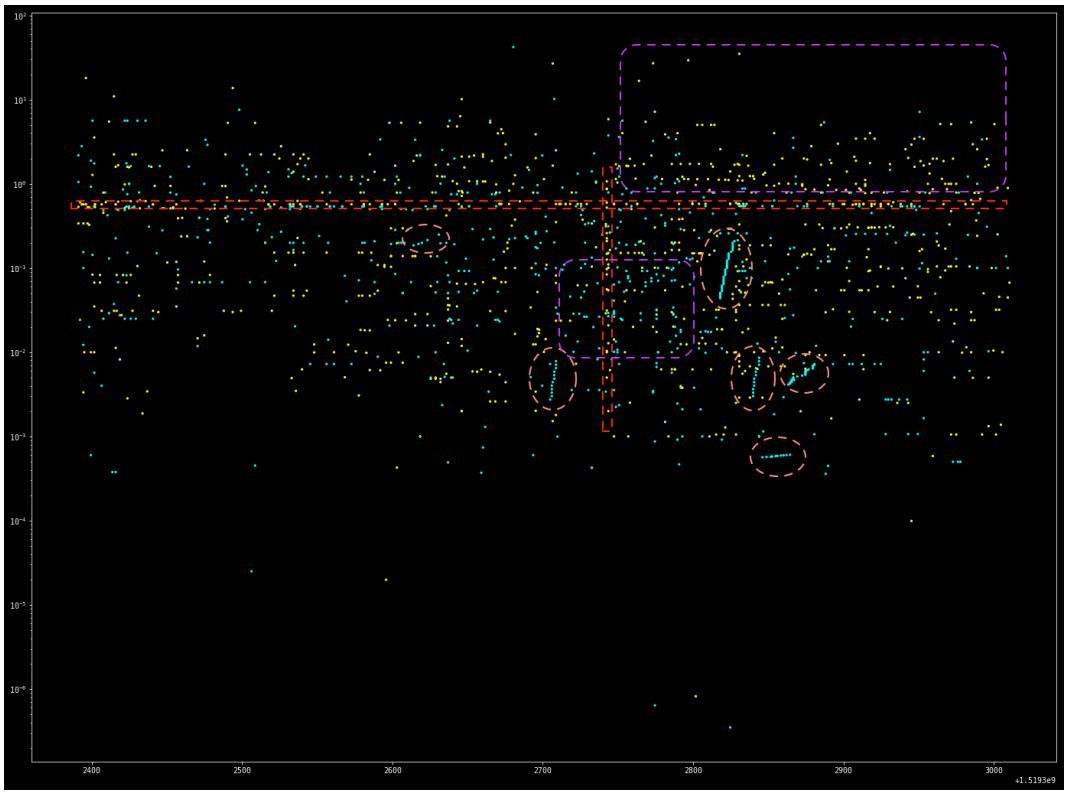


Figure 4.5: Volume map of created bid (cyan) and ask (yellow) orders.

What might not be obvious due to the log scaling is the fact that most of the orders were placed with volume between >0 and <0.5 and significantly less with larger volume. Only a few orders had a volume greater than 10 BTC, meaning that most of the participants either are willing to buy and sell only small quantities or split their orders to minimize the market impact.

From a horizontal perspective, one can detect some orders of both sides, bids and asks, being listed with the same time interval. Particularly evident is such a behaviour at the volume just below 10^0. This is likely one or multiple bots posting orders with the same volume and perhaps at a different price level. Examples of

this behaviour is marked with a red-dashed horizontal box. A similar behaviour can be seen from a vertical perspective where one or more traders post orders at different price levels, with the same price segmentation. This is market with a red-dashed vertical box. Furthermore, a very distinctive pattern is when a trader posts orders within a short period of time but changing volume. This might be evidence of someone splitting a larger order into small pieces and is marked with an orange-dashed circle. Some refer to such a behaviour as *buy-wall* (*sell-wall* in case of an ask orders) and surprisingly, this behaviour appeared the most when before and during the price started rising again (compare time stamps from Figure 4.2). Last but not least, examples of areas dominated by one particular order side (buy or sell) is marked with a purple box.

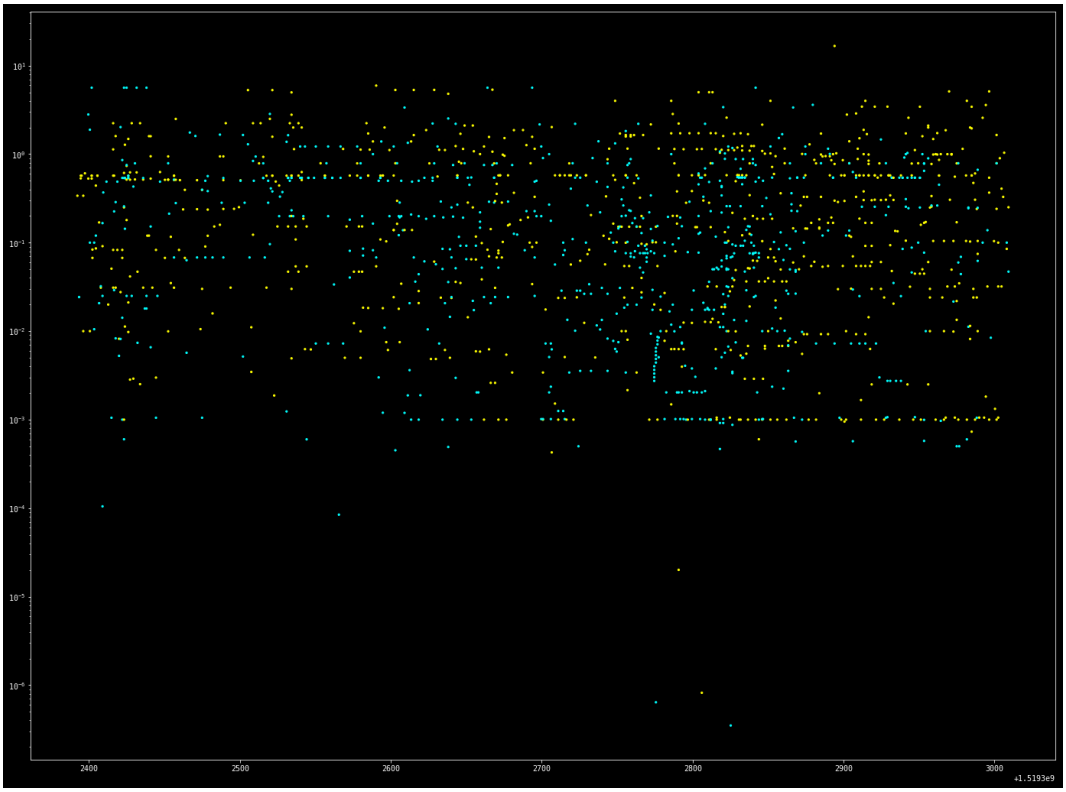


Figure 4.6: Volume map of cancelled bid (cyan) and ask (yellow) orders.

It is to be expected that at least for some of the orders, which did not result in a trade, a cancellation followed. Figure 4.6 therefore shows the cancel orders posted by traders over time.

Continuous cancellations become especially evident at volume levels just below 10^0 and 10^{-1}, correlating with the continuous create orders appeared at the same price level above. Hence, there is likely a trader following some strategy. A cancellation of one of the created buy-walls is particularly evident at the same time stamp and with equal volumes as previously discovered. Hence, making it more even more likely that the wall was created and cancelled by a single trader.

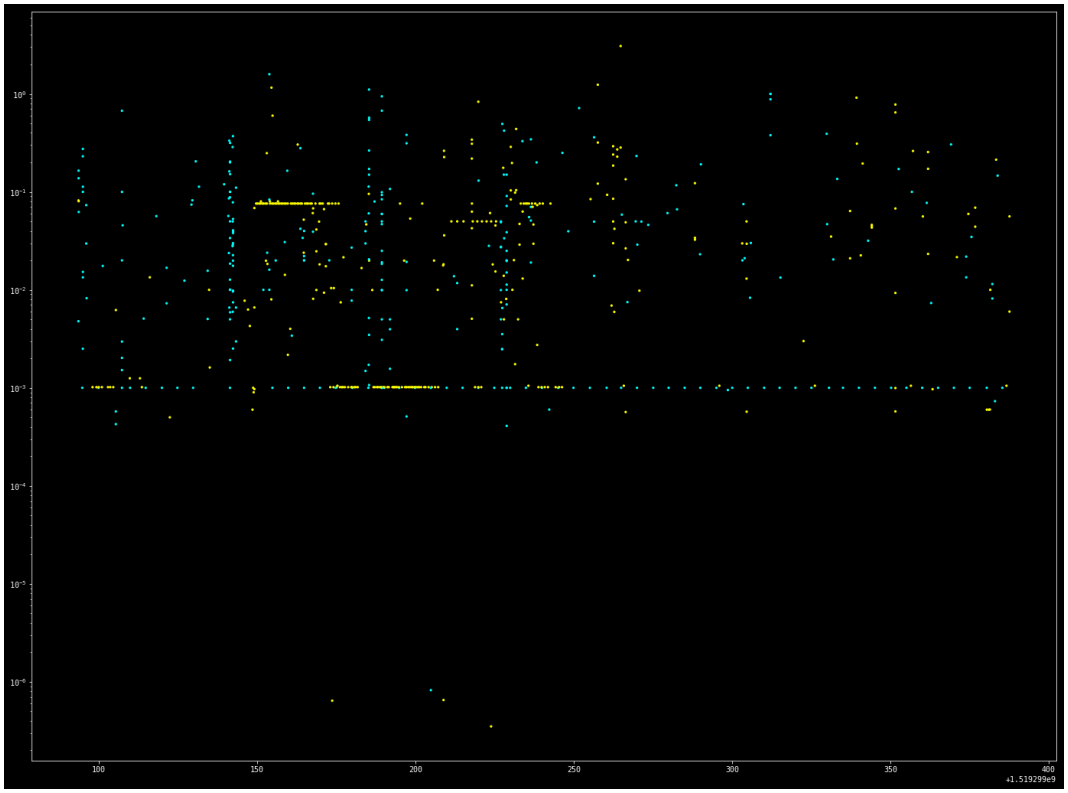


Figure 4.7: Volume map of trades initiated by a bid (cyan) or ask (yellow) order crossing the spread.

