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

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 of *order placement*.

Our findings show...

Preface

Preface…

Marc B. Juchli
Delft, January 2013

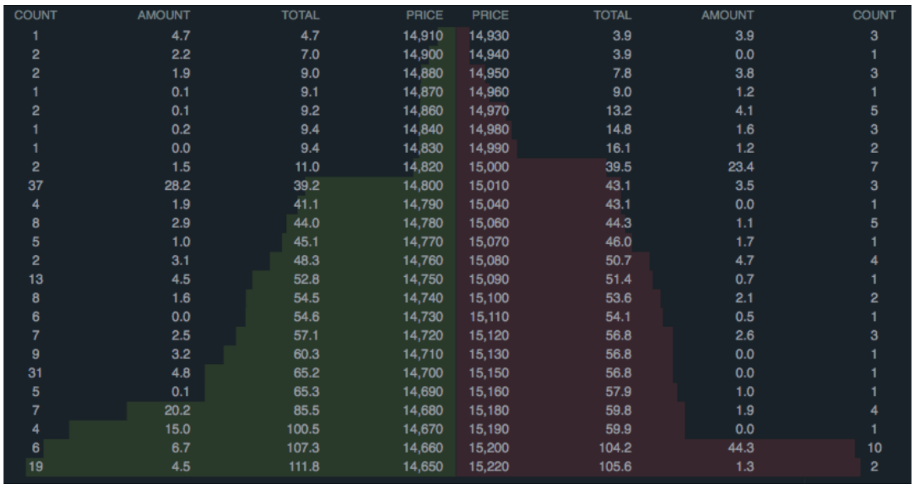
Contents

1 Introduction

Financial institutions make decisions to buy or sell assets for many reasons, including: customer requests, fundamental analysis[3], technical analysis[12], top-down investing[11], and bottom-up investing[1]. High-level trading strategies oftentimes define how an institution positions itself in financial markets and, if applicable, towards its customers. Regardless of the high-level trading strategy that is being applied, the invariable outcome is a decision to buy or sell assets. This work aims to take a step towards answering the important question of how one can optimize a purchase or sale of an asset on a stock exchange with the use of reinforcement learning techniques. The subsequent sections will elaborate this problem briefly and state the research objectives of this work. We will list the contributions made to 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 on stock exchanges. There is little consensus as to when corporate stock was first traded; some argue that the exchange, in the form we know it today, dates back as far as 1531, when East Indian Company stock was traded in Antwerp[10]. Modern financial markets such as the London Stock exchange (LSE), the New York Stock Exchange (NYSE), but also the numerous crypto-currency exchanges which have appeared suddenly in the last few years, all rely on the very same principles as back then. They allow participants (called "traders") to buy or sell a given amount of a security at a particular price. In the late 1990s, the regulatory authorities started to let traders access the markets using electronic communications networks (ECNs) and so a new era dawned [25]. Since then, high frequency trading (HFT) and sophisticated algorithmic trading vehicles have made up a substantial and ever-increasing part of electronic market participants. Their servers are oftentimes located within exchanges and specialized computer networks have been constructed to provide millisecond-level advantage in the arbitrage of trades between exchanges. Ever since, traders without such equipment and techniques have felt that they are at a disadvantage in such an environment. [25] While anything except trading through electronic channels would be unthinkable today, a certain gap still exists between trading companies that have fibre access to the exchanges or supporting algorithms and investors who do not. As a result, investors are forced to take an initial loss into account when buying or selling securities, which they might not even be aware of. In order to understand why these losses are incurred, we have to have 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 of the trading pair Bitcoin (BTC) versus US dollar (USD) taken on the Bitfinex[1] cryptocurrency exchange. The order book shows two columns, the parties who are willing to buy are on the left and the parties who are willing to sell are on the right. The two columns indicate the number of buyers and sellers (*count*) who are willing respectively to buy and sell a certain *amount* for a given *price*. The column *total* is the cumulative sum of the amount, or volume, on each side. The difference between the figures in each column is 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.

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

1. Buy 1.0 shares immediately for \$14,930.00 from a seller. To do so, we submit a *market* order.

2. State a price at which we are willing to buy 1.0 shares at price p, for example at \$14,910.00, and wait until someone is willing to sell at this price. To do so, 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 has a prior claim to them and that the seller does not cancel his/her listing. In this case, we are automatically willing to pay 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 accept their offers at 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 will match the offers 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 demanded price, and this will force us to buy the shares demanded at a later point in time. As the price of a share evolves over time, we might get lucky and be able to buy at 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 at 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 this brief understanding of how traders can interact with the exchange, we can define the problem of *order placement* as follows. Order placement determines the price at which a trader places its order. The aim of optimizing order placement are to minimize the opportunity cost and, ideally, achieve a more favorable price payable (or receive) than what is currently being offered at the market price. Literature therefore specifies a time scale of from ten to one hundred seconds within which a trader has to complete his task of either buying or selling the shares [16]. A time scale of less than ten seconds applies in high frequency trading and one of above 100 seconds is known as *order execution optimization*. Thus, could we define order placement optimization as the price p at which one should attempt to buy or sell i shares within a time horizon H of 100

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

seconds? As we shall see, optimizing placement is not as trivial as one might think, even though the concepts of the order book and the three order types a trader can choose from are admittedly simple. There are various properties in a limit order book, as well as the behaviour of the market participants, that changes over time. All of which can drastically interfere with the intention of buying and selling. Furthermore, since the foundation of electronic trading networks and algorithmic trading, the amount and sophistication of other market participants has been ever-increasing, with everyone aiming for an advantage over others.

The fact that reinforcement learning functions by maximizing rewards makes this technique unarguably suitable in this context. That is, 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. Ideally, such a learner will be able to foresee short-term trend changes such that the investor ultimately benefits from a better price at which to buy or sell the asset.

1.2. Research objectives

This work extends the findings of Kearns et. al. who have studied the behavior of order placement and order execution[23], and developed a reinforcement learning strategy[24] for the purposes of optimization. Their work explains how features derived from order book data were pre-processed and applied to a reinforcement learning algorithm which is similar to Q-Learning. In this thesis, rather than constructing features by hand, we will describe a particular instance of how deep reinforcement learning techniques were employed in order to benefit from patterns in raw market data. In addition, the cryptocurrency domain was chosen, instead of the traditional stock market. Furthermore, while the previously mentioned work of 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 successful. Hence, our ambition is to determine if deep reinforcement learning is perhaps an even more suitable choice in order to deal with unexpected market situations. Therefore we have formulated the following research question to be answered in this thesis:

RQ 1 (A): How can the application of deep reinforcement learning mitigate the impact of the order placement problem?
RQ 1 (B): Which cryptocurrency market events enable us to foresee opportunities to place limit orders at a favorable price with the use of deep reinforcement learning?

We chose to divide the research question into sub-questions as this follows the logical structure of this document and provides an understanding of the steps taken in order to give an answer to the main research question.

RQ 1.1: Which historical market data patterns drive market participants to buy or sell assets, and how can these patterns be incorporated into features used by a deep reinforcement learning agent?

Traders participating in financial markets, including cryptocurrency markets, can submit and cancel orders to trade shares. These events are recorded by the cryptocurrency exchange and are publicly accessible as raw market data. We intend to find patterns in the data which reflect the behavior of these market participants. The patterns found will form the basis of hypotheses suggesting that some behavioral patterns are more likely to lead traders either to buy or sell an asset in the short term. Whenever a hypothesis is true, one can determine a favorable price at which to buy or sell the asset, and hence, mitigate the impact of the order placement problem. Thus, we have to determine how to shape historical market data and construct features with the patterns found and incorporated that enable deep reinforcement learning agents to learn an order placement strategy.

RQ 1.2: How should one design a reinforcement learning environment and agents, in the context of order placement?

In order to simulate and understand the outcome of order placement and, more importantly, build a reinforcement learning environment that allows interaction with agents, this work will suggest a way of translating the given problem into a reinforcement learning context. Consequently, we are required to build a framework which should provide collection and market data processing capabilities in order to reproduce a historical order book that serves as a data source.

Further, we require that the framework provide the functionality of a match engine which emulates the functionality of a stock exchange that can match orders and determine the resulting price paid (or received) according to the historical order book. Ultimately, a reinforcement learning environment should be built to simulate and evaluate order placement. The environment should allow direct user interaction in order to place orders on demand and allow interaction with agents which act as intelligent traders. Therefore, we have made use of the OpenAI Gym[2] library and we contribute our work to the community to enable further investigations to be carried out.

Further, we shall then build two reinforcement learners which will both act as intelligent traders and thereby place limit orders that provide an incentive to buy or sell an asset at a favorable price. Both agents should be driven by an end-to-end learning process by which the agent improves based on the outcome of the orders placed and ultimately learns a strategy for buying and selling shares at favorable prices. The former agent is an adaptation of the well- known Q-Learning algorithm and optimizes only in accordance with the amount of assets to buy or sell and the given time horizon. This agent is a deep reinforcement learning agent that makes use of a convolutional neural network in order to detect patterns in market data.

RQ 1.3: How can one evaluate a reinforcement learning agent in the context of order placement?

The ability to quantify the capabilities of a limit order placement strategy learned by a reinforcement learning agent is of significant importance when answering the main research question in this thesis. Since there is no literature available which states results for the exact same data set, nor for any data set within the crypto- currency domain, a procedure has to be introduced by which different learning approaches and features considered can be evaluated. Therefore a measure is to be taken that is well-suited to the determination and comparison of the capabilities of a reinforcement learning agent in the order placement context. Furthermore, the evaluation procedure should identify the extent to which a limit order placement can be optimized in a given historical data set. As a result, the evaluation shall serve as a reliable measure of how well a learned order placement strategy performs.

RQ 1.4: In which way do the previously constructed features enable a reinforcement learning agent to improve the way it places orders?

The significance of deep reinforcement learning and the use of features derived from market data has to be determined in the application of order placement. The findings shall be compared to the performance achieved by a learner which has less information available, as well as the previously identified limitations of the market data available. Finally, the limitations of a learned strategy are to be identified.

1.3. Document structure

In Chapter 2, we first provide background information to the reader on the components of a stock exchange and the fundamentals of the closely related time series. Further, we make the reader familiar with (Deep) Reinforcement Learning. In Chapter 3, we elaborate on the behavior of order execution followed by approaches of both statistical and machine learning nature. Chapter 4 explains the process of data collection and preparation which was carried out prior to its use in the subsequent chapters. Chapter 5 explains the experimental setup of the reinforcement learning environment, the agents and the way processed features are used. In Chapter 6, we analyze the data, carry out execution placement, and include reasons for our findings. Finally, in Chapter 7, we formulate a conclusion of our findings and state a future research direction.

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

2
Preliminaries

In this chapter, we will provide background information in order to understand the previous work done in this field that was introduced in Chapter 3. We will also rely on the knowledge provided in this chapter when describing data collection and 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, although the interplay between the components we will introduce may not be immediately obvious, this will become clear in Chapter 5 when the components are used to build a reinforcement learning environment and agents. Firstly, the concept of the order book (which was introduced above) is described in greater detail, as this serves as the data structure for the historical data collected. Subsequently, a simplified match engine is described. We will use this to emulate a local broker that can match orders using the historical order book. The concept of a time series will then be introduced as the order book is essentially a multivariate time series. Furthermore, reinforcement learning is introduced in order to identify the differences between it and other machine learning techniques. This is 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 intentions to buy (or sell) a given asset, as described in Section 1.1). Orders listed in the limit order book provide *liquidity* to the market as other traders can accept these offers by posting an order with the equivalent price to sell (or buy) the asset. This section introduces the most popular order types under which traders can post their offers in a limit order book. We will identify the types that are better with respect to ensuring market liquidity and which therefore benefit from lower fees and those that enable traders to state their wish to immediately buy or sell assets and take liquidity from the market. Furthermore, the characteristics of a historical order book that is filled with orders from traders is explained as knowing them will assist when the match engine is explained in the subsequent 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 to buy or sell as the *Order Side*,

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

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

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

Limit order

A limit order refers to an 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 [8]. More precisely, with respect to buy orders, if the best price on the opposing side of the book equals or falls to lower than the limit price (or for sell orders, equals or exceeds it), the broker will match those two orders, resulting in a *trade*. The disadvantage of this order type is that there is no guarantee that the order will be executed. If no order appears on the opposing side, the order will remain (possibly forever) unexecuted.

Market order

A market order refers to an attempt to buy or sell a stock at the current market price, expressing the desire to buy (or sell) at 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. [9] The disadvantage is the less competitive the price one pays when the 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 be expensive, especially large ones.

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. To be precise, this is the *state* of an order book at some time t and shows the current limit orders from participants at this moment in time (ignoring the possibility that the state might have 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 historical sequence of order book states.

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

As we can see, every such state holds entries whose *price* or *amount* change, on both the buyer's and the seller's sides. We refer to each row that can be formed by participants who submitted limit orders of some amount at the same price level 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 the time stamp ts of the state,

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

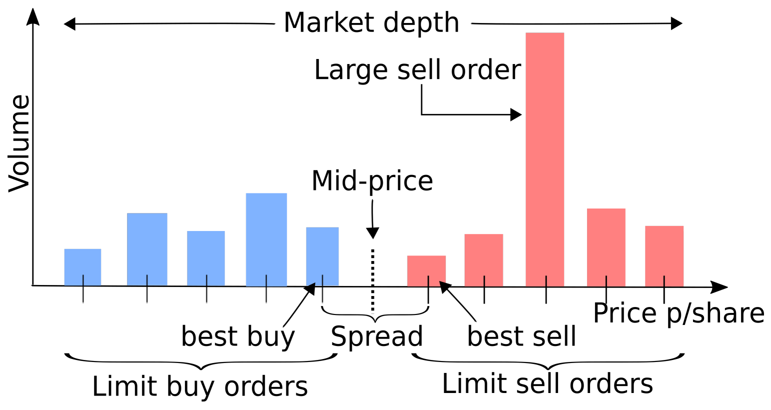


Figure 2.1: Figure taken from [5]. Simplified limit order book, which provides an understanding of some characteristics.

Figure 2.1 illustrates a simplified order book, from which we can derive definitions. The *limit level* specifies the position of an order book entry within the side of an order book state and the *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 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 at 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. By contrast, an *ask* refers to a price on the seller side and the *best ask* represents the lowest price at 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 of the best bid and best ask prices and the *spread* indicates the difference between the best bid and best ask.

The most recent price upon which a buyer and seller agreed 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 of shares provided by parties of the opposing side and is what effectively enables one to buy and sell securities. Liquidity is achieved by submitting limit orders which are not immediately executed. A *market maker* provides liquidity to the market by posting limit orders which are not immediately executed. In return, the market maker pays a lower fee than a market taker, the *maker fee*. By contrast, the *market taker* takes liquidity out of the market by posting either market orders or limit orders which are immediately executed by the exchange. As loss of liquidity is not beneficial to the exchange, the market taker pays a fee known as *taker fee* on a slightly higher scale.

2.2. Match Engine

The *matching engine* is the component which is responsible for the process of matching buy and sell orders at a traditional stock exchange such as *NASDAQ* or *NYSE*, or cryptocurrency exchanges such as *Bitfinex*, or *Bittrex*. In order to determine the outcome of an order, the trader typically submits the order to an exchange and either trades on the live market or gets access to a test environment, which if one exists, would be costly. Consequently, the order is processed on the current market and there is no option to process it on a historical data set in order to determine its hypothetical 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 evaluates the outcome of order placements using a historical order book data set, free of charge. This local match engine is a key element of the order placement optimization process as the outcome of matched orders will directly affect the reward received by an agent which, in turn, will use the reward to try to improve its own capabilities.

This section will first define a *trade* as the result of two matching orders. Subsequently, a time horizon - as an addition to the previously introduced order types (Section 2.1.1) - is presented so that we can describe the interface of the match engine that will be used throughout the learning process. Finally the rules relating to the implementation of the local match engine are outlined; these 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 on opposing order sides (Eq. 2.1) agree on a quantity of shares and its price. 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 elapses. *(Some implementations define this as Good Till Date, which involves specifying a validity expiry date and time for the order.)*

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

The match engine built in this project made available an interface that represents a function $match$ which takes any type of *Order* (Section 2.1.1) and 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* (which means "fulfilled") 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 differently depending on the submitted order type, and this is explained in the following paragraph.

2.2.3. Rules

Compared to the rules applicable to match engines used in electronic trading networks, the rules presented below are rather primitive. Yet they are sufficiently accurate within the subset of the limited capabilities provided to it, as compared to the capabilities of a real world exchange. The rules used by the order matching engine are mainly derived from [4]:

1. Limit orders (defined in Section 2.1.1) may be partially filled or not filled at all if there are no parties on the opposing side.
2. Market orders (defined in Section 2.1.1) will execute immediately if an opposite order has been previously submitted to the market.
3. Market orders may be partially filled, at different prices, depending on the liquidity on the opposing side of the book.
4. Attempts are made to match limit orders from a given point in time onward, or in the case of a Good Till Time (GTT), for as long as is 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 of what a conventional match engine, used by electronic trading networks, is capable of. That said, the following limitations have to be taken into consideration:

Participants: most importantly, 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 to simulate the past actions of market participants. While this is valuable real world data, unfortunately it does not cover the possibilities of hidden participants 1) entering or 2) leaving the market upon placing an order during the simulation. Participants who would enter the market would likely be favorable to us 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 any major advantage or disadvantage.

