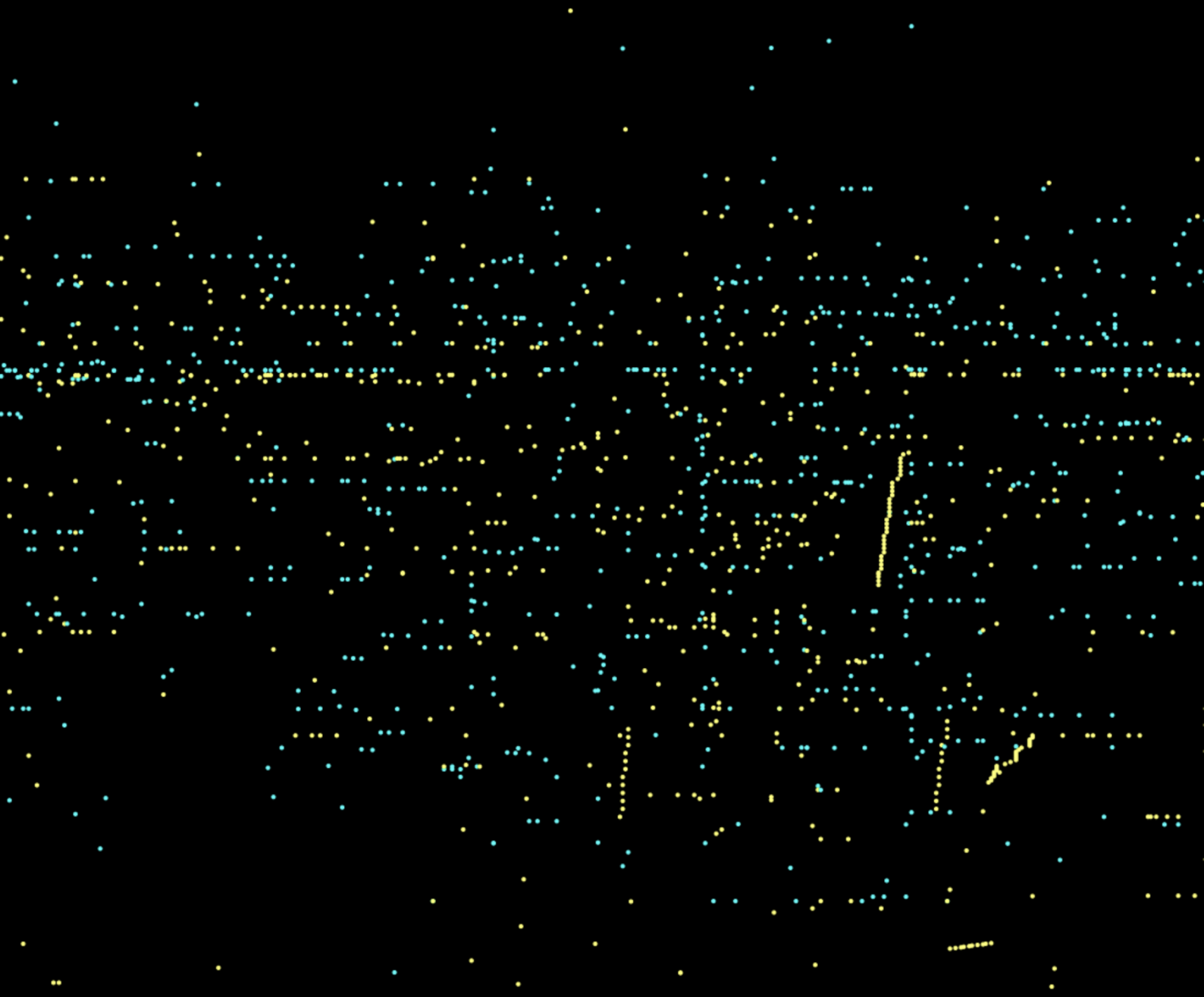
Limit order placement optimization with Deep Reinforcement Learning



Marc B. Juchli

Limit order placement optimization with Deep Reinforcement Learning

Learning from patterns in 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:	Dr. M. Loog,	TU Delft, supervisor
	Dr. J. Pouwelse,	TU Delft, co-supervisor
	Prof. dr. ir. M.J.T. Reinders,	TU Delft

This thesis is confidential and cannot be made public until July 12, 2018.