So far only posted limit orders or their cancellation where observed. Not all of those limit orders might have executed and resulted into a trade as there is always a market order required, initiated by one of the two parties involved in a trade. Figure 4.7 therefore illustrates the volume map of the actual trades occurred within this sample time range of the data set.

Immediately evident are the trades transacted with volume 10^{-3}, with oftentimes identical time interval. Additionally, an intensive series of consecutive sales occurred at the time when the price dipped. After the dip, such sales are not present anymore. Furthermore, intensive and immediate purchases are visible with various volume at some distinctive time stamps before and during the price fall. We remember that there was one spike towards a higher price during the dip, and the time stamp of which correlates strongly to the purchases visible in this figure.

Various behavioural patterns have been observed by investigating events initiated by market participants over time. For some, their impact on the market price is immediately obvious, for others it is hard to interpret by hand. However, an attempt to find correlation between the behaviour of the events and the resulted trades with the application of learning techniques seems promising.

Hypothesis: *patterns arising from posted volume in events determines future short-term trading behaviour which can be exploited in favour of order placement.*

4.3.4. Impact of traded price and volume

The price levels and volume of events over time was investigated for each type in the previous subsections. Patterns were found and a hypothesis was stated which encourages the various offerings of volume of an order to be an indicator of future behaviour of market participants, which eventually influences the market price and implicitly determines optimal order placement. The next logical step is to investigate the sum of traded volume at a given point in time (as shown in Figure 4.7) in combination with the price the asset was traded for.

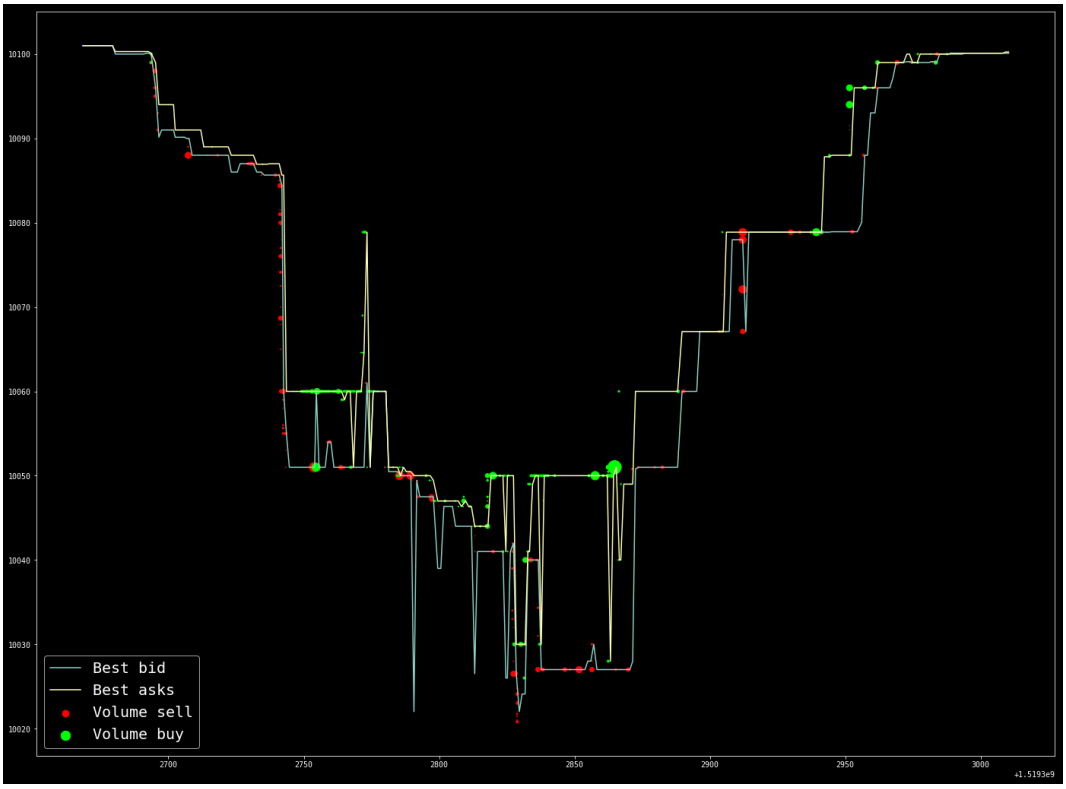


Figure 4.8: Relation of trade volume to price movement.

Figure 4.8 illustrates the volumes of trades on both bid and ask side, which resulted in a buy or sell at a certain price. The volume of these trades are illustrated as bubbles according to their size (e.g. traded volume) and price. As is known, a buy appears when one crosses the spread towards the sellers side (ask) and a sell appears when crossing towards the buyer side (bid). One can clearly see how buys are listed on the best ask price level and sells are listed on the best buy price level. Before and during the dip sells appear consecutively, followed by a series of buy orders with low volume, which caused the short spike. Interestingly, a rather large trade caused by a bid appeared shortly before the price started rising again. Even though sellers confronted the market in the middle of the price rise, participants continued buying shortly after with approximately equal volume. Concluding this observation it is evident that few trades with small volume caused a certain noise in the overall trend. Multiple consecutive trades initiated one side or a single large the like, however, led the market price to move for substantially longer period of time.

Hypothesis: *consecutive small or one large trade give an impulse that drives the market price up or down.*

4.4. Feature construction

The previous section demonstrated certain trading behaviours of the market participants in an order driven market, which ultimately determines the evolution of the limit order book. Hypotheses were laid out which give reasons to believe that the outcome in terms of a change in the order book constellation and price development can be related to aforementioned trading behaviour. Consequently this implies that orders can be placed and filled at limit levels which result in a favourable price. Therefore, the following subsections will introduce features that are derived from the previously collected (Section 4.1) and processed (Section 4.2) data and cover the assumptions stated in Section 4.3. Instead of hand-engineer features such as shown in [17, 20, 29], the aim of this project is to learn the knowledge directly from raw inputs, similar to what has been proven to be a successful method in the gaming sector[26] and was recently applied to the trading context[23].

4.4.1. Feature: price and size of historical orders

The order book was defined in Eq. 2.5 and generated in Section 4.2 serves as the first feature. More precisely for each sample at time t, we use n order book entries (Eq.2.6) of m of the order book states (Eq. 2.7) with

time stamp $ts \leq t$. Therefore, as shown in Eq. 4.3, $s_{bidask} \in \mathbb{R}^{+\,m\times 2\times 2n}$ is the state observed by a reinforcement learning agent. The order book states are ordered such that m is the closest to t. The n order book entries are closest to the spread whereas only the price bp (respectively ap) and size bs (respectively as) are considered.

$$s_{bidask} = \left[\begin{pmatrix} bp_{11} & bs_{11} \\ bp_{12} & bs_{12} \\ \vdots & \vdots \\ bp_{1n} & bs_{1n} \\ ap_{11} & as_{11} \\ ap_{12} & as_{12} \\ \vdots & \vdots \\ ap_{1n} & as_{1n} \end{pmatrix} \begin{pmatrix} bp_{21} & bs_{21} \\ bp_{22} & bs_{22} \\ \vdots & \vdots \\ bp_{2n} & bs_{2n} \\ ap_{21} & as_{21} \\ ap_{22} & as_{22} \\ \vdots & \vdots \\ ap_{2n} & as_{2n} \end{pmatrix} \cdots \begin{pmatrix} bp_{m1} & bs_{m1} \\ bp_{m2} & bs_{m2} \\ \vdots & \vdots \\ bp_{mn} & bs_{mn} \\ ap_{m1} & as_{m1} \\ ap_{m2} & as_{m2} \\ \vdots & \vdots \\ ap_{mn} & as_{mn} \end{pmatrix} \right] \tag{4.3}$$

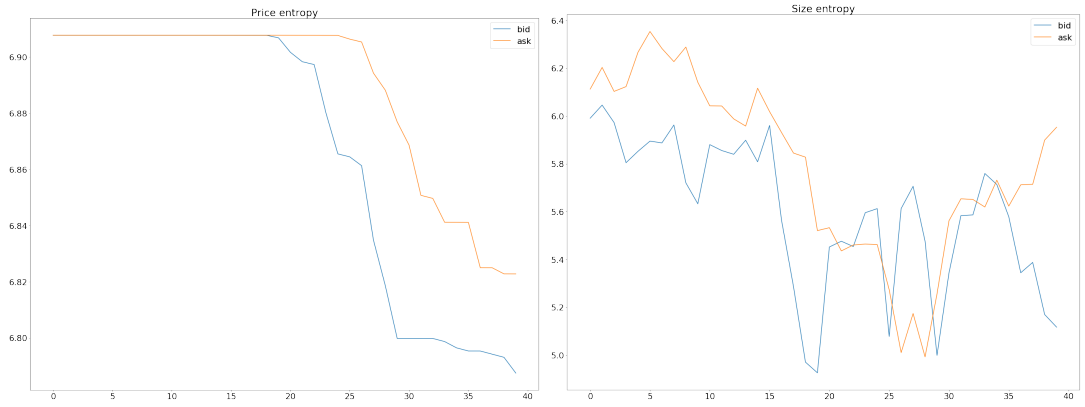
Considering that the state will be observed by a deep learning agent, which makes use of a neural network, scaling of inputs will contribute to a faster learning process. We therefore further apply normalization the the prices (bp, ap) with respect to the best ask price for each state, that is ap_{i1}. Equivalently we normalize the sizes (bs, as) with respect to the size provided at the best ask as_{i1}. While this method does not scale the values of prices and sizes within a predefined range, the values sill decrease significantly. Furthermore, empirical observations show that the minimum and maximum of prices within a single order book state do not differ more than 2%, which determines the approximate scaling boundary.

This feature incorporates some of the previously stated hypotheses and therefore enables the learner to determine whether the statements were valid or not. Particularly, the feature includes historical order prices (hypothesis 4.3.1) their volume (hypothesis 4.3.2 and partly 4.3.3).

Might be unnecessary according to our talk.

The question remains how large the window of m order books states and the number of limit levels n should be chosen. The following observation provides an intuition about the parameters, however, this by no means aims to make an estimate how well the agent may perform under the consideration a certain parameter setup.

In order to reason about the impact of n limit levels, we take the average of 100 evaluations whereas we take 1000 random order book states for which we measure the Shannon entropy[31] for a range of 40 limit levels (maximum of what goes from collection) on the bid and ask side, applied to price and size. The entropy therefore serves as an indicator of how much information can be gained for each limit level, derived from the their change in price and size for each state.