Ordering this match engine is restricted to simulating 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 are relying on historical data and the time stamps of the orders submitted from participants in the past is a limitation when submitting an order throughout 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 following order book state is nearer to the future than t would allow. We will therefore have to abort the matching process early.

2.3. Order execution and placement

From the above descriptions of the order book and the match engine, it is obvious that a trader has a variety of ways to approach a market and fulfill his duties to buy (or sell) shares. Conceptually, the process a trader follows involves these two steps: *order execution* and *order placement*; the latter is the main subject of this thesis.

Many useful definitions which highlight the difficulties related to the domain of order execution were stated by Lim et al. [18] and Guo et al. [16]. Most importantly, *order execution* concerns 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 its execution over an extended time horizon (typically on a daily or weekly basis), the impact cost can be lessened. By contrast, *order placement* concerns optimally placing orders within ten to hundred seconds. Placing refers to the setting of the limit level for a limit order as described in Section 2.1.1. Its aim is to minimize the *opportunity cost* which arises when the price moves against us.

Literature[16, 24] suggests using the *volume weighted average price (VWAP)* as measures 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. Reinforcement Learning

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

2.4.1. Advantages of end-to-end learning

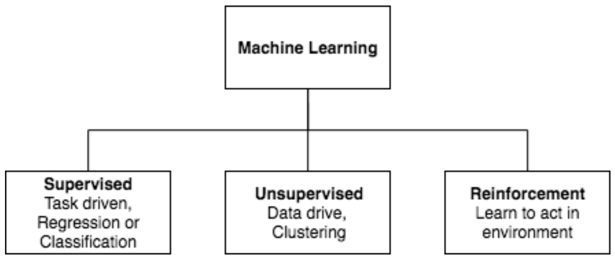


Figure 2.2: Categorization of machine learning techniques

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 future decisions and eventually an outcome, the result is that we learn more about the optimal sequence of decisions with reinforcement learning.

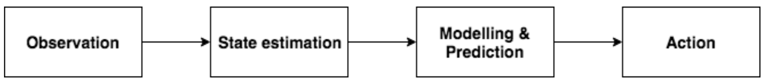


Figure 2.3: Reinforcement learning end-to-end learning pipeline

With respect to the optimization of order placement in limit order books, statistical approaches have long been the preferred choice. While statistics emphasizes inference from a process, machine learning 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 already-labeled data presenting a specific situation provided with the right action to do. From there, the algorithm tries to generalize the model.

In reinforcement learning, by contrast, there is no supervision and instead an agent learns by maximizing rewards. The feedback retrieved while executing a task that has 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 update 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 is executed, the agent receives a *reward* and will take this into consideration during future prediction phases. The beauty of this 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 report how well the task was executed. In our context, this means 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 nor identically distributed (I.I.D). The agent might in fact, while exploring, miss out on some important parts to learn the optimal behavior. Hence, time is crucial as the agent must explore as many parts of the environment as possible to be able to take the appropriate actions. [7]

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 agent's purpose is to observe features of the environment, 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 sent to the environment which determines whether this was a good or bad choice, for example, whether we made a profit or a loss.

2.4.2. Markov Decision Process (MDP)

A process such as the one outlined 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 future rewards.

2.4.3. Interaction

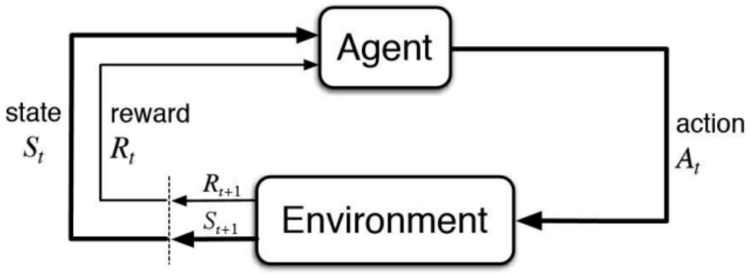


Figure 2.4: Figure taken from [7]: interaction between a reinforcement learning agent and the environment. An action is taken by the agent that results in some reward and a new state.

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

With the interfaces provided above (Section 2.4.2), 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* that 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 $\pi(s) : S \to A$ 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 agent's 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 in order to maximize 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 can define the future discounted *return* at time t as

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} * r_i \tag{2.11}$$

, 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 purposes: it prevents the total reward from going to infinity (since $0 \leq \gamma \leq 1$), and it enables the preferences of the agent for immediate rewards or potential future ones to be controlled. [6]

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 to be guided by:

$$V^{\pi}(s) = \mathbb{E}[R_t] = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \; \forall s \in S \tag{2.12}$$

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

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \; \forall s \in S \tag{2.13}$$

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

$$\pi^{*} = \arg\max_{\pi} V^{\pi}(s) \; \forall s \in S \tag{2.14}$$

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:

$$Q : S \times A \rightarrow \mathbb{R} \tag{2.15}$$

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,

$$Q^{*}(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{2.16}$$

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 their names suggest, easily obtained as

$$V^{*}(s) = \max_{a} Q^{*}(s, a) \; \forall s \in S \tag{2.17}$$

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

$$\pi^{*}(s) = \arg\max_{a} Q^{*}(s, a) \; \forall s \in S \tag{2.18}$$

2.4.4. Environment

There are two types of environments: In a *deterministic environment*, both the state 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. In a *stochastic environment*, there is 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 each time. *Deterministic environments are, in general, easier to solve as the agent learns to improve the policy without uncertainties in the MDP.*

2.4.5. Agent

The goal of the agent is to solve the MDP by finding the optimal policy, which means finding the sequence of actions that leads to receiving the maximum possible reward. However, there are various approaches to this, which are commonly categorized (see [6]) 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. 2.13). As shown in Eq. 2.12 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 it finds the optimal policy (Eq. 2.18). Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). As stated in Eq. 2.14, 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 behavior.

2.4.6. 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. 2.16) and the fact that this value is estimated for each state separately–which becomes totally impractical for large state spaces that can occur in real world scenarios. Deep reinforcement learning generally means approximating the value function, the policy, or the model of reinforcement learning via a neural network. As is preferred in reinforcement learning, neural networks approximate 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. 2.16). Therefore, we represent the action-value function with weights ω as,

$$Q(s,a;\omega) \approx Q^*(s,a) \tag{2.19}$$

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.

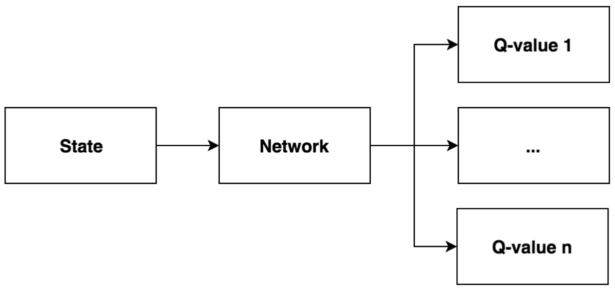


Figure 2.5: Neural network outputs q-values

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 [21], as illustrated in Figure 2.6:

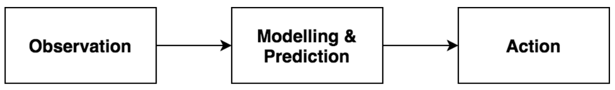


Figure 2.6: Deep Reinforcement learning end-to-end learning pipeline

3 Related Work

Compared to the execution problem, the literature for the order placement problem is sparse (as confirmed by Guo et al. [16]). In this Chapter, we will provide an overview of the work relied upon for the foundations of this project and for the insights they provided. We will first consider an empirical study of the general behavior of order placements, which serves as the conceptual basis for this project. Then, we will present a statistical approach which provides contrast to the subsequent overview of previous machine learning approaches. todo

3.1. Execution/Placement behaviour

Kearns et al. [23] determined which limit order price results in the most advantageous execution price. 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 resulting prices will identify the *risk* that comes 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 can be categorized between order execution and placement. No splitting of orders was performed, however a time horizon of several hours was chosen, resulting in an evaluation of order placement with an extended time horizon.

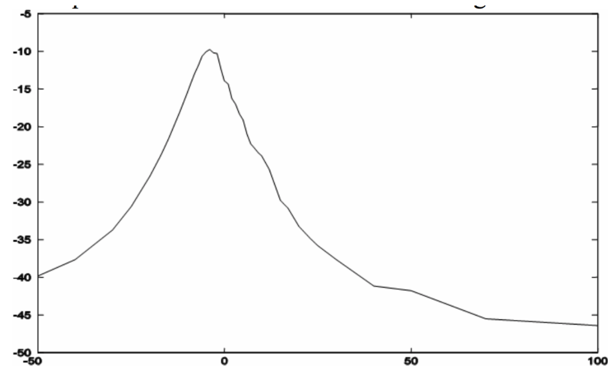


Figure 3.1: Taken from [23]. An illustration of the pricing strategy that produces the most favorable expected execution price.

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 ranging from -\$50 to +\$100. As is evident from the figure, the expected execution price is at its most favorable when setting the limit price close to the price of the spread, although only on the buyer side with a price of approximately \$10 lower than what is currently offered. The return becomes worse when placing orders deeper in the order book (in other words, offering a lower price) as the orders then do not get filled within an hour and instead, the inventory has to be bought by means of 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 (i.e. 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 filled instantly by paying a premium.

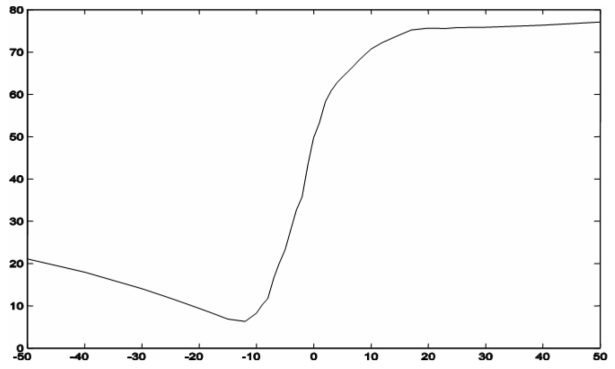


Figure 3.2: Taken from [23]. An illustration of the uncertainty of the expected execution price.

Risk is 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 illustrates the danger that arises from placing limit orders at less favourable limit levels. As has been already demonstrated, orders which are placed deep in either side of the book are less likely to be executed and their final prices are therefore necessarily less certain.

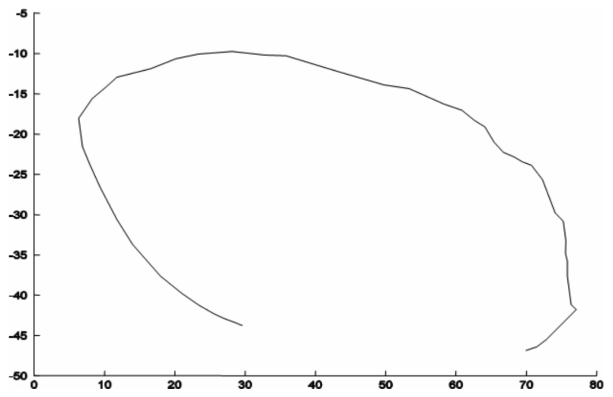


Figure 3.3: Taken from [23] An illustration of the trade-off between risk and return indicated by a efficient pricing frontier.

Lastly, both techniques were combined and this resulted in an efficient pricing frontier (based on the *efficient frontier* initially formulated by Harry Markowitz in 1952 [20]). 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 accept.

3.2. Statistical approach

Substantial work in a statistical context was carried out by Chaiyakorn Yingsaeree in his dissertation [29]. A framework was proposed for the making of 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 ($\implies$ risk) while placing orders at a certain limit level. This is 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, with the reinforcement learning approach presented in this work, we may be able to understand the limitations of market data to a greater extent and avoid the shortcomings of the former strategy.

We are providing here 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, which leaves the trader with the following options:

1. Do nothing.
2. Submit a market order at $t = 0$ at price p_0^M
3. Submit a market order at $t = T$ at price p_T^M
4. Submit limit order at 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 the event of an execution at price p, and a function $U_{NE}(p)$ defines the cost if the order is not executed at the end of the period at 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 by a) the probability that the limit order at price p^L will be executed before the end of the period and b) 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 defined as $V[U(p^L)]$, and this was 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 only about the profit, and when $\lambda = 1$ the trader is equally concerned about profit, risk, and missed opportunities. As a result, the trade-off between profit and risk was defined as,

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

3.3. Supervised Learning approach

Fletcher et al. [14] investigated order books with the aim of forecasting the movements of bid and ask prices at time $t + \Delta t$. Although this was not directly applied to the optimization of order placement, the resulting predictions 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 were used. The authors' approach 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, is defined 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. When a shorter time delta was chosen, this resulted in significantly better performance. However, this was mostly due to the fact that the zero movement prediction was accurate. An increased time delta resulted in significantly worse prediction accuracy.

3.4. Reinforcement Learning approach

A large-scale empirical application of reinforcement learning to optimize trade execution has been presented by Kearns et al. [24]. Although the title of their research suggests otherwise, their work is related to order placement with a larger time horizon H (2 minutes and 8 minutes), according to our definition in Section 2.3. Their research objective is accordingly defined as:

> to sell (or buy) V shares of a given stock within a fixed time period (or horizon) H, in a manner that maximizes the revenue received (or 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 considered three stocks, AMZN, NVDA, and QCOM; each with an inventory I of 5,000 shares. The relative improvement over a submit-and-leave strategy ranged from 27.16% to 35.50%. An additional improvement of 12.85% was achieved by considering the following market variables: Spread, Immediate Cost, and Signed Volume.

The architecture developed is as follows. *States* are represented by a vector $x \in X$ and correspond to an observation state that has the function of making a partially observable environment 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 $((ask + bid)/2)$. If the order is 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 that was developed by the authors, and which explores the state space inductively. Starting from $t = T...0$ the algorithm explores the inventories $i = 0...I$. At each step, all possible actions in this state are evaluated, leading to the most rewarding strategy for $t = 0$.

4 Market data curation and feature construction

In this chapter, we will outline the details of our data collection process, and how this data can subsequently form a historical order book in order to serve as the historical data source for the match engine. We will describe the way that raw market event data was collected from an exchange (Bittrex[1] in our case) and processed in order to form a historical limit order book. A sample period from the data set collected will then be investigated in order to find and visualize the properties of the market in question and the behavior of the corresponding 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 construction process which determines the input for the learner. Therefore, in the last section of this chapter, we will construct features which adequately address the stated hypotheses. Finally, the features constructed will serve as the observed state that is to be evaluated by the reinforcement learning agents described in Chapter 5.

4.1. Collection of market events

In most exchanges in the cryptocurrency domain, real-time market data is freely available. There is often a limit as to how far into the past historical data is retained. However, continuously recording real-time data results in a historical data set which is complete from the time when recording was started and has provided 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 of accumulating the data and building the order book is illustrated in a high level pipeline in Figure 4.1 below.

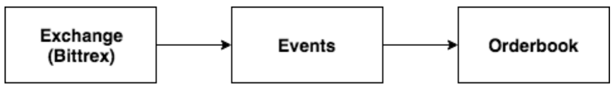


Figure 4.1: Data collection pipeline

Therefore, a complete order book is being reconstructed by processing market *events* that have occurred over time with 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 the event that a market order crosses the spread, resulting in a trade.

Our exchange of choice for collecting data was Bittrex as this 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, a status update is either a buy or sell order, or a fill (e.g. trade). Therefore, we subscribed to `https://socket.bittrex.com/signalr` and filtered the data field `M` for `updateExchangeState`. The

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

data type of the message contains the name of the trading pair and a nonce to identify the unique status update. That is,

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

, whereas $buy \in Order_{Limit}$, $sell \in Order_{Limit}$ (see Eq. 2.3) and $fill \in Trade$ (see Eq. 2.8). In this regard, the orders hold an additional field $type \in \{0, 1, 2\}$ which specifies whether it was a *create*, *cancel* or *change* in the order, whereby the changes of orders are neglected in our setup as this function is rarely used by traders.

It is evident that multiple events can be sent within one status update message. We segmented the status update into separate events with the same nonce, so that each event expressed either a limit order of a side bid or ask or a filled order resulting in a trade. We then defined an *event* as,

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

whereas $isTrade \in \{0, 1\}$ and $isBid \in \{0, 1\}$ indicating whether the update contains an order or a trade. Finally, we made use of the data types defined in Chapter 2 (Sections 2.1 and 2.2) such that $order \in Order_{Limit}$ and $trade \in Trade$.

4.2. Reconstruction of an order book with market events

The next step was to transform the events collected into an order book structure. By chronologically iterating over the processed events (Eq. 4.2), we created a new order book state (Eq. 2.7) for each such event. During this iterative process, we ensured that the correct order book entries remained in future order book states, by handling the events in accordance with their respective type, as follows:

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

Order cancelled: the amount of shares as specified in the canceled order is subtracted from the entry in the current state at the corresponding price level.

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

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

4.3. Formulating hypotheses of the market behaviour

Let us take a random, ~10 minute period of the recorded market event data, and try to extract essential information from either raw events or the order book that has been generated. Figure 4.2 shows the price movement of the chosen sample period, indicating a movement from \$10,100 to \$10,030 and back within 10 minutes. We first obtain an insight into the market situation with regard 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 will formulate *hypotheses* which propose why certain properties might be beneficial to the order placement process, and thereby suggest which properties might be worth considering in the feature construction process. However, we acknowledge that the observations gleaned are based on a random sample of a historical data set, By no means does this method guarantees that the same observations are true for any order book.