TU Delft

Abstract

For various reasons, financial institutions often make use of high-level trading strategies when buying and selling assets. Many individuals, irrespective or their level of prior trading knowledge, have recently entered the field of trading due to the increasing popularity of cryptocurrencies, which offer a low entry barrier for trading. Regardless of the intention or trading strategy of these traders, the invariable outcome is their attempt to buy or sell assets. However, in such a competitive field, experienced market participants seek to exploit any advantage over those who are less experienced, for financial gain. Therefore, this work aims to make a contribution to the important issue of how to optimize the process of buying and selling assets on exchanges, and to do so in a form that is accessible to other traders.

This research concerns the optimization of limit order placement within a given time horizon of 100 seconds and how to transpose this process into an end-to-end learning pipeline in the context of reinforcement learning. Features were constructed from raw market event data that related to movements of the Bitcoin/USD trading pair on the Bittrex cryptocurrency exchange. These features were then used by deep reinforcement learning agents in order to learn a limit order placement policy. To facilitate the implementation of this process, a reinforcement learning environment that emulates a local broker was developed as part of this work. Furthermore, we defined an evaluation procedure which can determine the capabilities and limitations of the policies learned by the reinforcement learning agents and ultimately provides means to quantify the optimization achieved with our approach.

Our analysis of the results of this work includes the identification of patterns in cryptocurrency trading that were formed by market participants who posted orders, and a conceptual framework to construct data features containing these patterns. We developed a fully-functioning reinforcement learning environment that emulates a local broker and, by means of this process, we identified which components are essential. With the use of this environment, we were able to train and test multiple reinforcement learning agents whose aims were to optimize the placement of buy and sell limit orders. During the evaluation, we were able to improve the parameter settings of the constructed reinforcement learning environment and therefore improve the policy learned by the agents. Ultimately, we achieved a significant improvement in limit order placement with the application of a state-of-the-art deep Q-network agent and were able to simulate purchases and sales of 1.0 BTC at a price that was up to $33.89 better than the market price.

We have made use of the OpenAI Gym[1] library and contributed our work to the community[2] to enable further investigations to be carried out. The work done in this thesis can be used as a framework to (1) build a component that acts as an intermediary between trader and exchange and (2) to enable exchanges to provide a new order type to be used by traders.

[1] https://github.com/openai/gym
[2] https://github.com/backender/ctc-executioner

Preface

This document contains the work done for my thesis for the completion of the Master of Science in Computer Science (Data Science track). The work was conducted and completed at the Pattern Recognition Laboratory (PRLab) at Delft University of Technology. This research stemmed from my interest in cryptocurrencies, for which I am grateful in many ways. My passion for software engineering and finance, as well as my fascination with machine learning, which I developed during my studies at TU Delft, drove me into advances in automated cryptocurrency trading. I gained an understanding of some of the many concepts present in order-driven financial markets. Soon I realized that market orders were the limiting factor for my strategies to become profitable, and that appropriate placements of limit orders would solve this problem. But how? This thesis attempts to provide answers to this question. The work mainly concerns the transposition of a financial problem into the reinforcement learning context and the explorations that were necessary thereafter. My efforts included work that is well summarized in the following quote from Marcos Lopez de Prado:

> It takes almost as much effort to produce one true investment strategy as to produce a hundred, and the complexities are overwhelming: data curation and processing, HPC infrastructure, software development, feature analysis, execution simulators, backtesting, etc. [19]

Many times during this project, I was reminded how important software engineering skills are, how much time a small mistake can cost, and how crucial it is to pay attention to details. I was able to strengthen my skills and expand my knowledge in mathematics and machine learning.

I am grateful for this journey and would like to thank my supervisor Marco Loog for his guidance and advice during this research. Our meetings and reviews he provided me were always a great help and urged me to take the extra step and think outside of the box. Additionally, I would also like to thank Johan Pouwelse, as my co-supervisor, for his equally valuable advice and perspective from an applied point of view. Our meetings were always inspiring and pointed towards the practical applications of my research, which made it without doubt an even more enjoyable project. I would also like to thank Mateusz Garbacz, with whom I had many discussions during our "hacker-nights"; he helped in this project by providing me with his perspective and creativity. I should also mention Satoshi Nakamoto, whoever that may be, who is not only the father of Bitcoin but has also created a new field of research. Last but not least, I would like to thank my friends and family for their moral support.

Marc B. Juchli
Delft, July 10, 2018

Contents

1 Introduction

Financial institutions make decisions to buy or sell assets for many reasons, including: customer requests, fundamental analysis[4], technical analysis[14], top-down investing[13], 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 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, 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[12]. 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 [39]. 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 [39]. 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 traded 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 [22]. 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[36], and developed a reinforcement learning strategy[37] 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: How can the application of deep reinforcement learning contribute to the optimization of limit order placement?