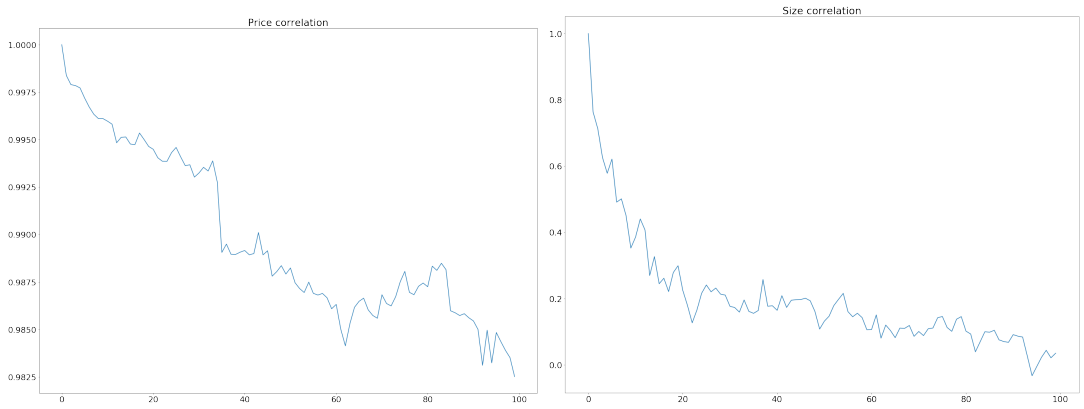
(a) Entropy of order prices

(b) Entropy of order sizes

Figure 4.9: Entropy measured for 40 limit levels

It is noticeable that the entropy remains high regarding the prices for for limit levels 0-30 on both, bid and ask side, as shown in Figure 4.9a. The price becomes slightly more constant for limit levels > 30. The entropy

for order sizes, as shown in Figure 4.9b, drop after 20 limit levels, which indicates that the accumulated order size deep in the book is more constant. We therefore suggest to consider at least 30 limit levels of the bid-ask feature.



(a) Correlation of order prices

(b) Correlation of order sizes

Figure 4.10: Correlation measured for 100 order book states

After having a brief understanding of how limit levels n affect order prices and sizes, we try to make a statement about how order book states are related to the most recent state. More precisely, we determine the correlation of the order prices and sizes from the previous m states to the most recent order book state. We take the average of 100 evaluations whereas we take a single order book state at time t and a sequence of 100 previous order book states for each of which we measure the Shannon entropy[31] to the state at t, whereas $n = 40$ (maximum). As can be seen in Figure 4.10a, the correlation of the price positions drop rapidly, however the effective change is not significant, indicating that the price changes are noticeable but overall do not differ much. Order book states which lay more than 40 states in the past are slightly less correlated to the current state. The correlation of order sizes, as shown in Figure 4.10b drops more rapid and to a much greater extent than the order prices. This indicates that traders choose a broad range of order sizes. As a result, a window size of order book states m greater than 40 states is suggested to benefit from price differences within the feature.

4.4.2. Feature: price and size of historical trades

The previously worked out a feature provides information to the learner in order to reason about the hypotheses which are derived from order placed and cancelled. In order to reason about whether or not trade events can serve as appropriate input for a learner to optimize order placement, we construct a feature that covers the hypotheses 4.3.4 and partially 4.3.3 as follows. A $Trade$ (Eq. 2.8) carries an order side os, a quantity q and a price p. Similar to the previous feature, we take n trades into consideration which occurred within a time window m, measured by the time stamps ts of the trades. Since the feature is generated at some time t when an order should be placed, the time stamp of the historical orders must satisfy $ts \leq t$.

A straightforward approach would be to construct the feature s_{trade} as,

$$s_{trade} = \begin{pmatrix} p_1 & q_1 & os_1 \\ p_2 & q_2 & os_2 \\ \vdots & \vdots & \vdots \\ p_n & q_n & os_n \end{pmatrix} \forall\ p, q, os, ts \in Trade \tag{4.4}$$

, whereas $ts_n - ts_1 \leq m$. However, trades do not occur in fixed time intervals and therefore causes the length of the vector to vary. We therefore accumulate the prices and sizes of the trades with equal order side over a fixed amount of time (in seconds) Δs. That is,

$$Trade' = (ts', os', \sum_{i=ts}^{ts+\Delta s} p_i, \sum_{i=ts}^{ts+\Delta s} q_i)\ \forall\ p, q, os \in \{Trade \mid os = os'\} \tag{4.5}$$

As a result we can redefine the feature s_{trade}, that is used by the learner as observation state, as,

$$s_{trade} = \begin{pmatrix} p_1 & q_1 & os_1 \\ p_2 & q_2 & os_2 \\ \vdots & \vdots & \vdots \\ p_n & q_n & os_n \end{pmatrix} \forall\ p,q,os,ts \in Trade' \tag{4.6}$$

, whereas $ts_n - ts_1 \leq m\ \wedge\ ts_i \geq ts_{i-1} + \Delta s$.

As a result we constructed a feature vector of length $n = \frac{m}{\Delta s}$ that contains information about the price and quantity of historical trades.

4.5. Conclusion

Event data was collected from a crypto-currency exchange and a limit order book was reconstructed thereof. The limit order book serves as the historical data set and source for the match engine in order to simulate order placement. Subsequently the price chart as the result of the generated order book was shown and an investigation of the underlying event data was proceeded. Patterns were found which give insight in how market participants positioned their offerings with respect to price and size. It was shown that the price movement was likely due to (1) an imbalance in bid and ask orders; (2) a distinctive way of posting or cancelling orders; and (3) consecutive or impulsive trades. These findings were incorporated within the feature engineering process, which resulted in two features that can be used by the reinforcement learning agents.

5

Experimental Setup

Knowledge about the components and techniques required for optimizing order placement was provided in Chapter 2, and previous approaches were introduced in Chapter 3. The process of collecting historical event data and the construction of a limit order book was explained in Chapter 4. The data was investigated and hypotheses were stated as a guideline for a future learning process. However, in order to apply reinforcement learning, an environment has to be developed which is flexible enough to allow investigation regarding different types of features and learning algorithms that are incorporated in an agent. The correctness of such an environment is critical as it emulates a stock exchange and therefore determines how orders would have been transacted in the past. If the implementation varies from the one used in exchanges, or does not cover certain edge cases, the matching of placed orders would differ significantly from the one in a production setup.

This chapter aims to build an environment that emulates a subset of the capabilities of a real world exchange in order to determine how limit orders would have been processed, had they been placed at a given point in time in the past. Therefore, the setup of the environment is described at first and explains how the required components work in combination such that a learner can simulate order placement. Additionally, an extension of this environment is provided to simulate simultaneous order placement on both sides of the book. This process is commonly referred to as *market making*. Finally, two implementations of reinforcement learning agents are provided. A Q-Learning agent will serve as the learner when no market variables are provided and a Deep Q-Network agent is developed to handle complex features.

5.1. Order Placement Environment

The reinforcement learning environment (see Section 2.5.4) that emulates order placement on historical market data is introduced in this section and enables an agent to buy or sell V shares within a time horizon H. Therefore, the previously described components (introduced in Chapter 2) come into play. The main idea of its working is that, the agent observes a state s_t (so-called observation state) at some time t and responds with an action a_t that indicates at which position to place the order in the order book, relative to the current market price. The task of the environment is then to evaluate the outcome of the placed order and report to the agent accordingly with a reward r_{t+1} and the next state s_{t+1}. Subsequently, the order is cancelled such that the agent can submit a new action for the remaining shares to be bought or sold.

OpenAI Gym [15] is an open source toolkit for reinforcement learning. The interfaces of this toolkit were used in order to follow their standards while building this environment. The advantage of which is that any OpenAI Gym compatible agent and bench-marking tools can make use of this environment.

5.1.1. Overview of components

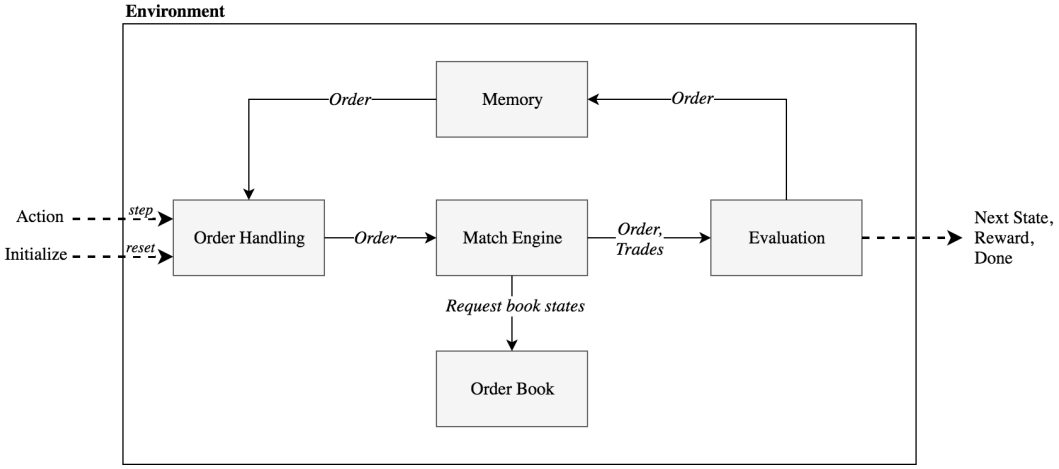


Figure 5.1: Overview of reinforcement learning order placement environment.

Figure 5.1 shows the inner workings of the order placement environment. An epoch is initialized with the `reset` function, which clears the internal state of the environment. The internal state consists of the remaining shares the agent has to buy or sell, denoted by the *inventory* i, the time step t the agent has left to do so and the previous *order*. The agent explores the environment using the $step$ function with which it sends the action to execute. The first component to react upon receiving the action is the *order handling component*, which determines whether the agent is currently trying to fill an order, that is when the agent has started an epoch, or a new order needs to be created, that is when the agent starts an epoch. In either case, a new order is created and the price is set according to the received action. Subsequently, the order is forwarded to the *match engine*, which tries to execute the order within the historical data set, namely the *order book*. The order and the possible resulted trades evolved during the matching process are then forwarded to the *evaluation component*. Since it can take multiple steps for the agent to fill an order, the remaining inventory and the consumed time of the order is updated and stored in the *memory*. Additionally, the index of the last visited order book state is stored such that in a next step the match engine will proceed the matching where it stopped last. In case no trades resulted during the matching process, only the consumed time is subtracted from the order. Otherwise, the sum of the size of the trades is subtracted from the order's inventory. Subsequently, the evaluation component calculates the reward based on the previously resulted trades. Finally, the reward, the next observation state and whether or not the order is completely filled (e.g. the epoch is done) is finally forwarded to the agent. Additionally, if the order is not completely filled after the last step taken by the agent, a market order follows in order to get to the final state.