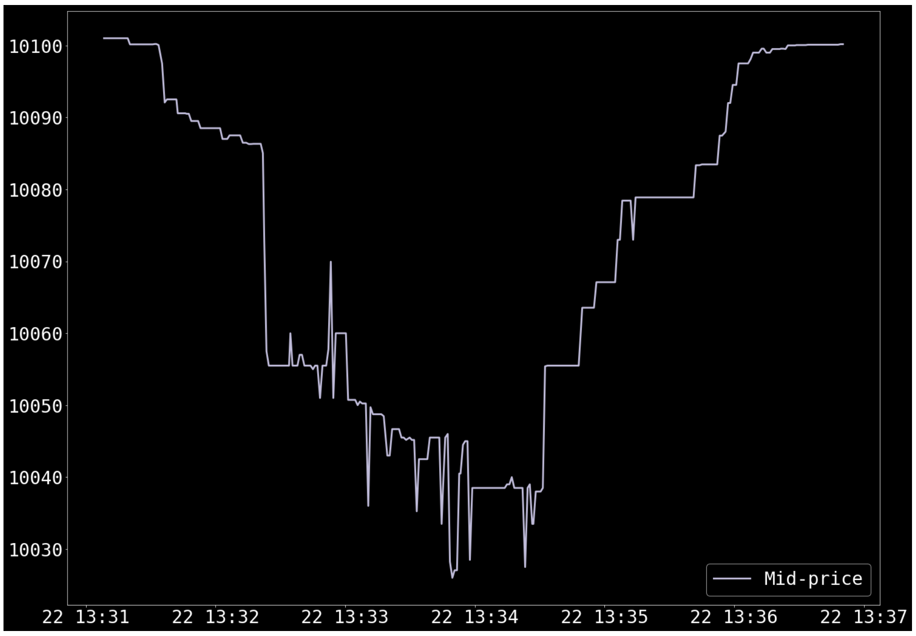
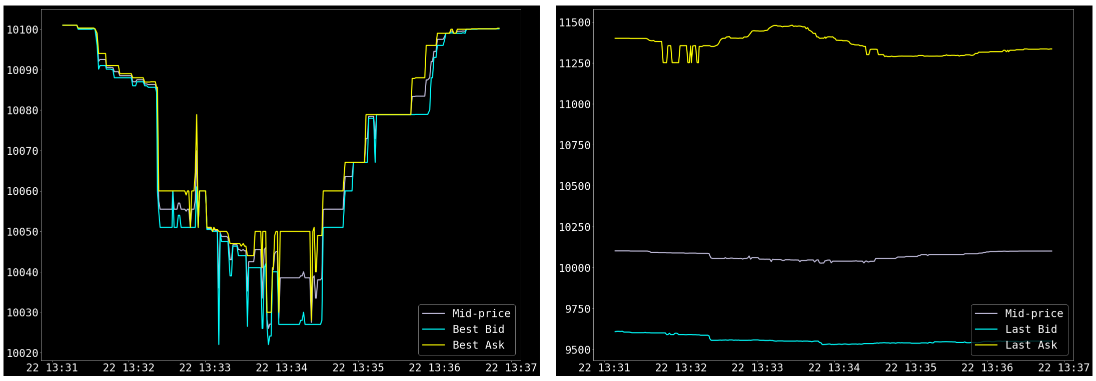


Figure 4.2: Price movement of sample data set

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

In Figure 4.3a we show the same price movement including the best bid and ask price. Furthermore, the deepest level of the bid and ask ask side is shown in Figure 4.3b. It is evident that the best bid and ask are close to the market 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 place limit orders close to the spread when the market price is stable and place them further from the spread when the market price is fluctuating.*

The orders placed at 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. On the contrary, bid prices remain much more stable. Likewise, during the fall and rise of the price, the ask price starts to increase at the deepest level of the seller side. This phenomenon is, at first glance, unexpected as one would expect the sell offers to decline as the price falls. As it can be seen that the price rises shortly after the dip, we can postulate that some sellers' intentions were to incentivize buyers with the fear that sell listings could rise even further

Hypothesis: *sellers incentivize buyers by placing higher prices on limit orders during a price fall.*

4.3.2. Importance of order volume

It was shown that market participants position orders at different price levels as the asset price moves 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 price movements in the given sample period.

	All events	**Trades only**	**Orders created**	**Orders canceled**
Bids	51%	41%	54%	57%
Asks	49%	59%	46%	43%

Table 4.1: Bid / Ask volume imbalance

Table 4.1 shows the balance between the volumes of bid and ask orders. It does this by categorizing the events as follows: all events, trade events only, order creations, and order cancellations. It is evident that, even though the price moved significantly within the recorded time range, all the events and trade events only (the first two categories defined above) are well balanced between the bid and ask side. Further, it is evident that the market participants reacted to the sale of the asset by not only creating but also canceling more buy orders than sell orders. This indicates that the market participants may have responded to the price dip by canceling their current buy orders and posting them at a lower price in the book. Interestingly, even though the price rose after the dip, the volumes of creations and cancellations of orders on the seller side were lower.

Hypothesis: *the balance (or imbalance) between volumes of bids and asks of all event types allows us to estimate the future behavior of market participants.*

4.3.3. Importance of volume of orders and trades over time

So far, volume has been investigated as a sum of events over time. In order to understand the behavior of participants in greater detail, a *volume map* will provide an insight into single events occurring over time, as shown in the following figures. The x-axis represents the time stamp and the y-axis is the volume of orders that were placed or resulted in trades. For visibility reasons, the y-axis follows a log scale, since the volumes of the majority of orders and trades are small and fewer have large volumes. Participants cannot be assigned to such orders or trades, as the trader "ID" is non-public information. However, as we will see, one can identify some participants on the basis of their behavior. Therefore, some of the more obvious patterns are highlighted in the figures.

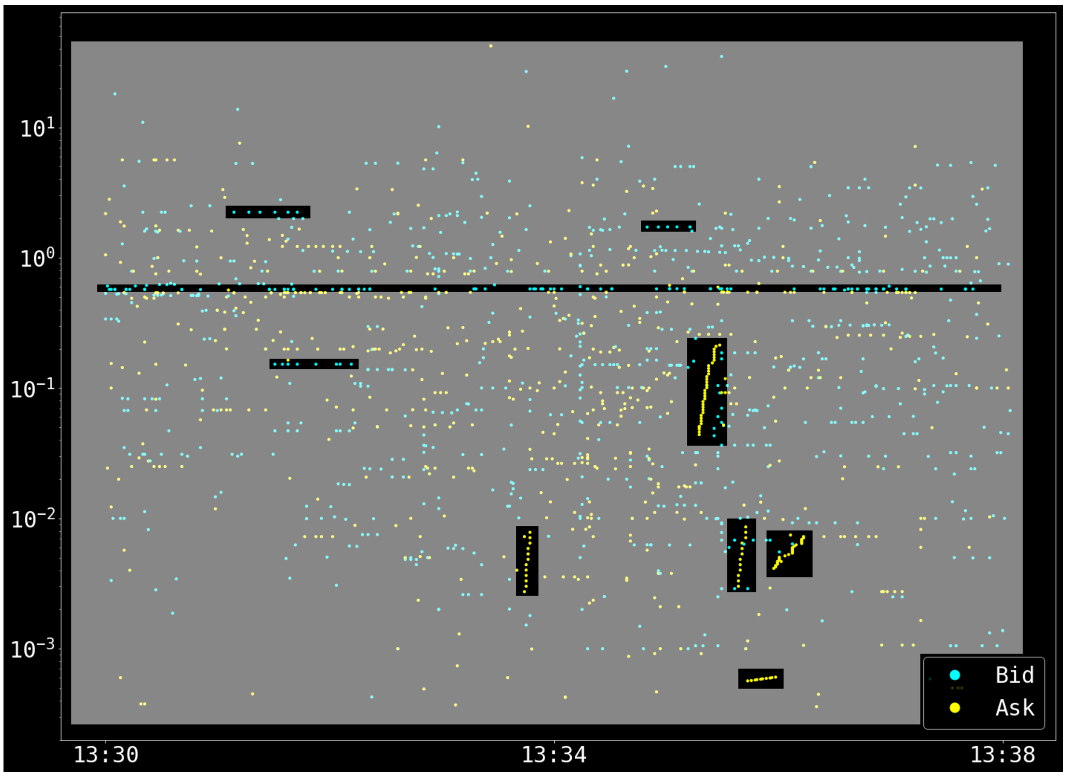


Figure 4.4: Volume map of created bid and ask orders.

4.2).

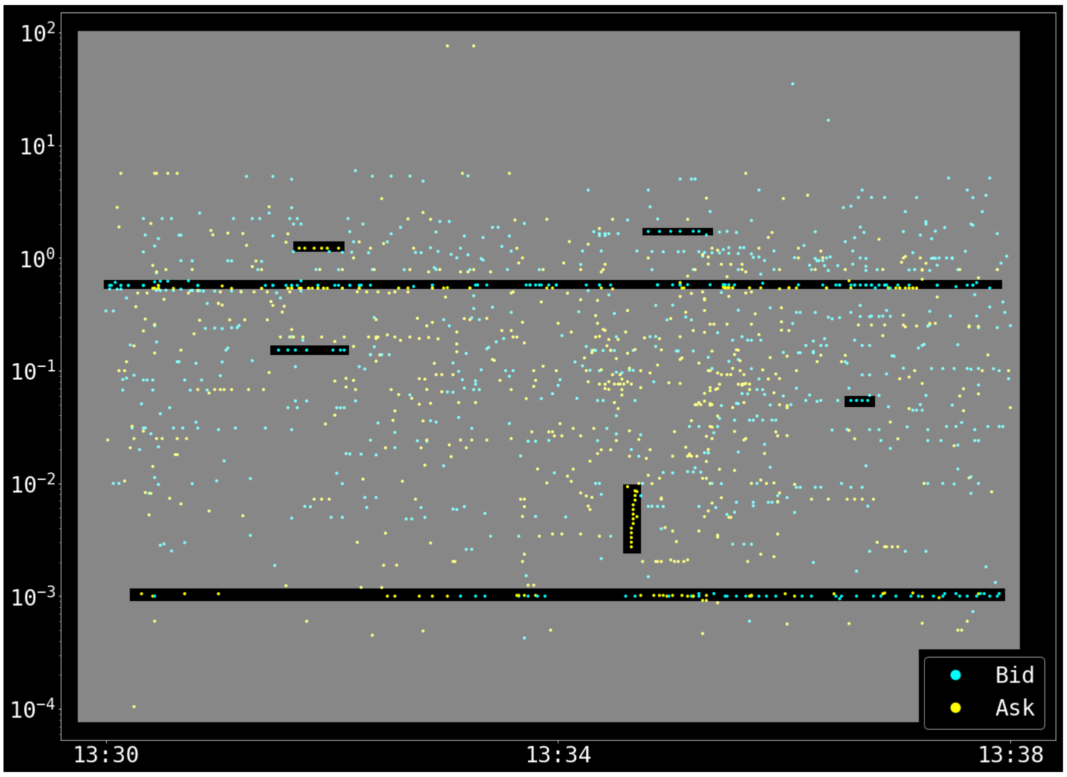


Figure 4.5: Volume map of cancelled bid and ask orders.

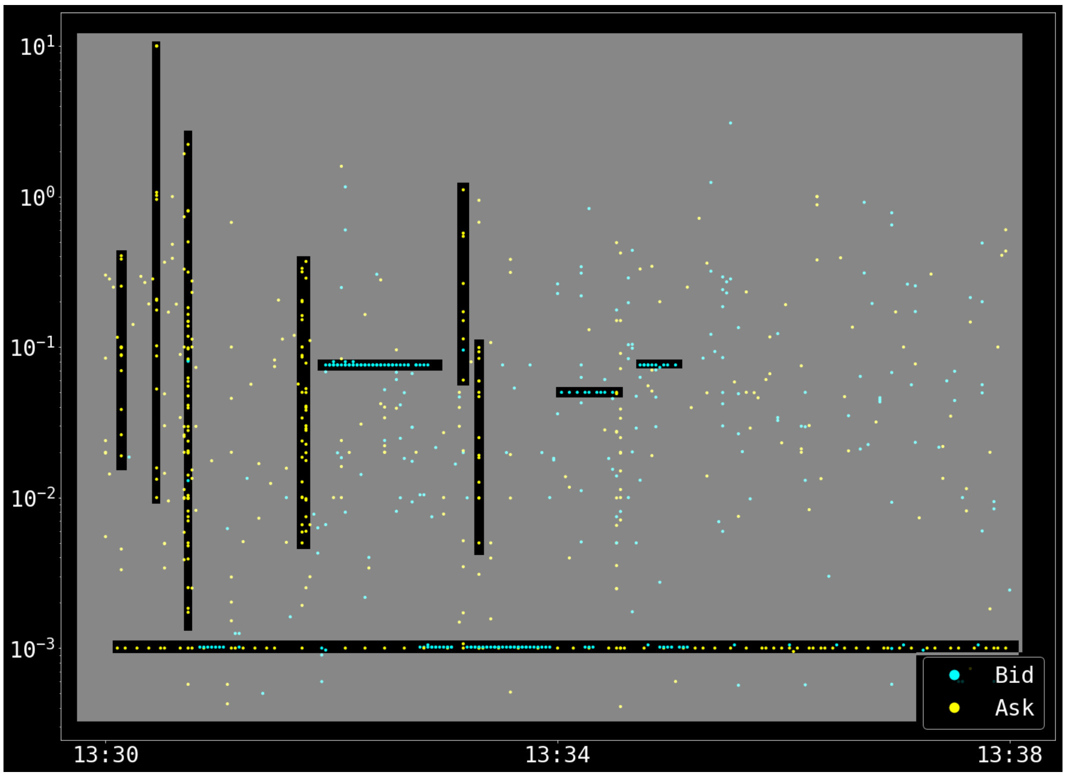


Figure 4.6: Volume map of trades initiated by a bid or ask order crossing the spread.

Figure 4.4 shows the volumes of the orders created. The volumes of most of the orders placed were less than 1.0 BTC and significantly fewer orders had a larger volume, indicating that most of the participants were either 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 on both sides, bids and asks, that were placed at the same time interval. This behavior is particularly evident at the regions highlighted on the volume axis just below 10^0. This is likely to be one or multiple bots posting orders of the same volume and perhaps at different price levels. Furthermore, a very distinctive diagonal-shaped pattern occurs when a trader posts orders of varying volumes within a short period of time. This might be evidence of someone splitting a large order into small pieces (known as a buy- or sell wall). Surprisingly, this behavior most often appeared both while and immediately after the market price started rising again. To illustrate this, the time stamps from Figure ? can be compared.

add reference

For at least some of the limit orders that did not result in a trade, the cancellation that followed was expected. Figure 4.5 shows the canceled orders over time. Sequences of cancellations emerged more clearly when volumes were just below 10^0 and at 10^{-3}, and when these volumes and their respective time intervals correlated with the sequences of the orders created, as shown in ?.

refernce

Hence, it is likely that there was a trader following a strategy which involved creating and canceling orders in equal volumes. A cancellation of one of the buy-walls that were created is particularly evident in both time stamps shortly after 13:34 and has a volume approximately equal to the one previously discovered during the observation of orders created. That makes it likely that this wall was created and canceled by a single trader, and perhaps created again as the same pattern occurred again in the order-creation figure at a later time stamp.

So far, only posted limit orders or their cancellations were observed. Not all of those limit orders might have resulted in a trade. Figure 4.6 illustrates the volume map relating to the actual trades that occurred. It is evident that the volumes of the trades transacted were 10^{-3}, and oftentimes they had identical time intervals. Additionally, a rapid series of many consecutive sales occurred around a time that correlates with the fall of the market price. After the price fell, such sales were not present anymore. Let us remember that there was one spike during the dip, and the time stamp related to it correlates strongly with the purchases (bids) visible in this figure before 13:34 with volumes of 10^{-1}.

Various behavioral patterns were 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 visually. However, an attempt to find a correlation between the behavior of events and the resulting trades by means of learning techniques seems promising.

Hypothesis: *patterns arising from events in which there were variations in the volumes posted determine future short-term trading behavior 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 in respect of each event type in the previous subsections. Patterns were found and a hypothesis was proposed to the effect that the various volumes of orders are an indicator of future behavior of market participants. If true, the market price would eventually be influenced and this allows us to determine the optimal order placement. The next logical step is to investigate the sum of traded volume over time, in combination with the price at which the asset was traded.