5.1.2. Configuration parameters

For the environment to be flexible, such that agents can place orders in various settings, a total of four configuration parameter have to be defined: *order side (OS)*, *time horizon (H)*, *time step length* (Δt) and *feature type (F)*. The *order side OS* (previously defined in Eq. 2.1) specifies whether the orders, which are created within the environment, are intended to be buy or sell orders.

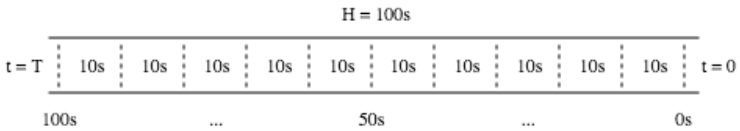


Figure 5.2: Segmented time horizon H with remaining time t.

The time horizon H defines the amount of time given in order fill an order. The default time horizon is 100 seconds for reasons described in Section 2.3, which reflects the Good-Till-Time (see 2.1.2) of the order. As illustrated in Figure 5.2, the time horizon is segmented into discrete time steps t and therefore limits the

number of steps an agent can take within one epoch. Each step is of the same length Δt, which is for illustration purposes set to 10 seconds. We pick T as the maximum value of t, indicating that the entire amount of time is remaining, whereas $t = 0$ states that the time horizon is consumed. Consequently, within a single epoch, the GTT of the order is being adjusted to Δt for each step, still resulting in a total GTT of H. Lastly, the *feature type* F determines the state observed by the agent, which represents one of the features described in Section 4.4.

5.1.3. State

Unlike in most traditional reinforcement learning environments, each step taken by the agent leads to a complete change of the state space. Consider a chess board environment, where the state space is the board equipped with figures. After every move taken by the agent, the state space would look exactly the same, except of the figures moved with that step. The epoch would continue until the agent either wins or looses the game and the state space would be reset to the very same setup for each epoch. In the order placement environment however, it is, as if, for each step not only one or two figures of the chess board change their position, but almost all of them. And a reset of the environment will result in an ever changing setup of the figures on the chessboard. The reason for this is that the chessboard is in our case the order book that underlies a time series which evolves over time. More precisely, the state space S is defined as a sequence of order book states from which an agent can observe an observation state O at some point in time. The final state is reached when the entire inventory was bought or sold, that is $i = 0$–Checkmate!.

There are two general types of variables that can be used in order to create an observation state: *private variables* and *market variables* [29]. For private variables, the size of the state space depends on the V shares that have to be bought or sold and the given time horizon H, resulting in a state $s \in R^2$. Market variables can be any information derived from the order book at a given moment in time. Therefore the specified feature set (see Section 4.4 below) defines the dimension of the state the agent observes. Consequently, market variables increase the state space drastically, due to (1) the initialization of the environment using a random order book state and (2) the dimensionality of the feature set. Hence, for each step taken by the agent, the order book states are likely to be different and thus the state the agent observes changes equally.

5.1.4. Action

A discrete action space A is a vector $(a_{min}, ..., a_{max})$ that represents a range of relative limit levels an agent can choose from in order to place an order. That is, how deep (a_{min}) and how high (a_{max}) the order can be placed in the book. The action $a \in A$ is an offset relative to the market price p_{m^T} before the order was placed (at time step $t = T$). Negative limit levels indicate the listing deep in the book and positive listings relate to the level in the opposing side of the book. Hence, the price of the order placement p at some time step t is $p_t = p_{m^T} + a_i * \Delta a$, whereas Δa is the step size of an action. An illustration of this concept is given in Figure 5.3.

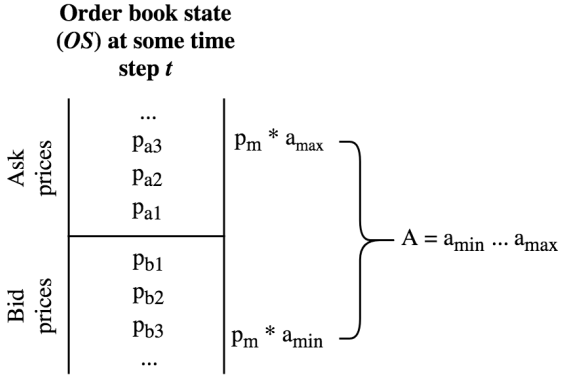


Figure 5.3: Actions represent an offset relative to the order price at which to place the order in some order book state OS_i at some time step t.

By default the action step size $\Delta a = \$0.10$. For example, with $|A| = 5$ that is $(p_{m^T} - 0.2, p_{m^T} - 0.1, p_{m^T}, p_{m^T} + 0.1, p_{m^T} + 0.2)$. The action space is configurable and the default implementation is of size $|A| = 101$, indicating that $a_{min} = -50$ and $a_{max=50}$ result in an order price $p = p_{m^T} - \$5$ and $p = p_{m^T} + \$5$ respectively.

5.1.5. Reward

As described in Section 2.3, the volume weighted average price (see Eq. 2.10 serves as a measure of the *return* of order placement. Consequently, the *reward* is defined as the difference of the market price before the order was placed p_{m^T} and the volume weighted average price paid or received after the order has been filled. Hence, for buying assets the reward is defined as $r = p_{m^T} - p_{vwap}$ and for selling assets $r = p_{vwap} - p_{m^T}$. In case no trades resulted during the matching process, the reward is $r = 0$, indicating that no direct negative reward is provided. Reasons for this are that in case the order could not be matched over the course of the time horizon, when $t = 0$, a market order follows which will likely produces trades with a worse price than the market price before the placement has started. Given the definition of the discounted return (Eq. 2.13) we calculate $R_t = \sum_{t'=t}^{t_0} \gamma^{t'-t} * r_{t'}$, where t_0 is the time step at which the agent has its time horizon for the ongoing order fully consumed.

5.2. Market making Environment

TODO

5.3. Q-Learning agent

The agent described in this section is generally known as *Q-Learning*[34]. In this work, Q-Learning serves to (1) optimize order placement by using private variables only and (2) to have a measure of comparison while evaluating possible advantages of featuring raw market data by using a Deep Q-Network agent (see Section 5.4 below), which is an extension of the Q-Learning agent.

The name Q-Learning refers to the application of the previously presented Q-function (Eq. 2.17). More specifically, it relies on the *action-value function* (Eq. 2.18) that obeys an important identity known as the *Bellman equation*. The intuition is that: if the optimal value action-value $Q^*(s', a')$ of the state s' at the next time step $t+1$ was known for all possible actions a', the optimal strategy is to select the action a' which maximizes the expected value of $r + \gamma * Q^*(s', a')$,

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a')] \; \forall s \in s, a \in A \tag{5.1}$$

, whereas $0 \leq \gamma \leq 1$ is the discount rate which determines how mach future rewards are worth, compared to the value of immediate rewards. The aim of the iterative value approach is to estimate the action-value function by using the Bellman equation as an iterative update,

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s', a')] \tag{5.2}$$

Value iteration algorithms then converge to the *optimal action-value* function $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$. [33]

Q-Learning makes use of the aforementioned Bellman equation (Eq. 5.1) that undergoes an iterative update. The algorithm has proven to be an efficient and effective choice to solve problems in a discrete state space. Limitations of this approach are known when the agent is applied to large or continuous state spaces [18]. This becomes more apparent when considering the presented algorithm above. The iterative update of the action-value function $Q(s,a)$ (defined in Eq. 2.18 and used in Eq. 5.1) is exposed to the size of the state s and action a, and thus if s is chosen too large, the optimal policy $\pi^*(s)$ (defined in Eq. 2.20) will likely not converge. As a result, the features derived in Chapter 4 are not applicable for this agent.

However, private variables of the environment, as described in Section 5.1.3, respect the aforementioned limitations. As a result, the observation the Q-Learning agent will receive from the environment is defined by the discrete inventory unit i and time step t, that is, $s = (i, t)$.

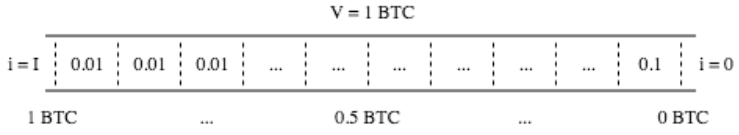


Figure 5.4: Inventory of V segmented shares with a remaining inventory i.

However, fractions of shares and time which evolve during the matching process would result in a vastly large state space. Therefore, similar to the discrete time steps which are described above, the V shares are divided into discrete inventory units i of size Δi, as illustrated in Figure 5.4. The inventory units approximate the order size in order for our policy to distinguish between when creating an order. We pick I as the maximum value of i, indicating that the entire inventory remains to be filled. The order is considered as filled when $i = 0$, meaning that no inventory is left. Given the inventory units and the time steps, the state space remains $s \in R^2$ but becomes much smaller in its size, namely $I \times H$. In the default setup a segmentation of 0.01 BTC steps is applied. For example, if the initial inventory is 1.0 BTC and the order is partially filled with 0.015 BTC during an epoch, the remaining inventory is 0.99 BTC (instead of 0.985) for the next step the agent will take.

Algorithm 1 Q-Learning algorithm

```
Initialize Q(s, a) arbitrarily
for each episode do
    for t=0...T do
        for i=0...I do
            s = (i, t)
            Choose a from s using π derived from Q (ε-greedy)
            Take action a, observe r, s'
            Q(s, a) ← Q(s, a) + α[r + γ max_a' Q(s', a') − Q(s, a)]
            s ← s'
```

Finall, Algorithm 1 describes the Q-Learning algorithm used in this work. An adaption was made to a conventional Q-Learning algorithm[33] regarding the steps the agents will proceed, and therefore follows the same concept as presented in [29]. The agent solves the order placement problem inductively, starting off the episode at state $s = (i, 0)$, when $t = 0$. This has the benefit that the environment enforces a market order (see Section 5.1.5) at the beginning of an epoch, for all inventory units i and therefore provides immediate reward to the agent. The agent then increments i and t independently and the previously seen reward will serve as the future reward as the agent increases i and t. As a result, for each epoch, the agent takes $I * T$ steps.

Apart from that, the algorithm follows the standard procedure. The Q function is updated given the state s and action a. Therefore, the existent estimate of the action-value function is subtracted from the value of the action that is estimated to return the maximum future reward. In addition, a learning rate α is introduced that states to which extent new information should override previously learned information, with weighting $0 \leq \alpha \leq 1$. Eventually, the agent completes the episode and at this point, every combination of t and i has been visited by the agent. Hence, the agent has learned each discrete step i and t in the process of buying or selling V within the time horizon H.

5.4. Deep Q-Network agent

The second agent, which is presented in this section, is known as Deep Q-Network (mnih2015human) [27]. DQN is a deep reinforcement learning method that combines reinforcement learning with a class of artificial neural network known as deep neural networks.

describe NN

Similar to the Q-Learning approach described above, the Q-values should obey the Bellman equation 5.1. The neural network treats the right-hand side, with weights ω, as a target, that is, $r + \gamma \max_{a'} Q^*(s', a', \omega)$. We then minimize the mean squared error (MSE) by stochastic gradient descent,

$$L = (r + \gamma \max_{a'} Q^*(s', a', \omega') - Q(s, a, \omega))^2 \tag{5.3}$$

Whereas the optimal q-value converges for the Q-Learning approach that uses a look-up table, the use of a non-linear function approximator can cause the convergence due to (1) correlation between samples and (2) non-stationary targets.

In order to remove correlations we use *experience replay* with which we build a data set D from the agent's own experience. Therefore we store the agent's experience $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step t in the data set, such that $D_t = e_1, ..., e_t$. During learning process, Q-learning updates on samples (or mini-batches) of these experience $(s, a, r, s') \sim U(D)$ which are drawn uniformly at random from the pool of stored samples in

D. Hence, we prevent the learner from developing a pattern from the sequential nature of the experiences the agent observes throughout one epoch. In our case this might be the case during a significant rise or fall of the market price. In addition, experience replay stores rare experiences for much longer such that the agent can learn these more often. That is for example when massive subsequent buy order led to a noticeable change in the order book.

In order to deal with non-stationarity, the target network parameters δ' are only updated with δ every C steps and otherwise unchanged between individual updates.

6

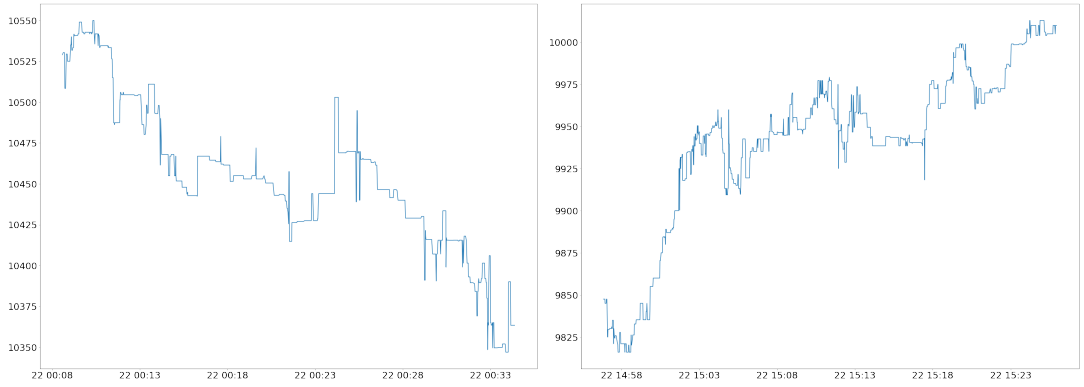
Analysis and discussion

In the previous Chapter we have built a reinforcement learning environment with the use of the components which were described earlier in Chapter 2. The environment allows to simulate order placement on a historical order book that was described in Chapter 4. Furthermore, two agents were introduced, a Q-Learner which learns on private variables and a Deep Q-Network which learns on market variables.

The aim of this chapter is to run simulations and observe whether or not reinforcement learning is indeed capable of optimizing the placement of limit orders. Throughout this chapter we make use of real world order books as well artificially created order books, whereas the latter allow to define distinctive price trends and eliminate the noise present in real market data. We first present the data sets chosen for this analysis. Before evaluating the learners, we investigate the reinforcement learning environment empirically by simulating an agent that places buy and sell orders at a range of limit levels. This will provide knowledge of how well we should expect the reinforcement learners to perform. Subsequently we make an attempt to build a strategy based on the private variables only, with the use of the Q-Learner. This gives insight of the performance of a naive reinforcement learner and serves as a benchmark for the following simulations proceeded in which we consider market variables. While applying market variables to the DQN agent we make use of both features, price and size of historical order as well as the price and size of historical trades, separately. In doing this, we determine the capabilities and limitations not only by evaluating the received rewards but also by looking at the submitted actions of the agent. Since we are interested in the general ability for reinforcement learning to learn how to place orders, potential maker or taker fees are neglected in this setup.

6.1. Data sets

We have selected two ~30 minute samples of historical order book recordings with which we proceed experiments in this chapter. Thereby we have consciously chosen one sample (I) order book with downwards trend (bid/ask mid-price) and the other sample (II) with an upwards trend, as shown in Figure 6.1. The sample in Figure 6.1a consists of 1132 order book states with a duration of 1681.8 seconds, resulting in 0.67 states per second. The sample in Figure 6.1b consists of 1469 order book states with a duration of 1746.0 seconds, resulting in 0.84 states per second, indicating that there was slightly more pressure in terms of orders placed and cancelled in this data set.



(a) 30 minute downwards trend

(b) 30 minute upwards trend

Figure 6.1: Bid/ask mid-price of 30 minute order book recordings.

Reasons for choosing two very distinguishable data sets include to determine the ability of the learners to react on a variety of market situations. As explained in the following section, the expected return of a learner for buying and selling assets heavily depends on the market price movement and therefore the situations become very different for the data sets in use.

When reinforcement learning is applied, the data sets are split with ratio 2 : 1, resulting in a training set of ~20 minutes and a test set of ~10 minutes.

6.2. An empirical investigation of the reinforcement learning environment

This section serves to investigate the relationship between the limit order placement and the received reward, within a fixed time horizon. The demonstrated methods are based on the related work described in Section 3.1 and provide the ability to empirically evaluate the reinforcement learning environment (Chapter 5) by simulating the behaviour of an agent that buys and sells shares at every possible limit level and records the received rewards (e.g. immediate returns). The return is denoted by the difference between the market price prior the order placement and the volume weighted average price (VWAP) paid, respectively received, as stated in Eq. 5.1.5. As a result we gain understanding of the behaviour of order placement within a given historical market and set a baseline for the reinforcement learners to come.

We recapitulate that according to [28, 35] there are three obvious trading strategies in order to determine the execution price of an order (considering limit and market order types only):

1. Submit a market order tor the entire amount immediately.
2. Wait until the end of the time period and then go to the market with the entire amount.
3. Submit a limit order at the beginning of the time period; then submitting a market order for the remainder of shares (if any) at the end of the interval.

Having a total time horizon of 100 seconds available, the last approach is of interested in this analysis. However, we investigate the behaviour on progressive increasing time horizons, starting from 10 seconds up to 100 seconds, as this demonstrates the discrete time steps to be taken by a learner. By doing so, the expected return is observed by the average return of placing (e.g. cross-validating) 100 orders of size 1.0 BTC at a random point in the given data set. The price of the orders are determined by a range of 201 limit levels (−100...100) with step size $0.10, resulting in orders priced in the range of $p_m - 10 \ldots p_m + 10$, whereas p_m is the market price before the order was placed.

6.2.1. Order placement behaviour on data set I

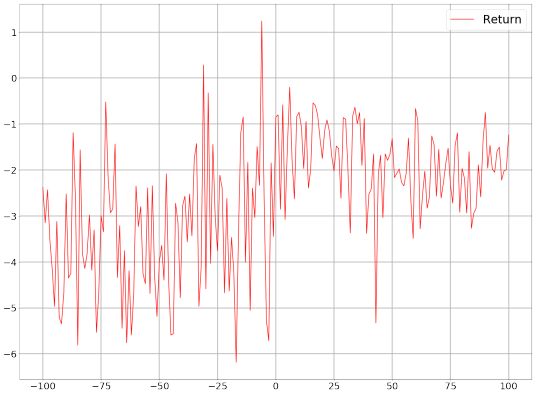
For the data set I, where the market undergoes a downwards trend, the intuition is as follows: We expect buy orders to result in better returns when placing deep in the order book, meaning with a highly negative limit level. Since the price tends to fall, the assumption is that an agent is able to buy for a lower price once time has passed. Therefore, the longer the time horizon, the lower the limit level can be chosen in order to still be able to execute the full amount of shares. Contrarily, we expect sell orders to provide better returns when the agent crosses the spread with a positive limit level. The assumption is that in a falling market it

is unlikely that market participants are willing to buy for higher prices and therefore the agent must place sell orders higher in the book in order to sell immediately. Otherwise, the longer the time horizon, the less return an agent would retrieve as the market order after the order has not been filled becomes costly. We proceed this investigation within our reinforcement learning environment as shown Figure 6.2. Therefore, time horizons of 10, 30, 60 and 100 seconds (y-axis) and limit levels reaching from -100 to +100 (x-axis) were chosen. The time horizons determine the various situations an agent is confronted while proceeding steps within an epoch. The limit levels are chosen broadly in order to retrieve understanding about the outcome of a variety of possible actions.