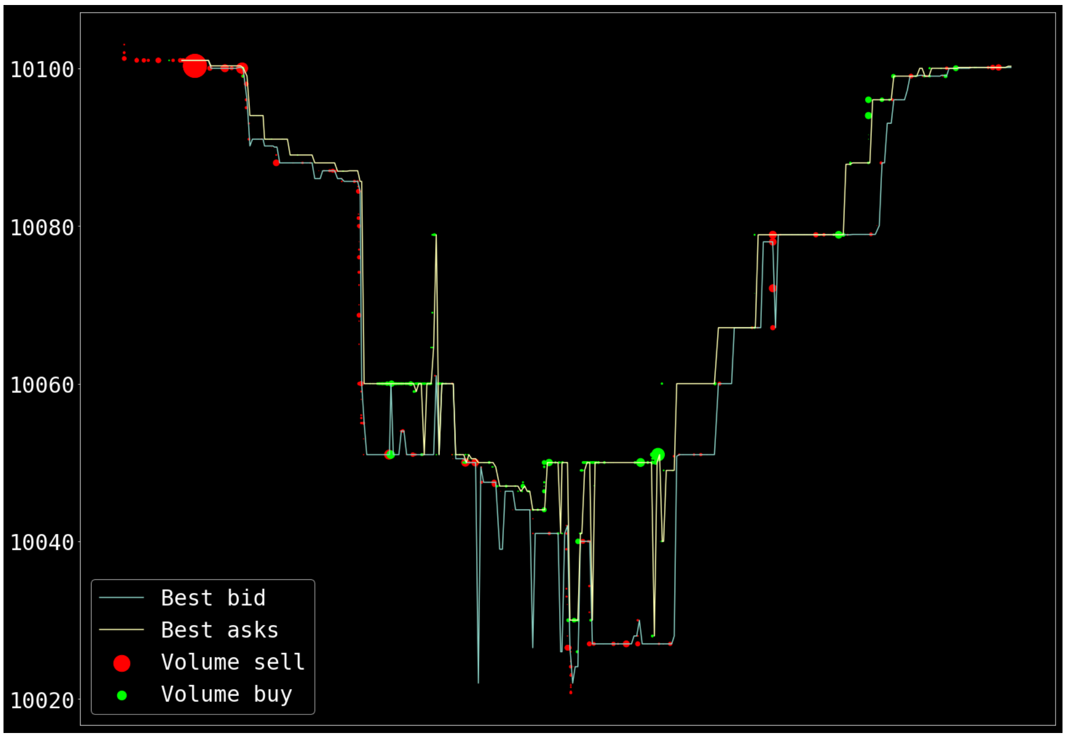


Figure 4.7: Relation of trade volume to price movement.

Figure 4.7 shows the volumes of trades on both bid and ask sides which resulted in a buy or sell at a certain price. The volumes of these trades are shown as bubbles of a size that indicates the traded volume and its position defines their price. As is known, a buy appears when one crosses the spread towards the seller side (ask) and a sell appears when crossing towards the buyer side (bid). One can clearly see how buys are listed at the best ask price and sells are listed at the best bid level. Before and during the dip, sells appear consecutively, followed by one large sell transaction. After that, a series of buy orders with low volumes occurs, which caused the minor spike. Interestingly, a rather large buy trade appeared shortly before the price started to rise again. Even though sellers confronted the market in the middle of the price rise, participants continued buying shortly thereafter. In conclusion, it is evident that a few trades with small volumes caused a certain noise in the overall trend. Multiple consecutive trades on one side or a single large trade, however, led the market price to move for a substantially longer period of time in one direction.

Hypothesis: *consecutive small trades 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 behaviors of market participants in an order-driven market, which ultimately determines the development of the limit order book. Hypotheses were laid out which posit that the outcome in terms of a change in the order book constellation and price development can be related to the aforementioned trading behavior. This suggests that orders can be placed and filled at limit levels which result in a favorable 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 those shown in [14, 17, 24], the aim of this project is to acquire knowledge directly from raw inputs, similar to a proven, successful method in the gaming sector[21] and which was recently applied in the trading context[19].

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 in which only the price bp (respectively ap) and size bs (respectively as) are considered.

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

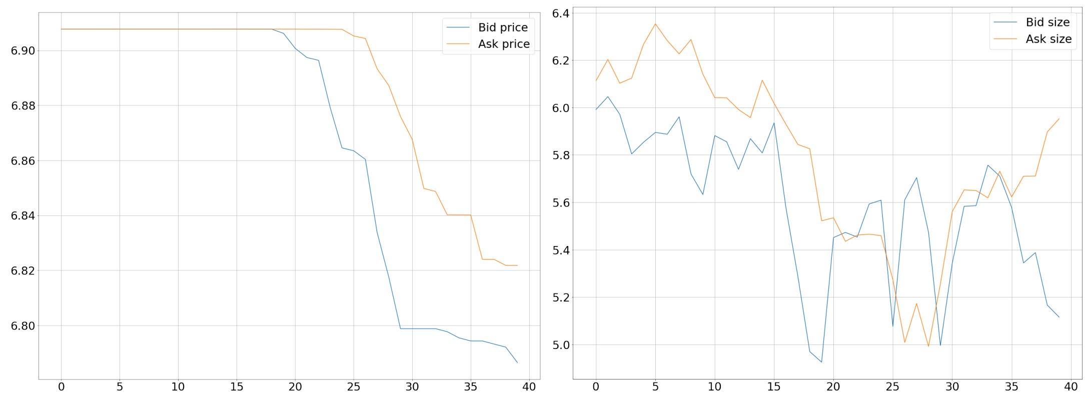
As the state will be observed by a deep learning agent that makes use of a neural network, the scaling of inputs will contribute to a faster learning process. We therefore also apply normalization to the prices (bp, ap) with respect to the best ask price for each state, that is ap_{i1}. Accordingly, we normalize the sizes (bs, as) by reference to the size provided at the best ask price 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 prices within a single order book state do not differ by 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.

There remain the questions of how large the window of m order books states should be and the number of limit levels that n should be chosen. The following observation provides an insight into the parameters; however, by no means does it aim to make an estimate of how well the agent may perform under the consideration of a certain parameter setup.

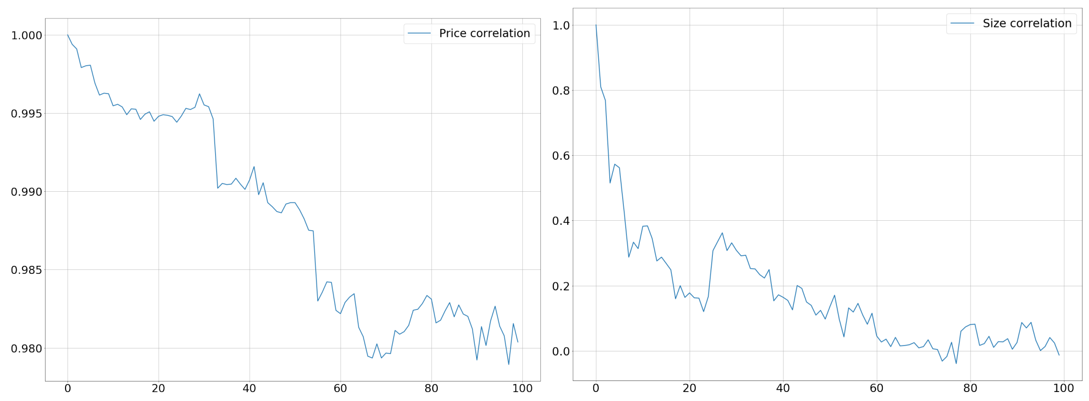
In order to determine the impact of n limit levels, we take the average of 100 evaluations whereas we take 1,000 random order book states for which we measure the Shannon entropy[26] for a range of 40 limit levels (maximum of what goes from collection) on the bid and ask side, applied to price and size. The entropy therefore serves as an indicator of how much information can be gained in respect of each limit level, derived from the change in price and size for each state.



(a) Entropy of order prices (b) Entropy of order sizes

Figure 4.8: Entropy measured for 40 limit levels

It is noticeable that the entropy of prices of limit levels 0-30 on both bid and ask sides remains high, as shown in Figure 4.8a. The price becomes slightly more constant for limit levels > 30. The entropy for order sizes, as shown in Figure 4.8b, drops after 20 limit levels, which indicates that the accumulated order size deep in the book is more constant. We therefore suggest considering at least 30 limit levels of the bid-ask feature.



(a) Correlation of order prices (b) Correlation of order sizes

Figure 4.9: Correlation measured for 100 order book states

With this brief understanding of how limit levels n affect order prices and sizes, we will try to make a statement about how order book states are related to the most recent state. More precisely, we will determine the correlation between the order prices and sizes from the previous m states and the most recent order book state. We will take an 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[26] to the state at t, whereas $n = 40$ (maximum). As can be seen in Figure 4.9a, 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 with the current state. The correlation of order sizes, as shown in Figure 4.9b drops more rapidly 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, in order to benefit from price differences within the feature.

4.4.2. Feature: price and size of historical trades

The previous feature provides information to the learner in order to reason about the hypotheses which are derived from orders placed and canceled. In order to reason about whether or not trade events can provide a positive learning effect to the agent 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, in this feature, we take n trades into consideration, which occurred prior the time of the order placement. More precisely, the feature is generated at some time t, when an order is placed, and therefore the time stamp ts of the historical trades 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 at fixed time intervals and this causes the length of the vector to vary. Accordingly, we calculate the time difference Δts between each historical trade and the subsequent historical trade. For the most recent historical trade, the time difference is measured to the time of the order placement t. That is,

$$\Delta ts_i = \begin{cases} t - ts_i & \text{if i = 1} \\ ts_{i+1} - ts_i & \text{otherwise} \end{cases} \tag{4.5}$$

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

$$s_{trade} = \begin{pmatrix} \Delta ts_1 & p_1 & q_1 & os_1 \\ \Delta ts_2 & p_2 & q_2 & os_2 \\ \vdots & \vdots & \vdots & \vdots \\ \Delta ts_n & p_n & q_n & os_n \end{pmatrix} \forall\ p, q, os, ts \in Trade \tag{4.6}$$

In addition, the new feature is normalized such that the prices are divided by the market price p_t and the quantities are divided by the size of the order q_t which is about to be placed at time t. As a result, we constructed a feature vector of length n (number of trades) that contains information about the price, order side and quantity of historical trades, as well as their order of occurrence.

4.5. Conclusion

Event data was collected from the Bittrex cryptocurrency exchange and a limit order book was reconstructed therefrom. This 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, derived from the order book generated, was shown and the underlying event data were investigated. Patterns were found which give insight into how market participants positioned their orders, with respect to price and size. It was shown that the price movement was likely to be due to (1) an imbalance between bid and ask orders; (2) a distinctive way of posting or canceling orders; and (3) consecutive or impulsive trades. These findings were incorporated within the two features constructed, and will serve as the observation state for the reinforcement learning agents that are described in the following chapter.

5 Experimental reinforcement learning setup

Details of the components and techniques required for optimizing order placement was provided in Chapter 2, and previous approaches pursued by other researchers 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 features were formed to be used within the reinforcement learning setup. The next step is to build a reinforcement learning environment which is flexible enough to enable a) investigations with various types of features and agents to proceed, and b) adjustments to be made to important environment parameters. 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.

Therefore, this chapter aims to build an environment that includes the desired 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. First, the setup of the environment is described, whereby we explain how the components involved work in combination such that a learner can simulate order placement. 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 the features previously developed.

5.1. Order Placement Environment

The reinforcement learning environment (see Section 2.4.4), that emulates order placement on historical market data, is introduced in this section. This environment enables an agent to buy or sell V shares within a time horizon H. Therefore, the components previously described in Chapter 2 come into play. It works principally by an agent observing a state s_t (observation state) at some time t and responding with an action a_t that indicates the price at which to place the order in the order book. The task of the environment is then to evaluate the outcome of the order placed and report to the agent along with a reward r_{t+1} and the next state s_{t+1}. Subsequently, the order is canceled so that the agent can submit a new action for the remaining shares to be bought or sold.

OpenAI Gym [13] 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 this is that any OpenAI Gym compatible agent and bench-marking tools can be applied in this environment.

5.1.1. Overview of components

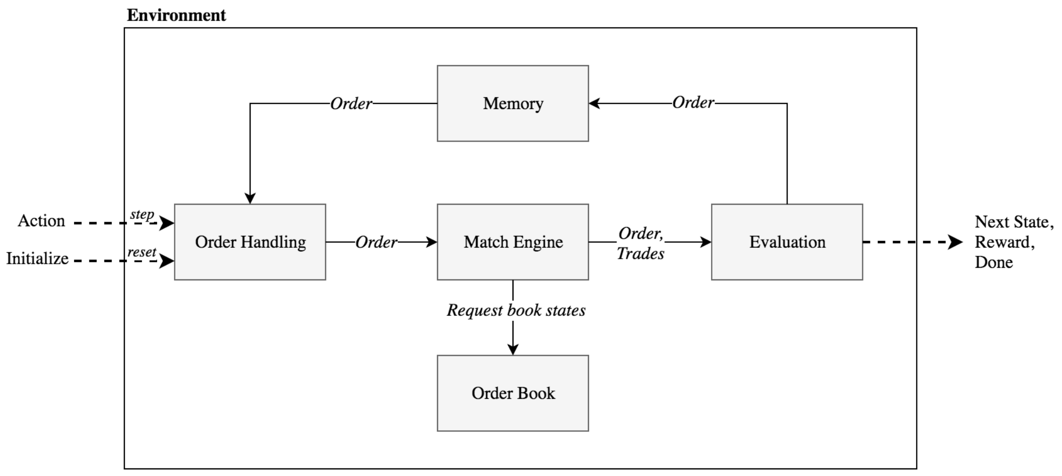


Figure 5.1: Overview of reinforcement learning order placement environment.

Figure 5.1 shows the inner workings of the order placement environment. An agent will simulate and complete the placement of one order of V shares with time horizon H within one *epoch*. More precisely, the agent initializes an epoch by using the `reset` function, which clears the internal state of the environment stored in its memory. The internal state consists of the remaining shares the agent has to buy or sell (as denoted by the *inventory i*), the time t that the agent has left to do so, and the ongoing *order* that is to be placed. In addition, a random point in time in the historical data set is chosen for the agent to proceed the initiated epoch (e.g. placement of the order). The agent explores the environment using the $step$ function which it uses to send the action to place the order at a certain price level. The first component of the environment to react to the an action when it is received is the *order handling component*. This recalls the order the agent is currently trying to fill and adjusts the price in response to the action received. Subsequently, the order is forwarded to the *match engine*, which attempts to execute the order within the historical order book. The order, as well as the possible resulting 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, this component is responsible for updating and storing the remaining inventory and the consumed time of the order in the *memory*. Additionally, the index of the last visited order book state is stored so that, in a consecutive step, the match engine will proceed the matching where it stopped last. In case no trades result during the matching process, only the consumed time is subtracted from the order. Otherwise, the sum of the sizes of the trades is subtracted from currently stored inventory in the memory. Subsequently, the evaluation component calculates the reward based on the resulted trades. In case the order is not completely filled after the last step taken by the agent, a market order is executed by the environment in order to get to the final state and therefore force the completion of the epoch. Finally, the reward, the next observation state and confirmation of whether or not the order was completely filled (e.g. the epoch is done) is finally forwarded to the agent.

5.1.2. Configuration parameters

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

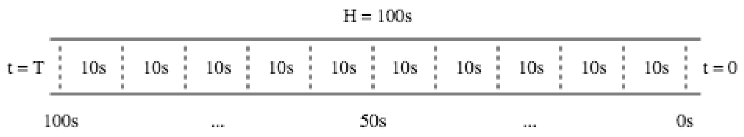


Figure 5.2: Segmented time horizon H with remaining time t.

The time horizon parameter H defines the amount of time given in order to fill an order. The default time horizon is set at 100 seconds, for the reasons described in Section 2.3, and this is the equivalent to the Good-Till-Time option of an order placed in a financial market (see 2.1.2). Furthermore, in this environment, we intend the agents to take discrete steps, rather than continuous steps, as the latter would not be computationally feasible within this work. The steps the agents will take are determined by the time horizon in which the agent has to completely fill the order, and therefore, the time horizon is segmented into discrete time steps t, as illustrated in Figure 5.2. As a result, the number of steps the agent can take are limited to the number of steps t. Each step is of the same length Δt which, for illustration purposes, has been 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 set to Δt for each step, until a total of total time of H is reached and the GTT is set to 0. Lastly, the *feature type FT* determines which state is to be observed by the agent. The feature time can either be formed by the private variables of the epoch only - inventory i and remaining time left t - or by a combination of the private variables and 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 for the movements of the figures in that step. The epoch would continue until the agent either wins or loses 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, in each step, not only one or two figures of the chess board change their position, but almost all of them. In addition, a reset of the environment would 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 which is in essence a multivariate, possibly non-stationary, time series which changes 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_t of the historical order book provided, at some point in time in the past. Therefore, the observation state is the result of the order book state applied to the feature type in use: $O_t = FT(OS_t)$. The final state is reached when the entire inventory is 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* [24]. 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. In our case, the specified feature type (constructed in Section 4.4) 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 reinforcement learning agent will state at which price the place the order. We define a fixed set of actions that an agent can take and that will correspond to a price level which is relative to the current market price of the state of the historical order book. Therefore, a discrete action space A is a vector $(a_{min}, ..., a_{max})$ that represents a range of relative limit levels that 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 on 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 a discrete step size chosen for the actions. An illustration of this concept is given in Figure 5.3.

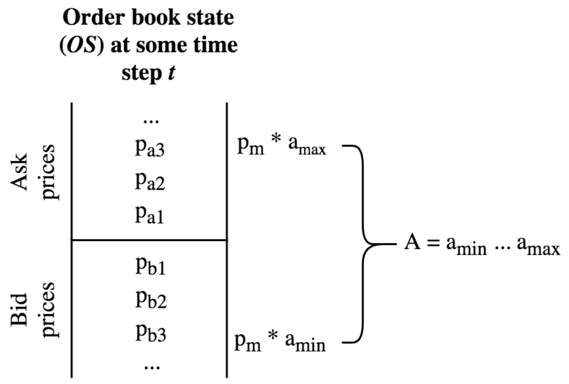


Figure 5.3: Actions represent an offset relative to the order price at which to place the order in some order book state OS_i at some time step t.