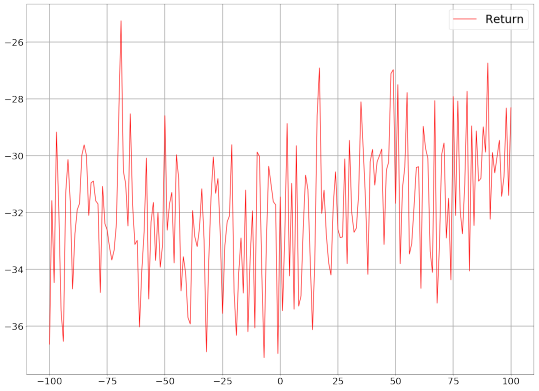
With a time horizon of only 10 seconds left, the expected behaviour is, however, proven wrong. For buy orders, shown in Figure 6.2a, the returns suggest to place the orders close to the spread, but still on the opposing side, at a limit level of ~+5. The spike at limit level ~-5 indicates that the overall best return was provided at this level, however it comes with the risk that the orders fails to execute, indicated by the downwards spike also close to level ~-5. For selling within 10 seconds, as shown in Figure 6.3b, the best return is given when crossing the spread with a positive limit level of ~+50. The spike at limit level ~-70 is likely caused by one of the orders during the cross-validation process, that was able to execute and therefore contributed to greater return.

With an increased time horizon of a total of 30 seconds, as shown in Figures 6.2c and 6.2d, the expected behaviour becomes more evident. Positive returns can be achieved by posting buy orders deep in the order book, whereas there is not much variance between the negative limit levels itself. Therefore, we can expect that in the given market situation the agent was able to execute the order partially at very low limit levels and for the unexecuted part a market order followed which ultimately averaged the price to be similar as when the agent placed the order initially at a price which is only slightly below the spread. This is confirmed by the fact that the most dense range of positive returns is indeed around the limit levels just below the spread. Crossing the spread causes increasingly lower returns, the more positive the limit level is chosen. That is a result of agents willingness to immediately buy for an increasing price by using market orders. The opposite effect occurs while selling assets. Market orders higher in the book result in result in better returns than limit orders deep in the book. Interestingly, orders which were placed very deep in the book, at limit level ~-50 and below, are rewarded better than the ones close to the spread. This is most likely a consequence of a minority of orders which were partially filled at this level during the cross-validation process.

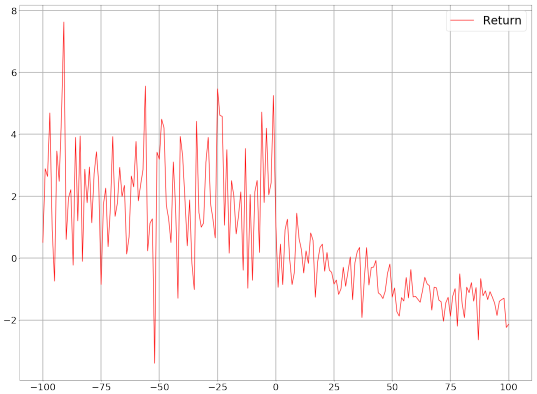
With time horizons of 60 and 100 seconds the expected behaviour of the orders is clearly given. Buy orders as shown in Figures 6.2e and 6.2g, best best off when placed very deep in the order book. However, when placed too deep, at level -100, the return is slightly less as a result of unexecuted orders which had to be filled with market orders once the time was consumed. In addition, positive limit levels become stable since there are more sellers in the market with the extended time horizon and therefore very high placed orders have the same effect as limit orders posted only slightly above the spread. The same applies to sell orders which are placed very deep in the book, as shown in Figures 6.3f and 6.3h. Placing orders very deep in the book have the same effect as when placing the order just below the spread, that is, there are no traders willing to buy at such a high price and therefore market orders follow once the time has passed.



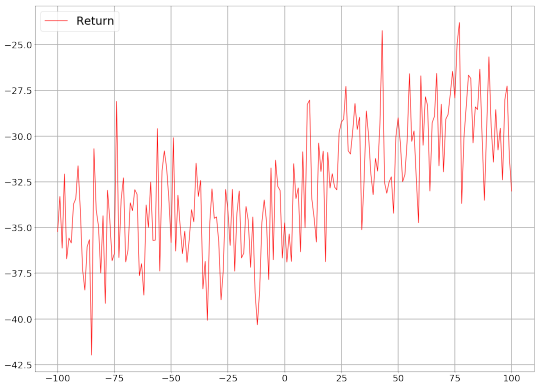
(a) Returns of buy orders within 10 seconds



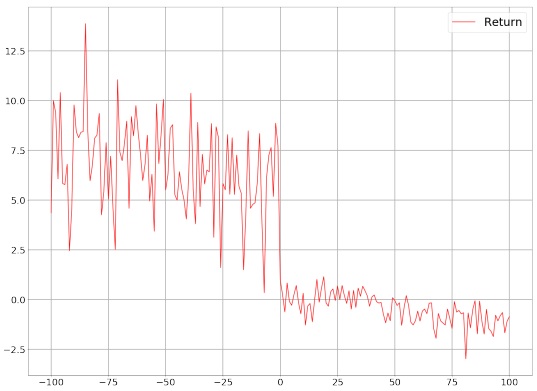
(b) Returns of sell orders within 10 seconds



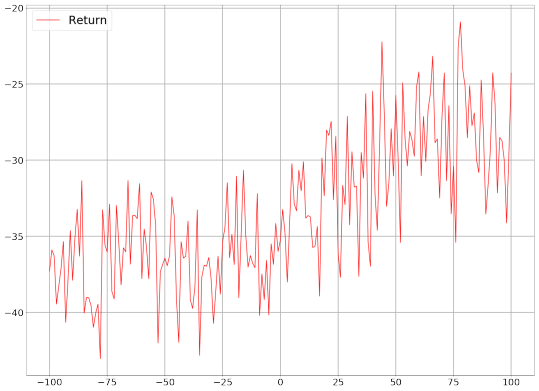
(c) Returns of buy orders within 30 seconds



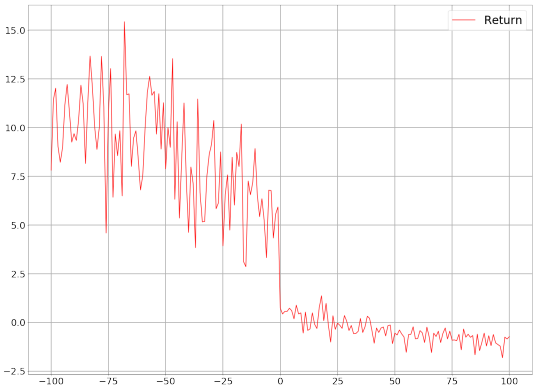
(d) Returns of sell orders within 30 seconds



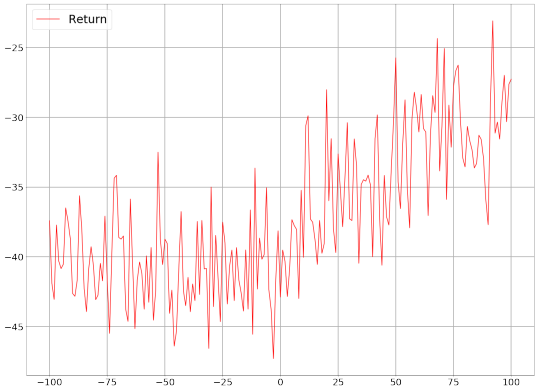
(e) Returns of buy orders within 60 seconds



(f) Returns of sell orders 60 seconds



(g) Returns of buy orders 100 seconds



(h) Returns of sell orders 100 seconds

Figure 6.2: Returns of buy and sell orders executed within 10, 30, 60 and 100 seconds on data set I.

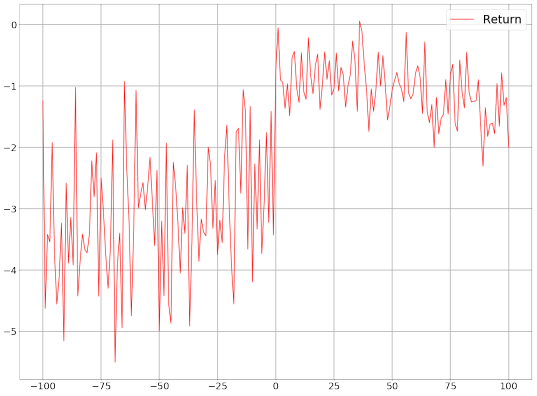
6.2.2. Order placement behaviour on data set II

For the data set II, which contains an upwards trend, the intuition is the opposite as during the investigation of data set I. Namely, we expect buy orders to result in better returns when immediately filled, that is when the agent crosses the spread and places the order high in the book. The assumption is that as time passes and the market price rises, other traders become less willing to sell for the market price or lower. Therefore, the longer the time horizon given to the agent, the more critical it becomes to execute immediately, as otherwise shares would have to be bought to an increased market price. Likewise, better returns of sell orders are expected when placed deep in the book, meaning to be sold at a higher price. The assumption is that as the price rises, market participants become more likely to buy assets for higher prices. Hence, the longer the time horizon, the deeper the agent should place a limit sell order in the book, as this will likely not require a following market order due to (partially) unexecuted shares. We investigate these assumptions by proceeding the same experiment as in the previous section, as shown in Figure 6.3, whereas time horizons of 10, 30, 60 and 100 seconds (y-axis) and limit levels reaching from -100 to +100 (x-axis) were chosen

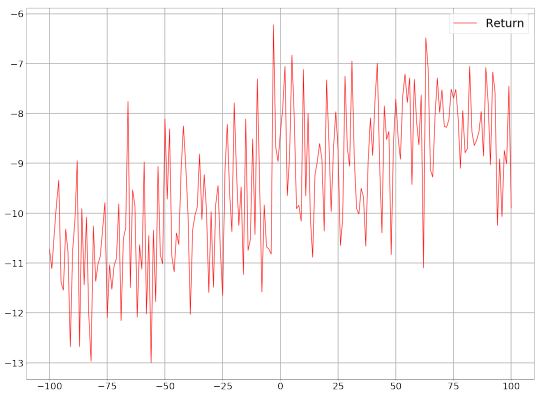
The returns of buy orders with a time horizon of 10 seconds, as shown in Figure 6.3a, correlate with the above stated assumptions. That is, highest returns are achieved when crossing the spread and although limit levels in the range of 1-50 tend to perform the same, the wises choice for the agent would be to choose the one closest to the spread as it comes with the least risk of paying a premium. With the same time horizon, the sell orders placed contradict the assumptions, as shown in Figure 6.3b. Here the agent is rewarded the most when choosing a price for the order at market price, as indicated by the limit level 0, that is on the spread. A highly negative limit level causes to receive approximately $3.00 less than when placing at the suggested market price.