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 of $p = p_{m^T} - \$5$ 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 between 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, wih respect to 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 result during the matching process, the reward is $r = 0$, indicating that no direct negative reward is provided. The reasons for this are that in case the order could not be matched over the course of the given time horizon, when $t = 0$, a market order follows which might produce trades at a price worse than the market price before the placement started. In that case, a negative reward is ultimately given. As a result, we define the discounted return (Eq. 2.11) as $R_t = \sum_{t'=t}^{t_0} \gamma^{t'-t} * r_{t'}$, whereas t_0 is the time step at which the agent has its time horizon fully consumed for the order of the current epoch.

Disclaimer: Since we are interested in the general ability of reinforcement learning to learn how to place orders, potential maker or taker fees are not considered in this setup.

5.2. Q-Learning agent

The agent described in this section is generally known as *Q-Learning*[28]. 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 5.3 below), which is an extension of the Q-Learning agent. The name "Q-Learning" refers to the application of the Q-function previously presented (Eq. 2.15). More specifically, it relies on the *action-value function* (Eq. 2.16) 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 the value of future rewards, 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 $Q_i \to Q^*$ as $i \to \infty$. [27]

Q-Learning makes use of the aforementioned Bellman equation (Eq. 5.1), which undergoes an iterative update. The algorithm has proven to be an efficient and effective choice for solving problems in a discrete state space. The limitations of this approach emerge when the agent is applied to large or continuous state spaces

[15]. They become more apparent when considering the algorithm presented above. The iterative update of the action-value function $Q(s,a)$ (defined in Eq. 2.16 and used in Eq. 5.1) is exposed to the size of state s and action a, and thus if s is too large, the optimal policy $\pi^*(s)$ (defined in Eq. 2.18) is not likely to 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, $s = (i, t)$.

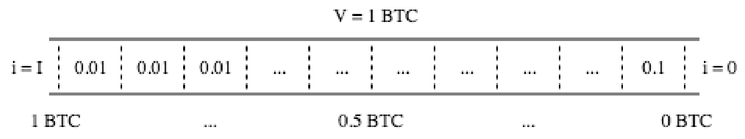


Figure 5.4: Inventory of V segmented shares with a remaining inventory i.

The resulting change in the fractions of the remaining inventory that occurs during matching process would, however, still result in a vast state space. Therefore, as for the discrete time steps described in Section 5.1.2, 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 an order is updated. We pick I as the maximum value for 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

```
1: Initialize $Q(s,a)$ arbitrarily
2: for each episode do
3:     for t=0...T do
4:         for i=0...I do
5:             $s = (i, t)$
6:             Choose $a$ from $s$ using $\pi$ derived from $Q$ ($\epsilon$-greedy)
7:             Take action $a$, observe $r, s'$
8:             $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
9:             $s \leftarrow s'$
```

Finally, algorithm 1 describes the Q-Learning algorithm used in this work. An adaptation was made to a conventional Q-Learning algorithm[27] regarding the order of the steps the agents will follow; this adaptation therefore follows the same concept as presented in [24]. The agent solves the order placement problem inductively, starting with the episode at state $s = (i, 0)$, when $t = 0$. This has the benefit that the environment forces 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 existing 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 specifies to which extent new information should override previously learned information, with weighting $0 \leq \alpha \leq 1$. Eventually, the agent completes the episode when every combination of t and i has been visited by the agent, that is in state (I, T). Hence, the agent has learned each discrete step i and t in the process of buying or selling V within the time horizon H.

5.3. Deep Q-Network agent

The second agent presented in this section is known as Deep Q-Network (mnih2015human) [22]. 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

As for 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) with stochastic gradient descent,

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

While 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* to build a data set D from the agent's own experiences. Accordingly, 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 the learning process, Q-learning updates on samples (or mini-batch) of these experiences $(s, a, r, s') \sim U(D)$, 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 occur during a significant rise or fall in the market price. In addition, experience replay stores rare experiences for much longer so that the agent can learn them more often. That is, for example, when massive subsequent buy orders 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 remain unchanged between individual updates.

6 Order placement evaluation and discussion of results

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 make use of this setup and to run simulations and thereby observe whether or not reinforcement learning is indeed capable of optimizing the placement of limit orders. Therefore, a comprehensive evaluation procedure is being introduced that allows to measure the capabilities of the reinforcement learning agents. Throughout which use of real world historical order books is being made, 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 outline the steps of the evaluation procedure. Subsequently, the real world data sets chosen and their use within the reinforcement learning setup are described. Finally, we will proceed the evaluation steps in the outlined order. As a result, this chapter quantifies how effective deep reinforcement learning and the use of the previously constructed market features.

6.1. Explanation of the evaluation procedure

This section explains the procedure undergone in the following sections of this chapter, with which we aim to make a statement about the capabilities of optimizing limit order placement with reinforcement learning and the use of raw market data. The following points list the steps of the evaluation to be done in chronological order.

Empirical investigation: Section 6.3 investigates the reinforcement learning environment empirically by simulating an agents behaviour that places buy and sell orders for a range of limit levels. This will provide knowledge about the limitations of the potential optimization possibilities within the given data set and how well we should expect the reinforcement learners to perform.
Results: the *estimated returns* to be received for (1) the optimally chosen limit order or (2) an immediate purchase or sale by using a market order.

Q-Learning agent strategy: In Section 6.4, we make an attempt to build an order placement strategy based on private variables only, by using the Q-Learner. This will provide insights of the performance of a naive reinforcement learner and serves as a benchmark for the following simulations proceeded in which we consider market variables.
Results: the *average reward* achieved by the Q-Learning agent that uses private variables.

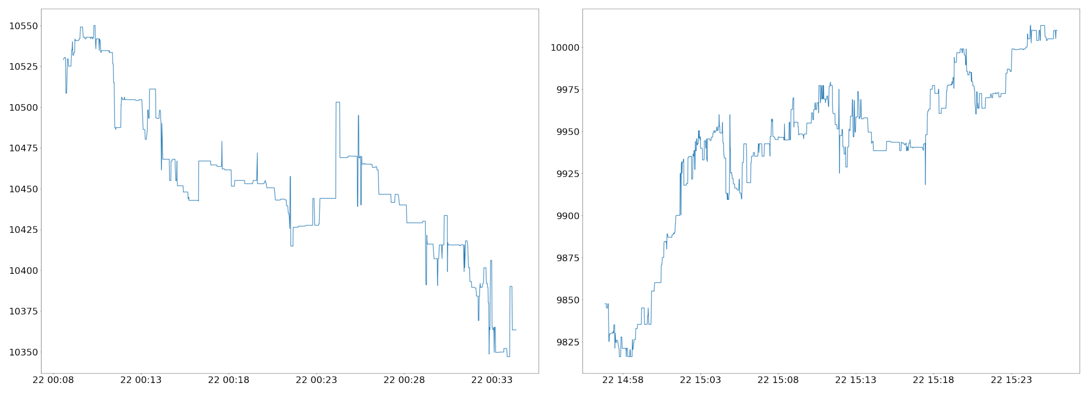
DQN agent strategy: Section 6.5 applies market variables to the DQN agent. Hereby, we make use of Feature I: price and size of historical orders as described in Section 4.4.1; as well as Feature II: the price and size of historical trades as described in Section 4.4.2. Similar to the previous evaluation step, we will find the average rewards produced by the agent.
Results: the *average reward* achieved by the DQN agent with the use of private variables and (1) historical orders or (2) historical trades.

DQN agent limitations: Section 6.6 aims to determine the capabilities and limitations of the DQN agent in greater detail. Therefore, we investigate the chosen actions selected by the agent in order to determine the agents limitations. In addition, the agent is applied to an environment which is equipped with an artificially generated order book and will allow to determine to which extent the agent is able to learn from certain price trends.
Results: (1) *insights* into when then DQN agent does not perform well and (2) the *average reward* achieved on order books that follow an artificially created trend (upwards, downwards and sine curve).

Given the found results throughout this evaluation, we will be able to determine and quantify to which extent deep reinforcement learning can optimize limit order placement and give reasons for its limitations.

6.2. Data sets and their usage in the reinforcement learning setup

We have selected two ~30 minute samples of historical order book recordings with which we proceed experiments in this chapter. Reasons for choosing two very distinguishable data sets include to determine the ability of the learners to react on a variety of market situations. And, 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. More precisely, *data set I*, as shown in Figure 6.1a), is a downwards trend (indicated by the bid/ask mid-price) and consists of 1132 order book states with a duration of 1681.8 seconds, resulting in 0.67 states per second. Contrarily, *data set II*, as shown in Figure 6.1b, consists of 1469 order book states with a duration of 1746.0 seconds, resulting in 0.84 states per second, which indicates that there was slightly more pressure in terms of orders placed and cancelled in this data set. 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.



(a) 30 minute downwards trend (b) 30 minute upwards trend

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

As is explained in Chapter 5, the historical data sets are not maintained by the reinforcement learning agents directly but instead by the reinforcement learning environment. The environment provides an observation state O, derived from the data set, to an agent, after which the agent decides to take an action a in form of a limit level. In turn, the environment prices the order at the received price level and returns the evaluated reward r and the next observation state O to the agent. This way, the agent can simulate the placement of limit orders, such that, within the given time horizon, the demanded inventory can be either bought or sold. For each *epoch* an agent proceeds, one order, with a specified inventory and time horizon, is defined and is to be filled. Therefore, the reinforcement learning environment selects, for each epoch the agent initiates, a range of order book states which form the given time horizon H within which the agent is supposed to complete an order.

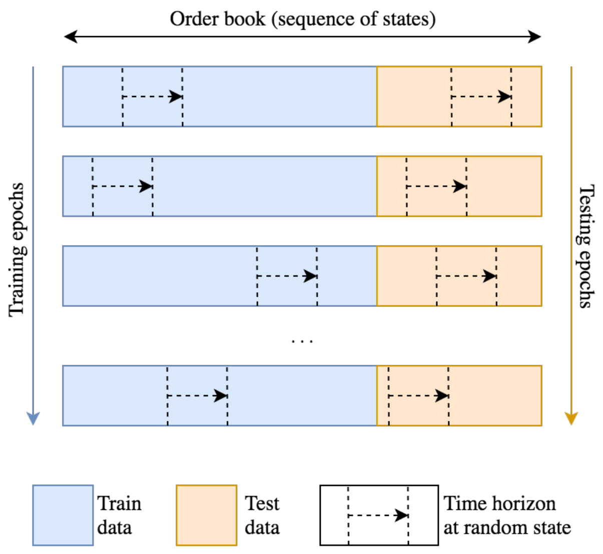


Figure 6.2: Order placement training and testing on an order book data set.

Figure 6.2 illustrates this process. A randomly chosen order book state defines the beginning of the time horizon and the set of order book states that fall into this window. This is very crucial since the states within this time horizon and set of states, not only lead to the observation states received by the agent, but also will determine the outcome of the matching process. More precisely, for each step the agent takes, a consecutive sequence of order book states (with a total difference of their time stamps of Δt) will be considered by the match engine, as explained in the previous chapter in Section 5.1.2. This process is identical for training and testing, except that the underlying data is different and the agent will not learn from the epochs proceeded during testing and instead will report the achieved rewards.

6.3. An empirical investigation of the reinforcement learning environment

This section serves to investigate the relationship between the limit order placement and the received return. 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). Therefore we simulate an agent that buys and sells shares at every possible limit level and records the received 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 or received respectively, as stated in Eq. 5.1.5. As a result, we gain understanding about estimated rewards for limit order placement using the given historical data set. In addition, these results set a benchmark for the reinforcement learners to come.

The setup of this investigation is as follows. We investigate the rewards of limit orders placed on progressive increasing time horizons, starting from 10 seconds up to 100 seconds, and therefore demonstrate by hand how a reinforcement learning agent would take discrete time steps. For each time horizon, we place (e.g. cross-validating) 100 orders of size 1.0 BTC at the beginning of the time horizon whose beginning is defined by a randomly chosen order book state. A market order follows for the remainder of shares (if any) once the time horizon is consumed. The expected return is then formed by the average of the received returns of these 100 orders. This process is repeated for a range of 201 limit levels ($-100...100$) with step size \$0.10, resulting in orders priced in the range of $p_m - 10 \dots p_m + 10$, whereas p_m is the market price before the order was placed. The limit levels are chosen broadly in order to retrieve understanding about the outcome of a variety of possible actions. Hence, a total 20'100 orders are submitted for each defined time horizon. Finally, the investigation is proceeded for both data sets I and II.

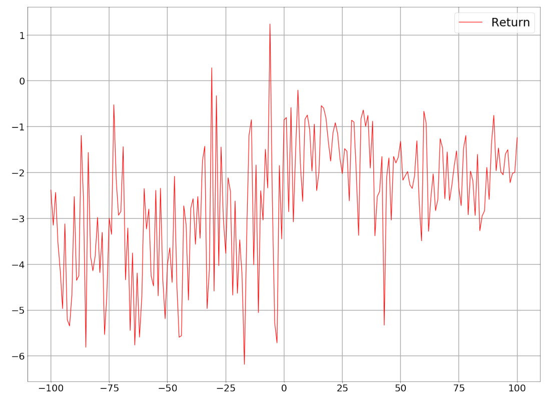
6.3.1. Order placement behaviour on data set I

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. This investigation is shown in Figure 6.3 for time horizons of 10, 30, 60 and 100 seconds respectively. The x-axis indicates the placement of the order at limit levels reaching from -100 to +100 and the y-axis indicates the average received return.

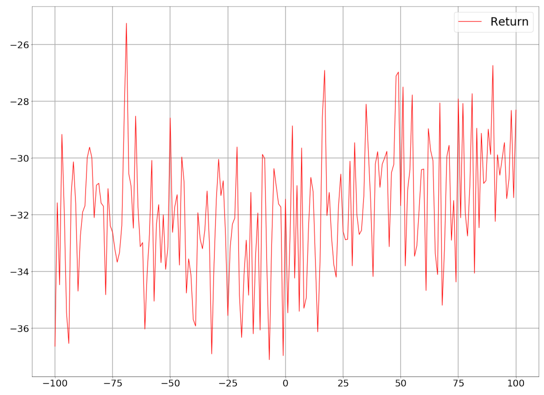
With a time horizon of only 10 seconds left, the expected behaviour is, however, proven wrong. For buy orders, shown in Figure 6.3a, the returns suggest to place the orders close to the spread, but still on the opposing side, at a limit level of ~+5. The spike at limit level ~-5 indicates that the overall 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 6.4b, the best return is given when crossing the spread with a positive limit level of ~+50.

With an increased time horizon of a total of 30 seconds, as shown in Figures 6.3c and 6.3d, the expected behaviour becomes more apparent. Positive returns can be achieved by posting buy orders deep in the order book. Therefore, we can expect that in the given market situation an agent would be able to execute the order partially at very low limit levels and for the unexecuted part a market order would follow. The most dense range of positive returns seen around the limit levels just below the spread. Orders placed deeper in the book result oftentimes in slightly lower returns, which indicates that the orders were only filled partially and expensive market orders followed. Crossing the spread causes increasingly lower returns, the more positive the limit level is chosen, as a result of agents willingness to immediately buy at an increasing price. 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.

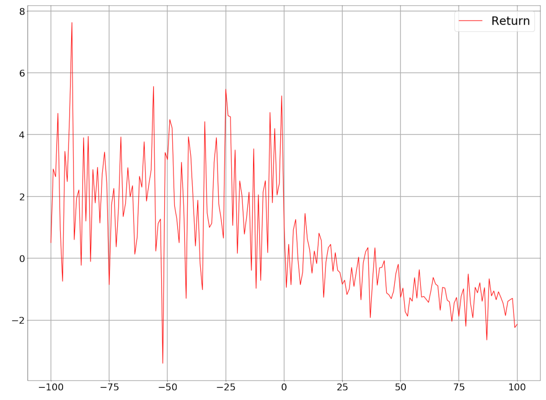
With time horizons of 60 and 100 seconds the expected behaviour of the orders is clearly apparent. Buy orders, as shown in Figures 6.3e and 6.3g, achieve most return when placed very deep in the order book. However, when placed too deep, at level -100, the return is slightly less as a result of unexecuted orders which had to be completed with market orders. 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. Furthermore, 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.