With 30 seconds left to buy 1.0 BTC, in Figure 6.3c, the orders placed above the spread become stable for any such limit level, much more so than in the previous investigation with the data set I. This is likely due to the higher order pressure of the data set II, as described in Section 6.1. Hence, there are more market participants willing to sell. The return curve that indicates sell orders placed by an agent, shown in Figure 6.3d, shifts towards a more evenly distributed returns compare to when only 10 seconds were left. Therefore, limit orders tend to become more rewarding and an agent might benefit from a slight increase in price within the given time horizon.

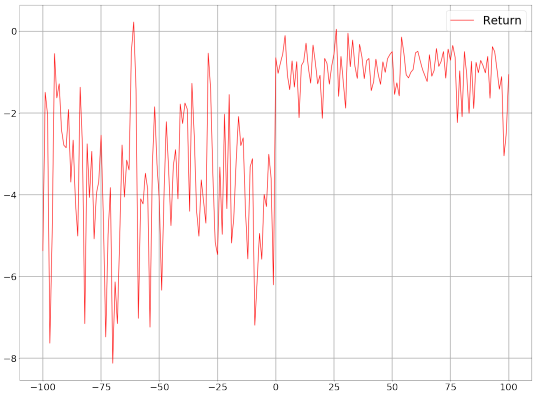
Even more so, this pattern becomes evident when a time horizon of 60 and 100 seconds were given, as shown in Figures 6.3f and 6.3h respectively. With the increased time horizon the assumptions stated in the beginning of this section are confirmed by shown that the agent, when trying to sell shares, should place orders indeed deep in the order book. When time passes and the market price rises, market participants are willing to buy for an increasing price and an agent is able to sell all assets for such an increased price without the need of a following market order. Contrarily, if the agent decides to offer to sell the assets for a decreasing price, as indicated by the higher limit levels above the spread, the less reward would be given. More precisely, for a time horizon of 100 seconds, the agent would receive up to $7.00 less when choosing to cross the spread with a limit level of +100 compared to some negative limit level. Figures 6.3e and 6.3g which show the result of an agent that tries to buy assets within the increased time horizon, the behaviour is clear. That is, during an uprising market, the damage can be minimized by crossing the spread and buying immediately. The advice stated before remains, that is, the agent should choose a price a few steps ($0.10) above the market price as there is enough liquidity in the market to buy the demanded number of assets.



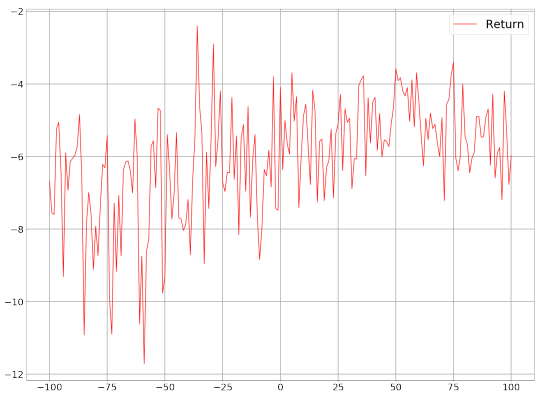
(a) Returns of buy orders within 10 seconds



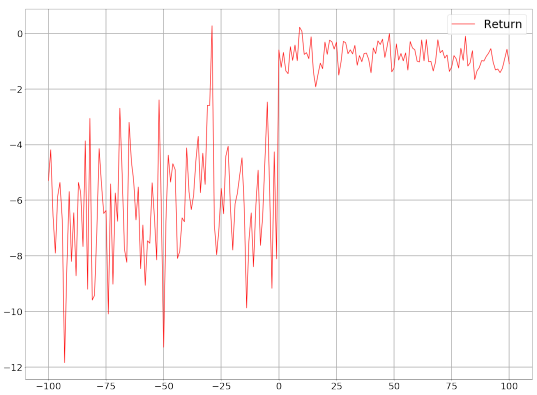
(b) Returns of sell orders within 10 seconds



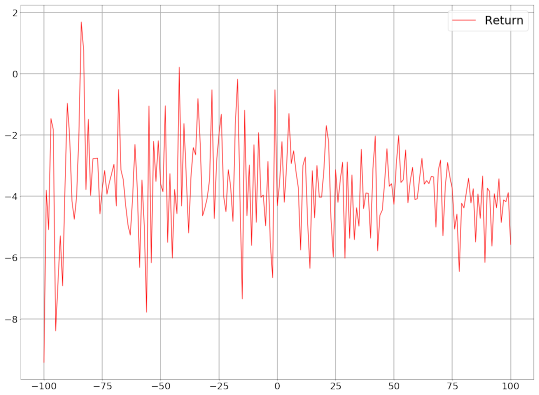
(c) Returns of buy orders within 30 seconds



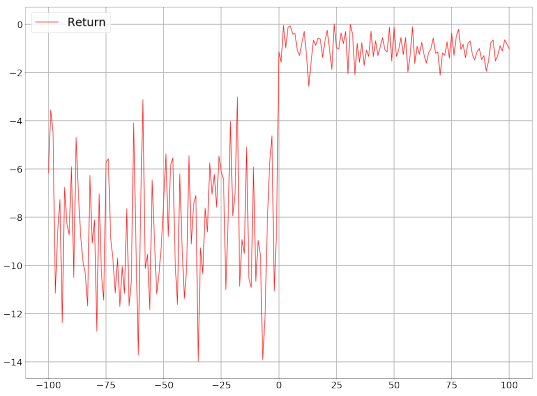
(d) Returns of sell orders within 30 seconds



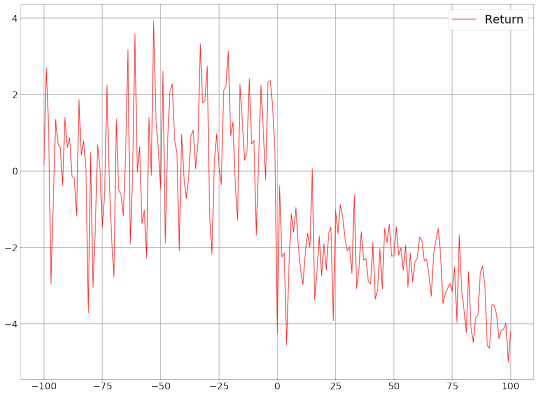
(e) Returns of buy orders within 60 seconds



(f) Returns of sell orders 60 seconds



(g) Returns of buy orders 100 seconds



(h) Returns of sell orders 100 seconds

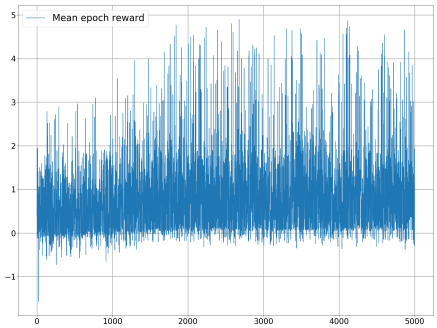
Figure 6.3: Returns of buy and sell orders executed within 10, 30, 60 and 100 seconds on data set II.

6.3. Q-Learning without market variables

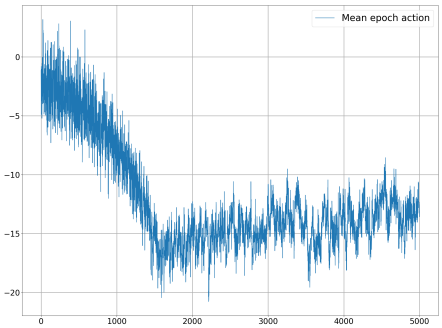
The previous section provided knowledge about the possibilities of the agent when placing buy and sell orders using the reinforcement learning environment and with the underlying data set I and II. The expected behaviour for each limit level has been observed under two significantly different market situations. For each such observation, a fixed time horizon was chosen for which an order was residing in the order book, followed by a market order in case the order has not been filled completely. It has been shown that during an upwards trend, market participants are willing to buy and sell shares for higher prices and contrarily, during a downwards trend for lower prices. However, it has been observed that an agent would therefore indeed be able to, if limit orders were placed accordingly, minimize the price to pay, respectively maximize the price for which to receive, the assets.

This section aims to investigate whether or not a Q-Learning agent, as described in Chapter 5 (Section 5.3), can reproduce the optimal achieved results shown in the previous section. Therefore the agent is allowed to cancel its order after every 10 seconds and place a new order with the remaining inventory, until the time horizon is fully consumed. For both data sets (I and II) an independent learning experiment is being proceeded, whereas the agent is supposed to either buy or sell shares. For each such experiment, the training is limited to 5000 epochs and 1000 orders are being back-tested on the test set. The Q-Learning agent is set up as follows: the learning rate $\alpha = 0.1$ is chosen small due to extensive amounts of steps the agent will make throughout the epochs. The discount factor $\gamma = 0.7$ is chosen slightly in favour of immediate rewards in the hope to prevent the agent from running out of time which would result in a following market order. Initially, the exploration constant ϵ is set to 0.1 but then multiplied by an epsilon decay factor for each step taken by the agent, such that $\epsilon = \sim 0.8$ once training is completed. This allows the agent first to explore the action space and then exploit on the learned optimal actions to take.

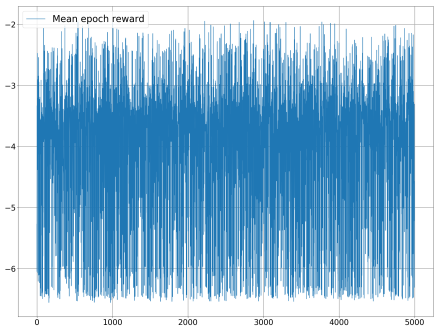
As a result of this setup, four observations were made and for each of which the training and testing results are stated below. During training, the mean rewards given and average action chosen for each epoch are recorded. Using the training model, a backtest is run on the test data sets, whereas the agent executes orders and chooses the learned optimal action to take. The result of which is shown as the average reward received while doing so. The difference to the occurring costs of a market order then serves as the guideline of how well the algorithm performs.