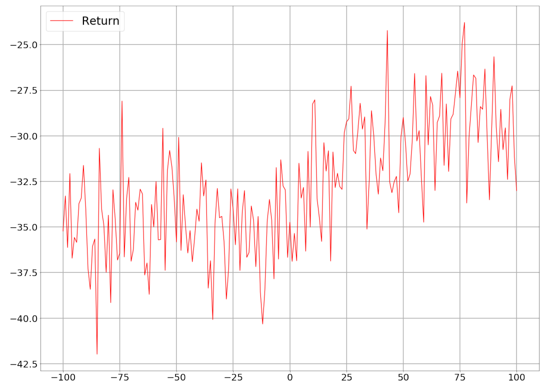
(a) Returns of buy orders within 10 seconds



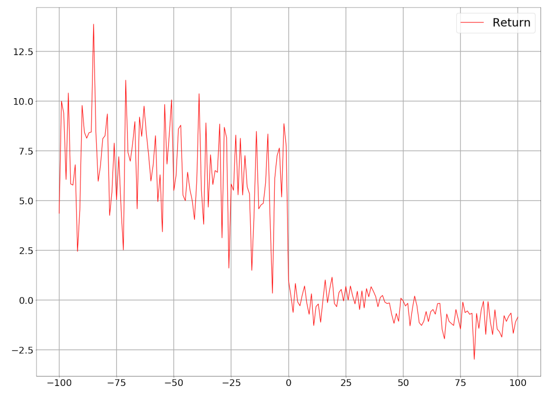
(b) Returns of sell orders within 10 seconds



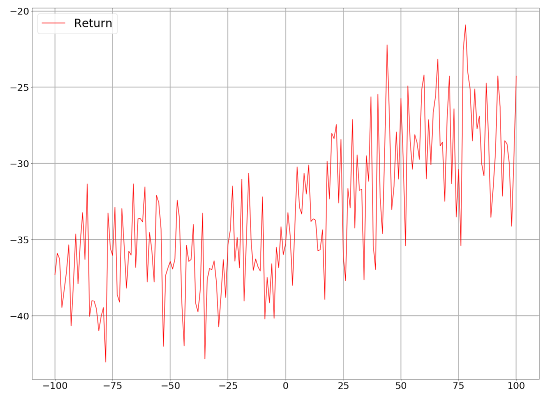
(c) Returns of buy orders within 30 seconds



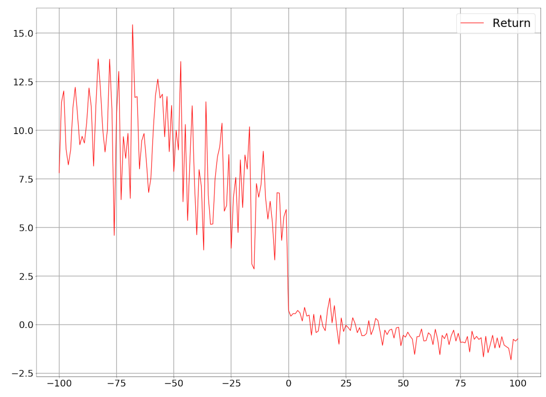
(d) Returns of sell orders within 30 seconds



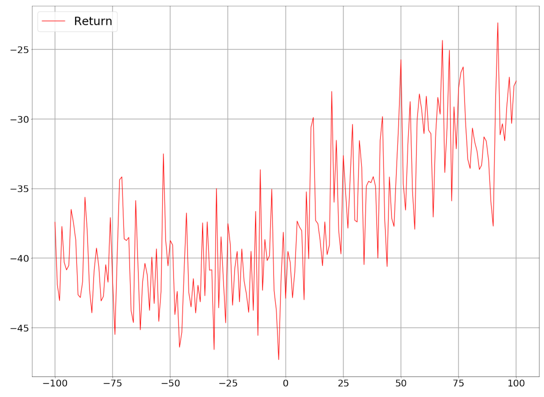
(e) Returns of buy orders within 60 seconds



(f) Returns of sell orders 60 seconds

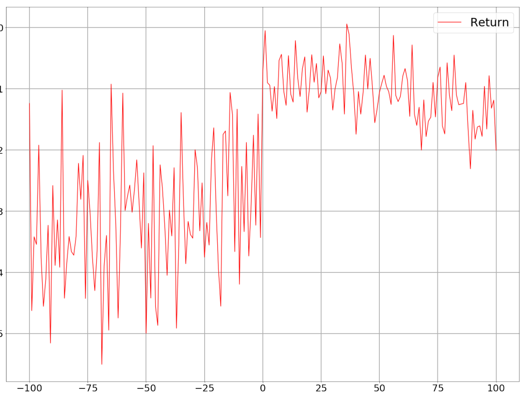


(g) Returns of buy orders 100 seconds

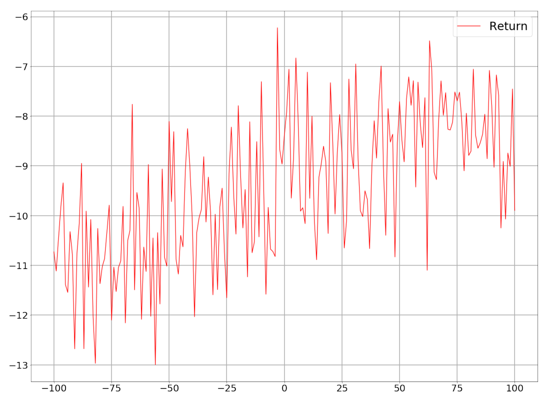


(h) Returns of sell orders 100 seconds

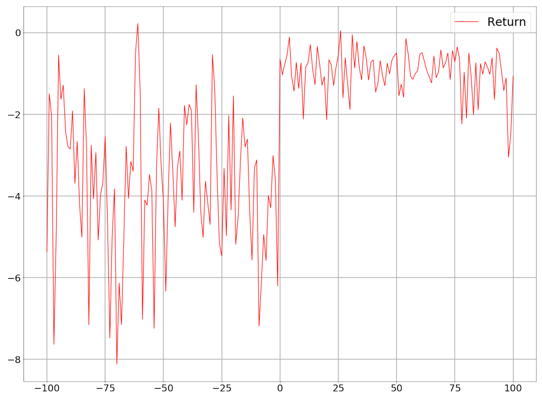
Figure 6.3: Returns of buy and sell orders executed within 10, 30, 60 and 100 seconds on data set I.



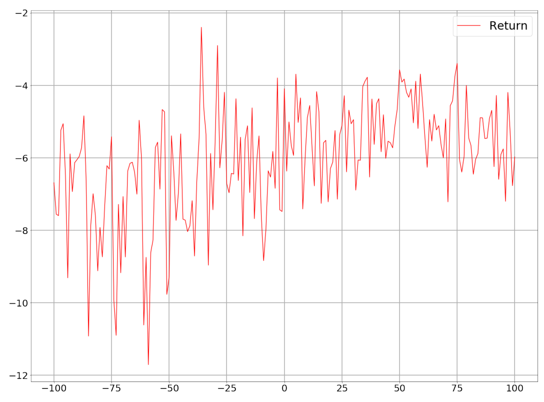
(a) Returns of buy orders within 10 seconds



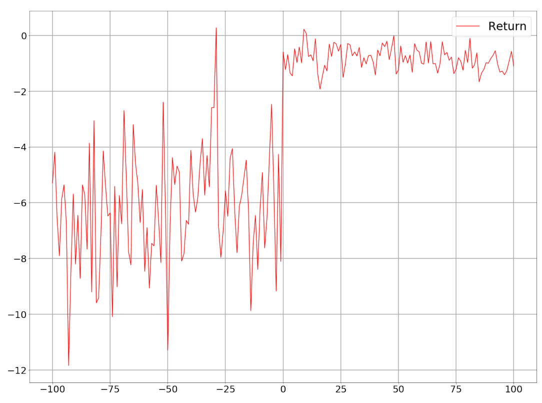
(b) Returns of sell orders within 10 seconds



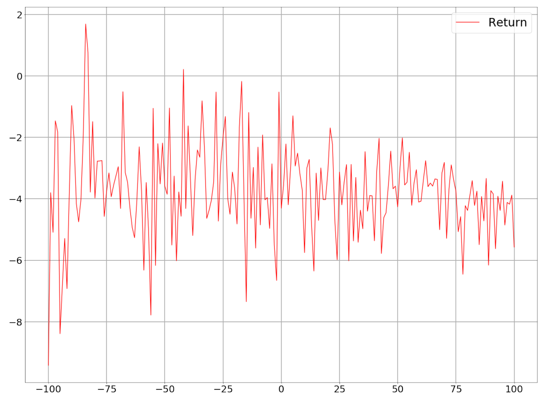
(c) Returns of buy orders within 30 seconds



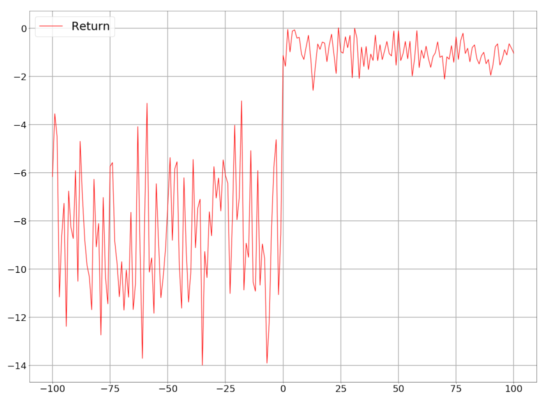
(d) Returns of sell orders within 30 seconds



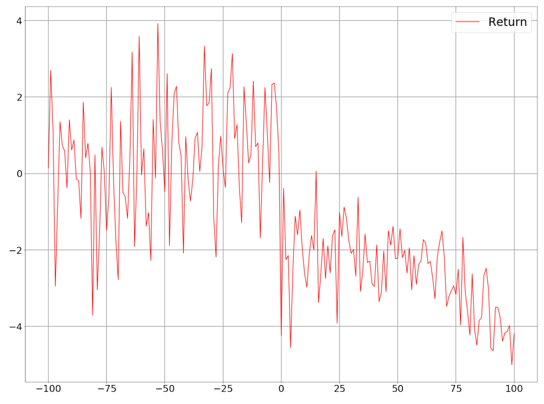
(e) Returns of buy orders within 60 seconds



(f) Returns of sell orders 60 seconds



(g) Returns of buy orders 100 seconds



(h) Returns of sell orders 100 seconds

Figure 6.4: Returns of buy and sell orders executed within 10, 30, 60 and 100 seconds on data set II.

6.3.2. Order placement behaviour on data set II

For the data set II, which shows an upwards price trend, the intuition is the opposite as for the investigation using data set I. 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. Contrarily, better returns with 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 6.4, with time horizons of 10, 30, 60 and 100 seconds respectively. The x-axis denotes the placement of the order at limit levels reaching from -100 to +100 and the y-axis indicates the average received return.

The returns of buy orders filled within a time horizon of 10 seconds, as shown in Figure 6.4a, correlate with the above stated assumptions. The highest returns are achieved when crossing the spread and although limit levels in the range of 1-50 tend to perform the same, and considering the risk of paying a premium, the wisest choice for the agent would be to choose the level closest to the spread. The sell orders placed, with a time horizon of 10 seconds, contradict the assumptions, as shown in Figure 6.4b. 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 6.4c, the orders placed above the spread become stable for any such limit level, much more so than with the same time horizon 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 6.2, as there are more market participants willing to sell. The return curve that indicates sell orders placed by an agent, shown in Figure 6.4d, becomes more evenly distributed. Therefore, limit orders tend to become more rewarding and an agent might benefit from a slight increase in price within the given time horizon.

This pattern becomes clearly appararent when a time horizon of 60 and 100 seconds was given, as shown in Figures 6.4f and 6.4h respectively. With the increased time horizon, the assumptions stated in the beginning of this section are confirmed and the agent, when trying to sell shares, should indeed place orders deep in the order book. As time passes and the market price rises, market participants are willing to buy for an increasing price and an agent is expected to be 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 can be expected. More precisely, for a time horizon of 100 seconds, the agent is expected to 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 6.4e and 6.4g which show the expected results of an agent that buys assets within the 60 and 100 seconds respectively. As is evident, during this uprising market, the expected damage can be minimized by crossing the spread and buying immediately. The advice stated before remains: 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.

6.3.3. Conclusion of empirical analysis

The results of the empirical analysis proceeded on data set I and II have shown that the tendency of limit order execution is as follows: during a downwards trend of the market price, buying assets can be optimized by placing limit orders deep in the order book and the least loss is taken when selling assets immediately to the market price. Contrarily, during an upwards market price trend, buying assets is suggested using a market order and selling can be optimized by placing the sell orders deep in the order book. This effect becomes more apparent when a longer time horizon is given for an order, as for shorter time horizons the order placement process gets either intercepted by short term fluctuations or a lack of market participants providing volume to buy and sell assets. The logical assumption has therefore been proven 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. In this analysis, the orders placed had to be completely filled without the ability to make intermediary steps of cancellations and replacements of the order within the given time horizon. It is therefore to be evaluated by the reinforcement learners whether or not such adaptions to the order will result in a more favorable reward. In order to have a measure of comparison for the reinforcement learning agents to come, Table 6.1 summarizes the findings and shows the expected rewards for (1) the optimal limit level chosen and (2) an immediate completion of the order using a market order.

	Limit order (optimal)	**Market order**
Buy (I)	15.20	-0.05
Sell (I)	-27.70	-27.70
Buy (II)	-1.06	-1.06
Sell (II)	3.68	-1.72

Table 6.1: Summary of rewards derived form the empirical analysis.

6.4. Q-Learning without market variables

The previous section provided knowledge about the expected rewards an agent will receive when placing buy and sell orders using the reinforcement learning environment and with the underlying data set I and II. 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.

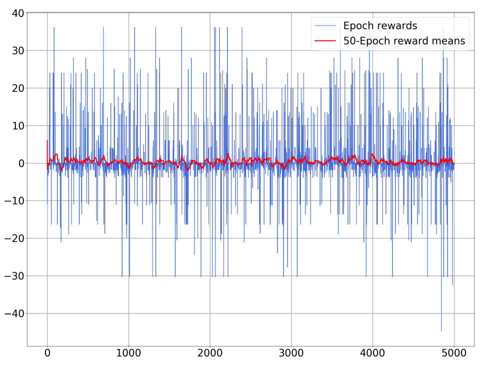
This section aims to investigate whether or not a Q-Learning agent, as described in Chapter 5 (Section 5.2), can reproduce the optimal achieved results shown in the previous section. In addition, the agent is allowed to cancel its order after every 10 seconds and place a new order with the remaining inventory, until the time horizon of 100 seconds is fully consumed. After which, and in case the order has not been filled completely, a market order follows for the remaining share to be bought or sold. For both data sets (I and II) an independent learning experiment is being proceeded where 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 placed and evaluated (known as "backtesting") 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 $\gamma = 0.5$ is chosen to balance the agents incentive to profit from immediate rewards and consuming the entirety of the given time horizon. Initially, the exploration constant ϵ is set to 0.8 and a decay is applied such that the factor reduces to $\epsilon =\sim 0.1$ by the time the training is completed. This allows the agent first to explore the action space and then exploit on the learned optimal actions to take.

With this setup, four observations were made and for each of which the training and testing results are stated below. During training, the mean rewards and the average action chosen for each epoch, were recorded. Once the model was trained, a backtest was run on the test data sets where the agent executed orders by choosing from the learned optimal actions.

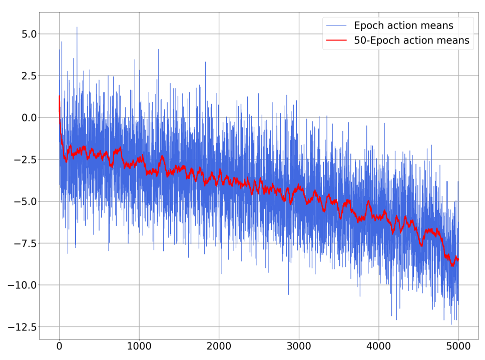
6.4.1. Results of training and testing on data set I and II

Figure 6.5 shows the training on data set I. The average received reward during the training is shown in Figure 6.5a. Over the course of 5000 epochs, the agent was able to improve the mean reward by approximately ~0.5, as a result of the change in chosen actions as illustrated in Figure 6.5b. The agent started off with the 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*. Comparing the results to the empirical analysis proceeded on the same data set, as shown in Figure 6.3, provides means for interpretation: The strategy learned by the Q-Learning agent performs worse than the expected cost of a market order, which was $-1.12 when buying 1.0 BTC and is shown in Section 6.3. The highly negative average actions the agent choose towards the end of the training indicates that the order might have oftentimes not been able to be filled within the time horizon and an expensive market order had to follow.

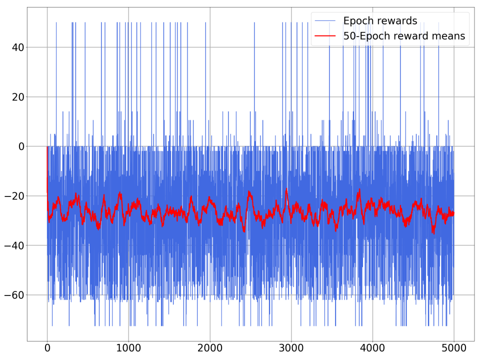
The rewards received for the agents tasks to sell the assets are much more volatile than for buying, as shown in Figure 6.5c, and no clear improvement can be seen. Consequently, there was no significant adjustment made by the agent regarding the chosen actions, as indicated in Figure 6.5d. The agent 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.



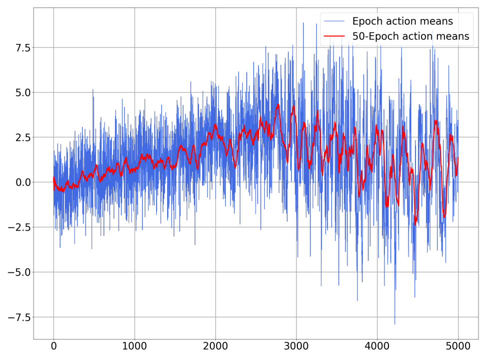
(a) Mean rewards per epoch (buy)



(b) Mean of actions per epoch (buy)

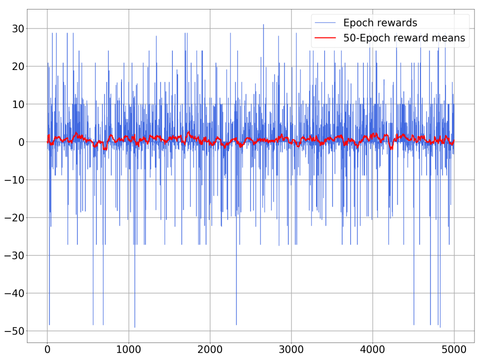


(c) Mean rewards per epoch (sell)

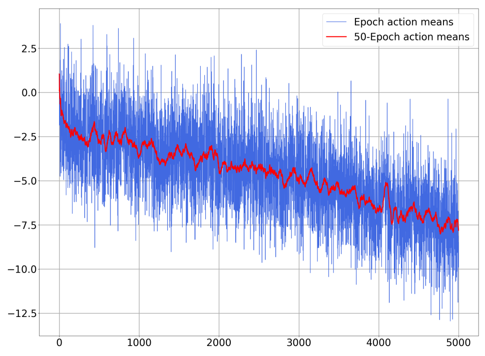


(d) Mean of actions per epoch (sell)

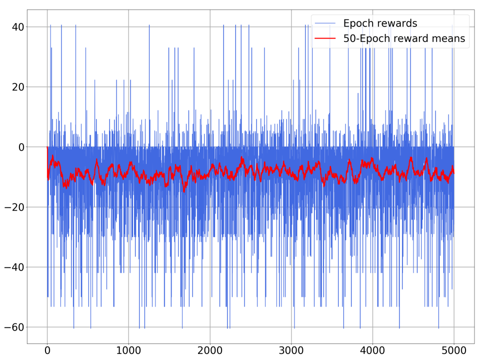
Figure 6.5: Mean rewards and actions for buying and selling on training data set I.



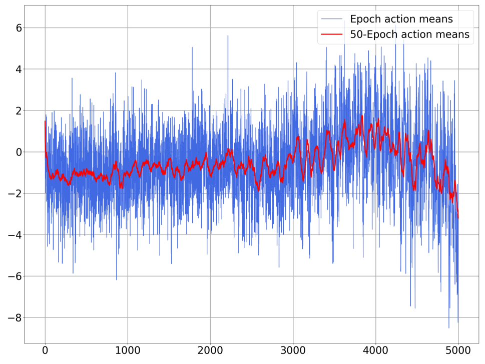
(a) Mean rewards per epoch (buy)



(b) Mean of actions per epoch (buy)



(c) Mean rewards per epoch (sell)



(d) Mean of actions per epoch (sell)

Figure 6.6: Mean rewards and actions for buying and selling on training data set II.

Figure 6.6 shows the experiment proceeded on data set II. The average received reward while training to buy the asset is shown in Figure 6.6a. Throughout the epochs, the agent was able to improve the mean reward like with the previous data by approximately 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 6.6b. 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 agents strategy is saving $0.02 when buying 1.0 BTC. Considering the empirical analysis proceeded for buying on this data set, as shown in Figure 6.4, and comparing it to the received rewards by the agent, implies that the agent failed to execute orders with the placed limit orders and oftentimes market orders must have followed.

Similar to the sell orders placed on data set I, the rewards received for the agents tasks to sell the assets on data set II are very volatile, as shown in Figure 6.5c. No improvement can be seen from the rewards during the training and no significant adjustment was made by the agent regarding the chosen actions, as indicated in Figure 6.6d. The backtest resulted in an average reward of -4.74 achieved by the agent, whereas market orders are expected to result in an average reward of -1.72. Hence, the agent causes to pay a premium of $3.02 for selling 1.0 BTC.

6.4.2. Conclusion of Q-Learning approach

	Q-Learner	**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

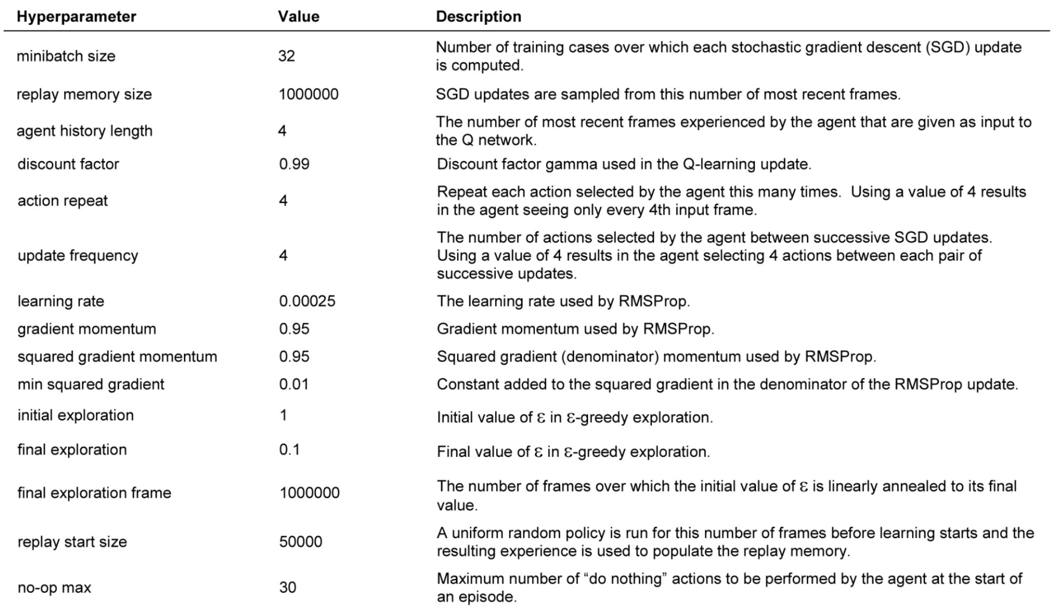
Table 6.2: Summary of rewards for the Q-Learning agent and market orders.

The findings of this section are summarized in Table 6.2. 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 purchase or sale, would 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 $\gamma = 0.3$ and therefore rely more extensively on the future rewards. However, no improvement could be achieved 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. Furthermore, it is to be assumed that the absence of market variables, while only relying on the given rewards, makes it hard for any learner to determine an optimal strategy. Therefore, the following section will make use of market variables in order to determine whether or not a learner can exploit the information hidden in the market and therefore act in favour of optimally placing limit orders.

6.5. Deep Q-Network with market features

In the previous section the Q-Learning agent was trained on the given data sets and no significant optimization in terms of buying and selling assets was achieved. By considering the previously found results of the empirical analysis of the limit order placement behaviour we found that most of the limit orders placed by the Q-Learning agent were not filled within the given time horizon and instead market orders had to be submitted after the time was consumed.

In this section we aim to determine whether or not the DQN agent is capable of extracting information provided by the raw market features and therefore improve the limit order placement strategy. The setup is the same as in the previous section whereas the agents task is to buy and sell 1.0 BTC within 100 seconds (discrete step size of 10 seconds) on both data sets I and II. Furthermore, both features, which were constructed in Chapter 4, will be applied and investigated separately. The input shape of the CNN is therefore determined by the chosen feature and is described below. Furthermore, for both DQN agents, we relied on the default hyperparameters worked out by Mnih et al. [22], as shown in Figure 6.7. Finally, we conclude our findings and determine the capabilities of the DQN approach (and its use of the market features) by comparing it to the expected rewards found in Section 6.3.

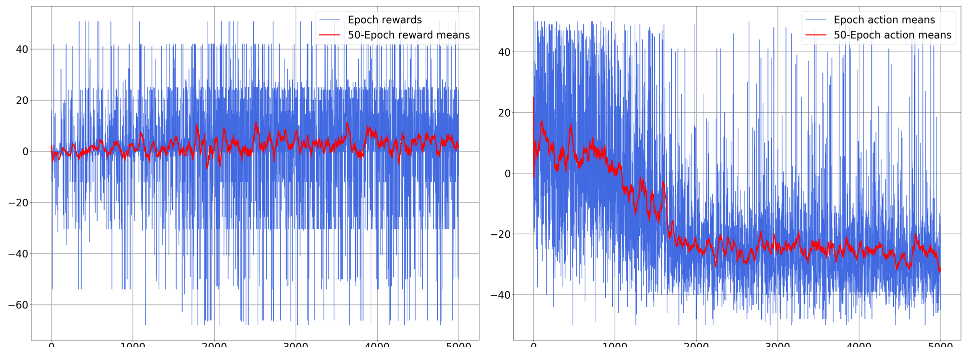


Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.
no-op max	30	Maximum number of "do nothing" actions to be performed by the agent at the start of an episode.

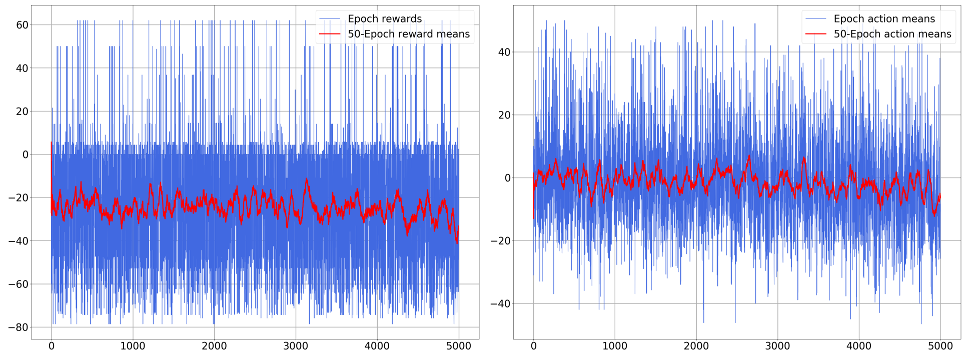
Figure 6.7: The values of all the hyperparameters were selected. We did not perform a systematic grid search owing to the high computational cost, although it is conceivable that better results could be obtained by tuning these hyperparameter values.

6.5.1. Application of historical order feature

The following evaluation of the DQN agent considers historical order book states as described in Chapter 4 (Section 4.4.1) which we denote henceforth as Feature I. Therefore, a "lookback" of 30 historical order book states are considered and the number of level in each state is limited to 20 for bids and asks respectively. Consequently this feature set is of size: $(2 * lookback, limitlevels, 2) \implies (60, 20, 2)$. Including the two private variables by appending a vector $[inventory, time]$ at the beginning of this feature vector results a feature set size of: $(61, 20, 2)$ and serves as the input for the CNN.

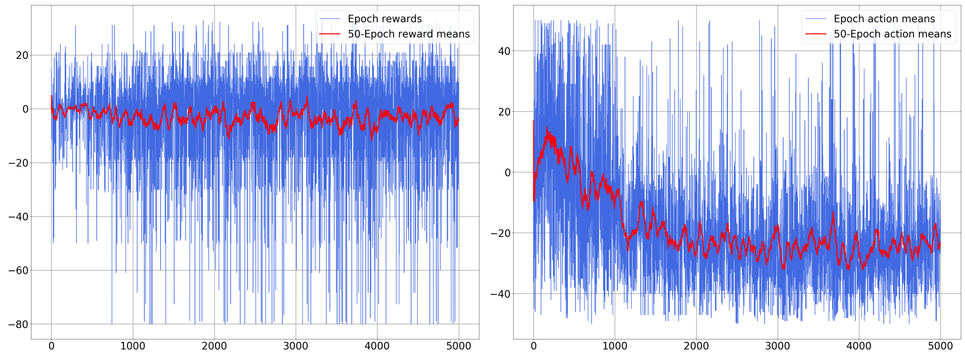


(a) Reward per epoch (buy) (b) Mean of actions per epoch (buy)

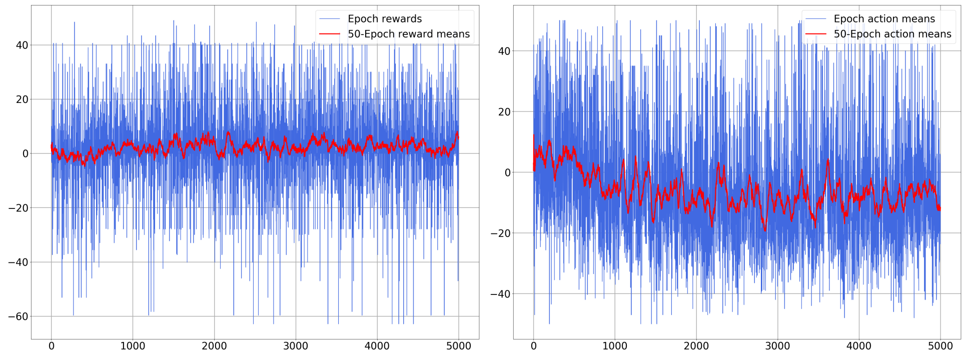


(c) Mean rewards per epoch (sell) (d) Mean of actions per epoch (sell)

Figure 6.8: DQN agent rewards and mean of actions for buying and selling on training data set I using feature I.



(a) Reward per epoch (buy) (b) Mean of actions per epoch (buy)



(c) Mean rewards per epoch (sell) (d) Mean of actions per epoch (sell)

Figure 6.9: DQN agent rewards and mean of actions for buying and selling on training data set II using feature I.

Figure 6.8 shows the learning process using the data set I. During the training process of buying assets, the received rewards were consistently improve over the course of the 5000 epochs (Figure 6.8a), throughout which the agent steadily adjusted the actions to be chosen more negatively (Figure 6.8b), indicating that limit orders placed at levels deep in the book were chosen. After 2000 epochs, the average chosen action stagnated just below -20, which is $2.00 below the market price. It is evident that this adjustment is not as linear as during the training of the Q-Learning agent as shown in the previous section. However, the adjustment is appropriate since the market price was falling. The achieved reward during the backtest resulted in 22.06, which is a significant improvement compared to the expected return of -0.05 for a market order.

The received rewards for selling is shown in Figure 6.8c and the average chosen action throughout the epochs is shown in Figure 6.8d. The rewards were not improved over the course of the training and the agent did not choose to adjust the actions on average. Possibly, this is due to the constant negative rewards received during the training under the difficult market conditions for selling assets while the market price is falling. As a result, during the backtest a negative reward of -49.26 was achieved, much worse than a market order which comes with the expected return of -27.70 and therefore indicate that the agent despite the fact of a falling market tried to sell with limit orders instead of market orders.

Figure 6.9 shows the agents learning process using the data set II. Over the course of 5000 epochs the agent was not able to improve its reward and instead stagnated on average at approximately $0.00 rewards per epoch, as shown in Figure 6.9a. The agent adjusted the chosen actions steadily such that the average limit level was below -20 at epoch 2000 and remained at this level, as shown in Figure 6.9b. Therefore the reward was a product of market orders as the orders with these limit levels were most likely not filled. After all, the backtest resulted in an average reward of -2.26 which is, compared to the expected market order reward of -1.06, slightly worse.

The rewards for process in which the agent was learning to sell assets is shown in Figure 6.6c. The reward could not be improved during the training, during which the agent lowered the average action chosen below -20, as shown in Figure6.6d. Under this uprising market condition, the observed rewards are a product of the market order, since the placement of limit orders as deep in the order book will will likely not result in a filled order.

The rewards for selling assets under the rising market conditions could be improved and, after 1000 epochs, kept constantly above $0.00, as shown in Figure 6.6c. Therefore, the average of actions chosen for an epoch was lowered within the first 1000 epochs and, going forward, remained at approximately -10, which correlates with the received rewards. Finally, the backtest resulted in an average reward of 0.84, which is an improvement compared to the expected reward of -1.72 for a market.

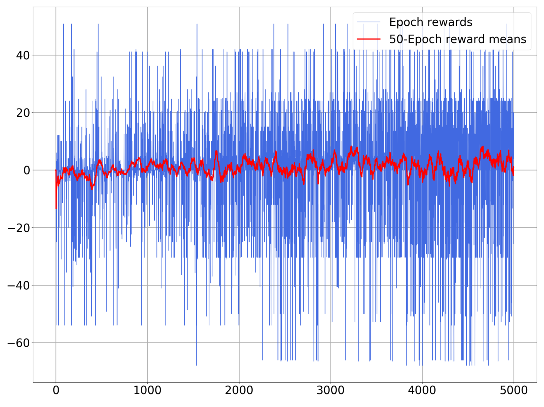
Table 6.3 summarizes the average rewards observed during the backtest using the DQN agent that uses Feature I, which includes 25 historical order book states as features and therefore has knowledge about the change in created and cancelled orders. Overall, the DQN agent was able to optimize limit order placement when the given market conditions came in favor of making a purchase or sale respectively. More precisely, significant improvements were made when the task was to buy assets and the market price was falling and minor optimization was achieved when the task was to sell assets and the market price was rising. Under these conditions, the DQN agent performed better than the Q-Learning agent. However, when market conditions did not come in favor of the intention to either buy or sell respectively, the agent performed not only worse than the expected return of a market order but also than the Q-Learning agent.

	DQN (Feature I)	**Market Order**
Buy (I)	22.06	-0.05
Sell (I)	-39.26	-27.70
Buy (II)	-2.26	-1.06
Sell (II)	0.84	-1.72

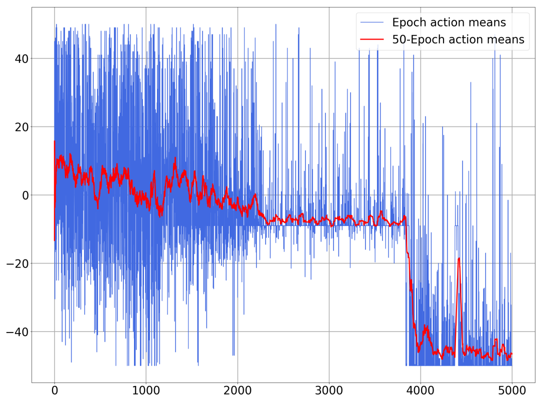
Table 6.3: Summary of rewards during backtest of DQN agent using Feature I (historical orders).