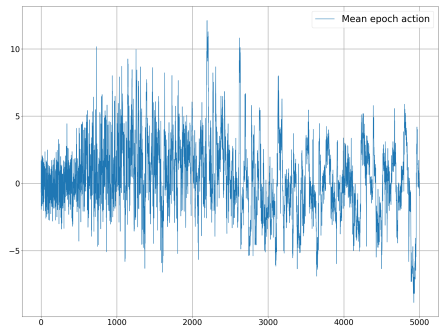
(a) Mean rewards per epoch (buy)



(b) Mean actions per epoch (buy)



(c) Mean rewards per epoch (sell)

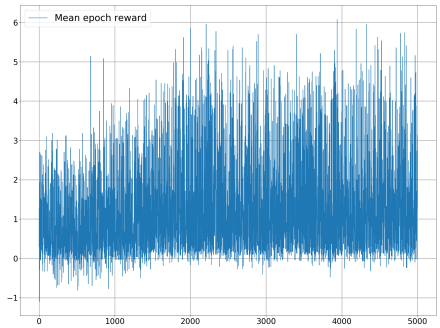


(d) Mean actions per epoch (sell)

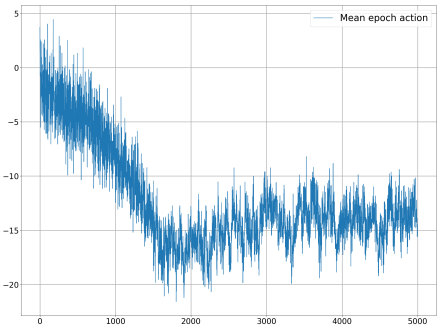
Figure 6.4: Mean rewards and actions for buying and selling on training data set I.

Figure 6.4 shows the experiment proceeded on data set I. The average received reward during the training where the agents task was to buy 1.0 BTC within 100 seconds is shown in Figure 6.4a. Over the course of 5000 epochs, the agent was able to improve the mean reward slightly, by ~0.5. Reasons for this is the change in chosen actions as illustrated in Figure 6.4b. The agent started off with an average action of ~-3 which is a result of the low epsilon parameter that makes the agent choose actions randomly. Actions where then adapted to the more negative side of the order book, such that after ~1500 epochs the agent choose actions as low as -20 and then adjusted and stagnated at ~-15. The backtest, during which 1000 orders were executed on the test data set, resulted in an average reward of *-1.17*–that is worse than the average reward received on the training close to the end of the training. The cost of a market order on the test data set is -0.05 such that the strategy of the agent results in additional costs of $-1.12 when buying 1.0 BTC. Comparing the found results to the empirical analysis proceeded on the same data set, as shown in Figure 6.2, provides means for interpretation: Considering the backtest results and the highly negative average action the agent chooses towards the end of the training indicates that the order was oftentimes not able to be filled within the time horizon and a market order was followed.

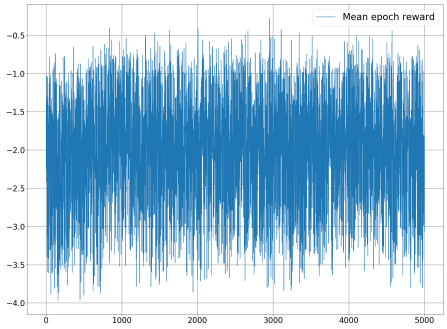
The rewards received for the agents tasks to sell the assets are much more volatile, as shown in Figure 6.4c, and no clear improvement can be seen. Consequently, there was no significant adjustment made by the agent regarding the chosen actions, as indicated in Figure 6.4d. The agent started off at limit level 0 and after some exploration concluded to keep choosing actions at the same level as it started off. The backtest resulted in an average reward of -21.34 achieved by the agent. The reward received for placing market orders on the test set account to a negative reward of -27.70. Hence, the agent is able to save $6.36 when selling 1.0 BTC.



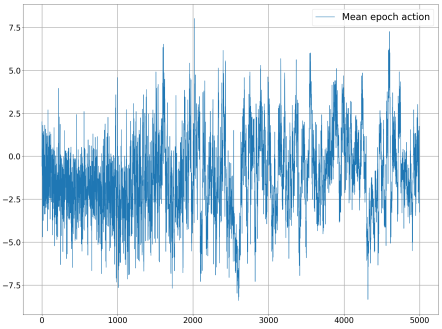
(a) Mean rewards per epoch (buy)



(b) Mean actions per epoch (buy)



(c) Mean rewards per epoch (sell)



(d) Mean actions per epoch (sell)

Figure 6.5: Mean rewards and actions for buying and selling on training data set II.

Figure 6.5 shows the experiment proceeded on data set II. The average received reward while training to buy the asset is shown in Figure 6.5a. Throughout the epochs, the agent was able to improve the mean reward like with the previous data by roughly 0.5. Even though the trend of this data set is the opposite, the change in chosen actions correlates to the previous findings and is illustrated in Figure 6.5b. The backtest on the test data set (of data set II), resulted in an average reward of *-1.04* – again worse than the average reward received on the training set. A market order on this test data accounts to an average reward of -1.06, indicating that the agent is saving $0.02 when buying 1.0 BTC. When taking the empirical analysis proceeded on this data set into consideration, as shown in Figure 6.3, the received rewards indicate that the agent again failed to execute orders with the placed limit orders and oftentimes market orders followed.

Similar to the order placed on data set I, the rewards received for the agents tasks to sell the assets are much more volatile when it comes to selling, as shown in Figure 6.4c. No improvement can be seen from the rewards during the training as no significant adjustment was made by the agent regarding the chosen actions, as indicated in Figure 6.5d. The backtest resulted in an average reward of -4.74 achieved by the agent, whereas market orders result to an average reward of -1.72. Hence, the agent causes to pay a premium of $3.02 for selling 1.0 BTC.

	Agent	**Market Order**
Buy (I)	-1.17	-0.05
Sell (I)	-21.34	-27.70
Buy (II)	-1.04	-1.06
Sell (II)	-4.74	-1.72

Table 6.1: Summary of rewards for the Q-Learning agent and market orders.

The findings of this section are summarized in Table 6.1. We conclude that the Q-Learning agent was not

able to place buy and sell orders in a way which would result in a price better than the current market price. Oftentimes, a market order which would cause an immediate buy or sell would even be the better choice. Clearly, this is due to the fact that the agent was not able to find the most suitable actions. Furthermore, in order to investigate whether or not these results were a result of the agent aiming for too much immediate reward, the same experiment was proceeded with $\gamma = 0.3$. However, no improvement could be made and instead the agent achieved similar results while requiring more epochs in order to converge to the same mean of actions.

In this section we have only investigated the mean of the actions chosen throughout an epoch, which gave enough proofs that the chosen actions resulted mostly in market orders. In the following section, where the DQN agent is being investigated, we plan to investigate the chosen sequence of actions within one epoch in order to gain even better understanding about the decision process of the agent.

6.4. Deep Q-Network on execution

6.5. Deep Q-Network on market making

6.6. Deep Q-Network with event flow data

7

Conclusion and Future Work

7.1. Conclusion

simulation of historical order matching

model parameters

7.2. Future Work

Bibliography

[1] Bottom-up investing. URL https://www.investopedia.com/terms/b/bottomupinvesting.asp. [Online; accessed April 30, 2018].

[2] Cs 294: Deep reinforcement learning. URL http://rll.berkeley.edu/deeprlcourse/. [Online; accessed April 30, 2018].

[3] Fundamental analysis. URL https://www.investopedia.com/terms/f/fundamentalanalysis.asp. [Online; accessed April 30, 2018].

[4] Introduction to time series analysis. URL https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4.htm. [Online; accessed April 30, 2018].

[5] Matching algorithms. URL https://www.cmegroup.com/confluence/display/EPICSANDBOX/Matching+Algorithms. [Online; accessed April 30, 2018].

[6] Enrique martinez miranda. URL https://nms.kcl.ac.uk/rll/enrique-miranda/index.html. [Online; accessed April 30, 2018].

[7] Deep reinforcement learning demysitifed (episode 2), . URL https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed\-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa. [Online; accessed April 30, 2018].

[8] Reinforcement learning demystified, . URL https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14. [Online; accessed April 30, 2018].

[9] Limit orders, . URL https://www.sec.gov/fast-answers/answerslimithtm.html. [Online; accessed April 30, 2018].

[10] Market order, . URL https://www.investor.gov/additional-resources/general-resources/glossary/market-order. [Online; accessed April 30, 2018].

[11] Stock exchange history. URL https://www.investopedia.com/articles/07/stock-exchange-history.asp. [Online; accessed April 30, 2018].

[12] Top-down investing. URL https://www.investopedia.com/terms/t/topdowninvesting.asp. [Online; accessed April 30, 2018].

[13] Technical analysis. URL https://www.investopedia.com/terms/f/technicalanalysis.asp. [Online; accessed April 30, 2018].

[14] Time series. URL https://en.wikipedia.org/wiki/Time_series. [Online; accessed April 30, 2018].

[15] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[16] Chris Chatfield. *Time-series forecasting*. CRC Press, 2000.

[17] Tristan Fletcher, Zakria Hussain, and John Shawe-Taylor. Multiple kernel learning on the limit order book. In *Proceedings of the First Workshop on Applications of Pattern Analysis*, pages 167–174, 2010.

[18] Chris Gaskett et al. Q-learning for robot control. 2002.

[19] Xin Guo, Adrien de Larrard, and Zhao Ruan. Optimal placement in a limit order book. *Preprint*, 2013.

[20] Ted Hwang, Samuel Norris, Hang Su, Zhaoming Wu, and Yiding Zhao. Deep reinforcement learning for pairs trading.

[21] Kiyosi Itô. *Encyclopedic dictionary of mathematics*, volume 1. MIT press, 1993.

[22] Marcus Lim and Richard J Coggins. Optimal trade execution: an evolutionary approach. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1045–1052. IEEE, 2005.

[23] David W Lu. Agent inspired trading using recurrent reinforcement learning and lstm neural networks. *arXiv preprint arXiv:1707.07338*, 2017.

[24] Burton G Malkiel. Efficient market hypothesis. In *Finance*, pages 127–134. Springer, 1989.

[25] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[28] Yuriy Nevmyvaka, Michael Kearns, M Papandreou, and Katia Sycara. Electronic trading in order-driven markets: efficient execution. In *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on*, pages 190–197. IEEE, 2005.

[29] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[30] Scott Patterson. *Dark pools: The rise of AI trading machines and the looming threat to Wall Street*. Random House, 2012.

[31] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[32] Robert H Shumway and David S Stoffer. Time series analysis and its applications. *Studies In Informatics And Control*, 9(4):375–376, 2000.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[34] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[35] Chaiyakorn Yingsaeree. *Algorithmic trading: Model of execution probability and order placement strategy*. PhD thesis, UCL (University College London), 2012.