6.5.2. Application of historical trade feature

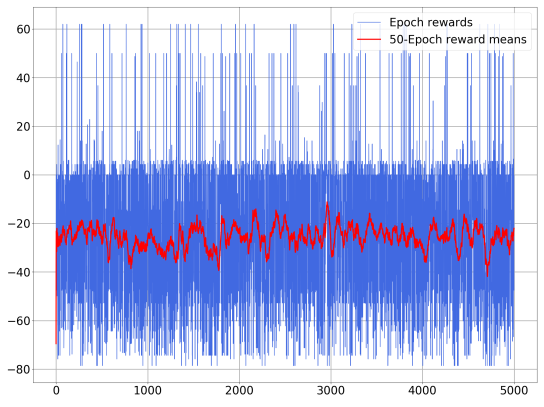
The following evaluation of the DQN agent considers historical trades as described in Chapter 4 (Section 4.4.2) which we denote henceforth as Feature II. Therefore, a "lookback" of 30 trades are considered. Consequently this feature set is of size: $(lookback, 4) \implies (30, 3)$. Including the two private variables by appending a vector $[inventory, time, 0, 0]$ at the beginning of this feature vector results a feature set size of: $(31, 4)$ and serves as the input for the CNN.



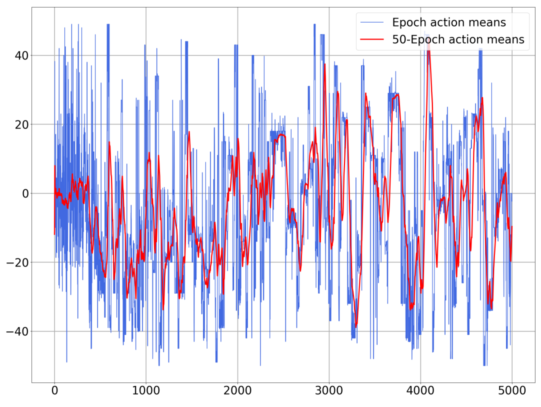
(a) Reward per epoch (buy)



(b) Mean of actions per epoch (buy)

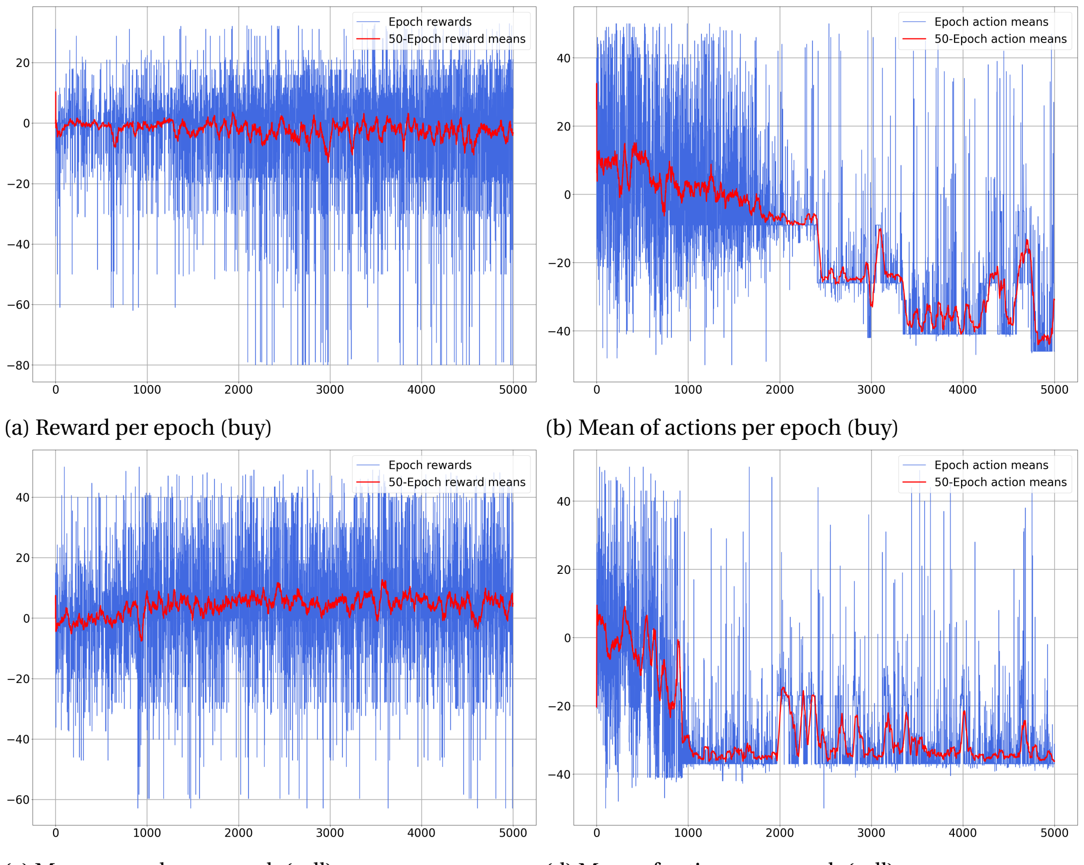


(c) Mean rewards per epoch (sell)



(d) Mean of actions per epoch (sell)

Figure 6.10: DQN agent rewards and mean of actions for buying and selling on training data set I using feature II.



(a) Reward per epoch (buy) (b) Mean of actions per epoch (buy)

(c) Mean rewards per epoch (sell) (d) Mean of actions per epoch (sell)

Figure 6.11: DQN agent rewards and mean of actions for buying and selling on training data set II using feature II.

Figure 6.10 illustrates the learning process of the agent for buying and selling using data set I. Similar to the agent that was provided with Feature I, the rewards received cold be slightly improved over the course of 5000 epochs, as shown in Figure 6.10a. Interestingly, the average action per epoch converged just below -40 after an abrupt adjustment was made after 4000 epochs. The average reward achieved during the backtest is as high as 31.92 and is a clear improvement compared to the expected market order return of -0.05.

The rewards received for selling remained at -20, as shown in Figure 6.10c. However, the average of the actions chosen remain volatile and no clear trend can be seen in Figure 6.10d. This is by no means a negative sign, perhaps the agent indeed learned to respond on distinct patterns with appropriate measures. Indeed, during the backtest an average reward of -35.15 was achieved, which is still worse than the expected return of a market order but significantly better than the DQN agent under the application of Feature I.

Figure 6.11 illustrates the same learning process with the application of data set II. The rewards for the buying process were not improved (Figure 6.11a) and the actions were adjusted towards limit levels deep in the order book (Figure 6.11b. The backtest resulted in an average reward of -3.56, slightly worse than the expected market return of -1.06.

Rewards for selling increased during the training, as shown in Figure 6.11c and the chosen average action remained just above -40 after 1000 epochs, as shown in Figure 6.11d. The resulted average reward after the backtest was 0.15, which is an improvement to the expected market order return of -1.72.

Table 6.4 summarizes the average rewards received for the DQN agent that makes use of Feature II and considers the past 25 trades prior each of the 1000 epochs and order placements respectively. Similar to the previous application of Feature I to the agent, significant optimization could be achieved when market conditions came in favour of buying or selling respectively. The application of Feature II resulted that the agent was able to outperform the DQN agent with Feature I when it comes to buying when the market price is in tendency to fall. Slightly worse performance was achieved when attempting to sell and the market price was rising, although the received reward was much better than the expected market order return. However, when the market conditions came not in favor of the intention to either buy or sell, the agent performed equally worse as the application of Feature I.

	DQN (Feature II)	**Market Order**
Buy (I)	31.92	-0.05
Sell (I)	-25.15	-27.70
Buy (II)	-3.56	-1.06
Sell (II)	0.15	-1.72

Table 6.4: Summary of rewards during backtest of DQN agent using Feature II (historical trades).

6.6. Determining the limitations of the DQN agent

In this section we intend to determine the capabilities and limitations of the DQN agent with the use of Feature I in greater detail. Sample order submissions are investigated which uncover the actions chosen by an agent throughout one epoch. Therefore, we will be able to see which market situations prevented the agent from achieving a positive reward and make a statement why the agent might have chosen inappropriate actions. In addition, artificial order books are being created which serve as new data sets to train and test the DQN agent on. By doing so, we aim to determine the capabilities of the deep reinforcement learning technique under market conditions which are 1) not affected by short term fluctuations and 2) a hold constant spread between best bid and best ask is given.

6.6.1. Limitation arising from market situations or inappropriate actions from the agent

The agents inability to place an order that leads to positive rewards can be due to the following two reasons: 1) when the market situation does not allow to do so and 2) when the agent submits an inappropriate action.

Our investigations have shown that for the first category, not just upwards and downwards trends arise difficulties for the order placement but also when the spread between the best bid and best ask price becomes large. The latter is shown in Figures 6.12 and 6.13 where an agent attempted the placement of a buy and sell order respectively.

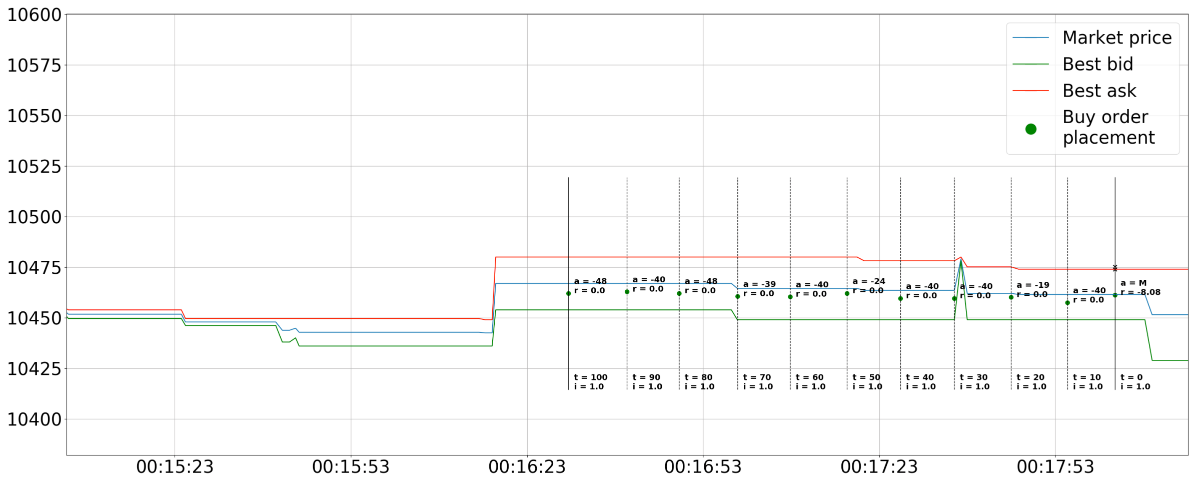


Figure 6.12: Wide spread between bid and ask prevents agent from buying.

Before the start of the buy order placement, the spread was very close between the best bid and best ask price, as indicated by the green and red lines. However, during the placement of this order, the spread widened and was almost for the entire time horizon larger than $50.00. Since the actions are segmented in discrete $0.10 steps, with a total of 101 steps, reaching from -$5.00 to +$5.00 relative to the market price, the agent had no chance of placing the orders close to the best ask price. As a result, the entire inventory of 1.0 BTC had to be bought by using a market order at the end of the time horizon, and the trades are marked with a cross. This generated a negative reward of -8.08. The same market situation is demonstrated during which the agent initiated the process of placing a sell order. Similarly, the pricing level was almost never close to the best bid price, except for once. As a result, a market order was followed with which the agent sold the remaining 0.9 BTC to a decreased price that resulted in a negative reward of -47.28. In addition, since there was not much liquidity offered by buyers, the market order was partially filled for ever decreasing prices, as indicated with the crosses.

A way to overcome this limitation would be to either increase the number of steps or to increase the action step size. Increasing the number of steps would enlarge the action space and therefore training the agent would become more difficult and computationally more expensive as the agent has more options to choose from. Increasing the action step size could arise inefficiencies when the spread between best bid and best ask is low as therefore the agent would likely place order too deep or too height in the order book.

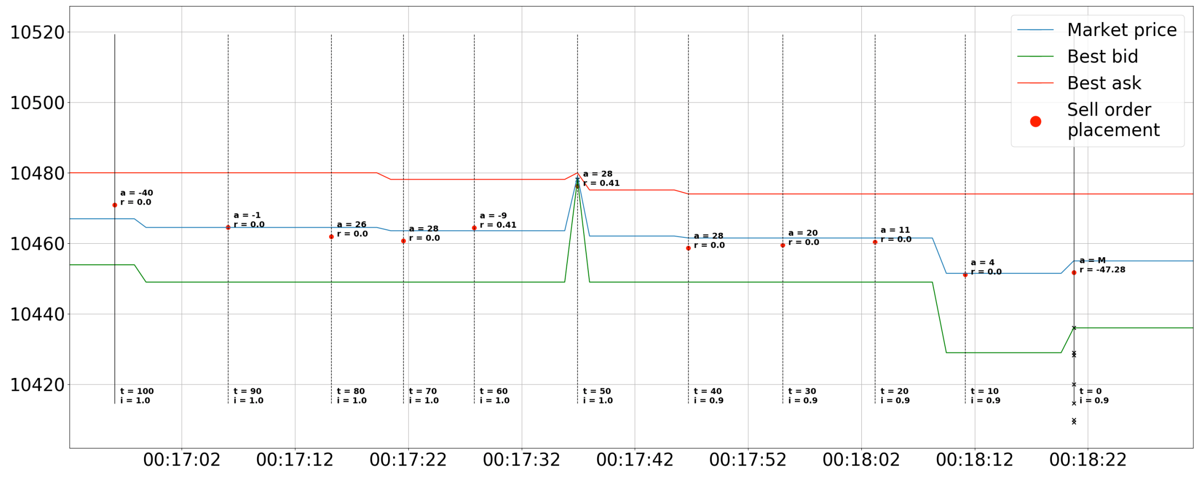


Figure 6.13: Wide spread between bid and ask prevents agent from selling.

An example of the seconds category, where the agent clearly failed to select an appropriate action, is shown in Figure 6.14 below. As is illustrated, the market price and the best bid and ask price were declining before the initialization of a sell order placement. The agent then decided to cross the spread with the first step and choose the action 28, in which it was willing to sell for $2.80 below the market price. By doing this, the order was immediately filled and resulted in a negative reward of -1.71. Ideally, the agent should have been patient and decide to place a limit order below the spread. The, in a subsequent step, the order could have been filled with a negative action, which would have resulted in a positive reward. It is to be assumed that the shown reaction of the agent was due to over-fitting of the previous declining market situation, in which it would have been indeed the better choice to immediately fill the sell order.

Such behaviour could likely to be improved by 1) enlarging the window size of the feature provided to the agent in order to cover long term market movements and 2) by increasing the training epochs which include such market situations.

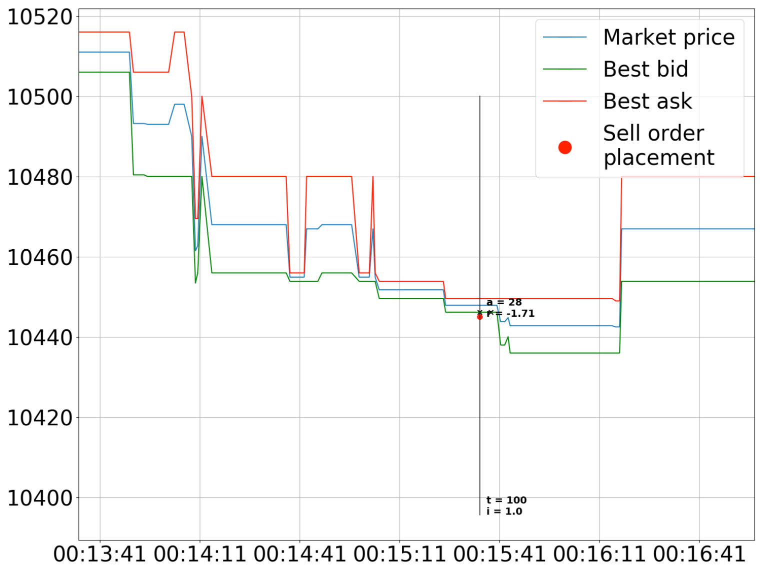


Figure 6.14: Wide spread between bid and ask prevents agent from selling.

6.6.2. Capabilities evaluated using artificial limit order books

6.7. Conclusion of reinforcement learning on limit order placement

Inability to learn under hard market conditions where rewards are dominated by negative values.

7 Conclusion and Future Work

7.1. Conclusion

simulation of historical order matching

data samples chosen during feature construction
data samples chosen during evaluation
advantages and disadvantages of q-learning and dqn
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] Matching algorithms. URL https://www.cmegroup.com/confluence/display/EPICSANDBOX/Matching+Algorithms. [Online; accessed April 30, 2018].

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

[6] 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].

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

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

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

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

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

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

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

[14] 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.

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

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

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

[18] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

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

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

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

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