We chose to divide the research question into sub-questions as this choice 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 and construct features with the patterns found incorporated that enable deep reinforcement learning agents to learn an order placement strategy.

RQ 1.2: How can 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 a 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.

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. The second agent is a deep reinforcement learning agent that makes use of a neural network in order to detect patterns in market data.

RQ 1.3: How can one evaluate a reinforcement learning environment and 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 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. Further, we make the reader familiar with (Deep) Reinforcement Learning. In Chapter 3, we elaborate on the behavior of order placement 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 order 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

Preliminaries

In this chapter, we will provide background information in order to understand the previous work 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 interaction 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. 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 [9]. 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 [10]. 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 (in milliseconds) of the state,

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

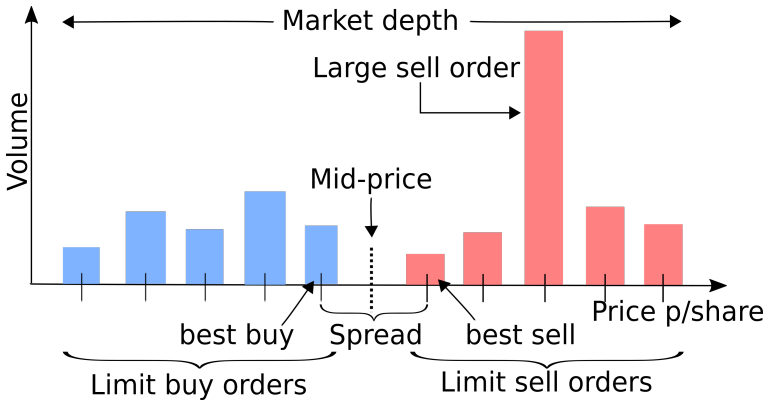


Figure 2.1: Figure taken from [6]. Simplified limit order book, which provides an understanding of some characteristics.

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. 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 [5]:

1. Limit orders (Eq. 2.3) may be partially filled or not filled at all if there are no parties on the opposing side.
2. Market orders (Eq. 2.4) 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.

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. [31] and Guo et al. [22]. 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[22, 37] 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

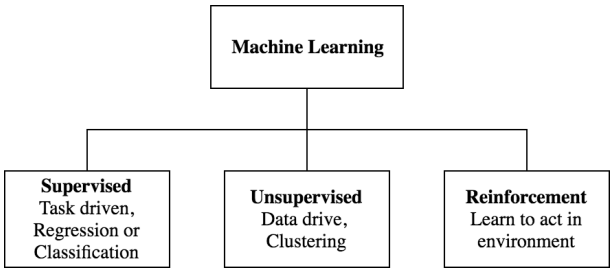


Figure 2.2: Categorization of machine learning techniques

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.

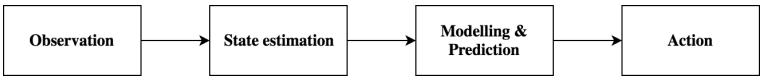


Figure 2.3: Reinforcement learning end-to-end learning pipeline

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 [8].

Example: Since we are working with financial systems, let us assume we want to buy and sell stocks on a stock exchange. In reinforcement learning terms, the trader is represented as an *agent* and the exchange is the *environment*. The details of the environment do not have to be known as it is rather regarded as a black box. The 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

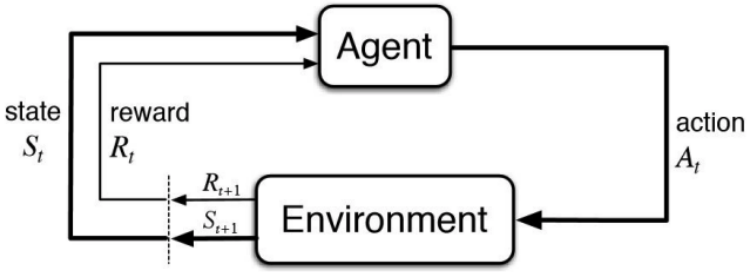


Figure 2.4: Figure taken from [8]: interaction between a reinforcement learning agent and the environment. An action is taken by the agent that results in some reward and a new state.

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 $\pi(s) : S \rightarrow A$ is used for a deterministic policy, whereas S is the set of possible states and A is the set of possible actions.

The stochastic *policy* at time step t: π_t is a mapping from state to action probabilities as a result of the 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 [7].

Value Functions

When the transition function of an MPD is not available, model-free reinforcement learning allows the agent to simply rely on some trial-and-error experience for action selection in order to learn an optimal policy. Therefore, the value of a state s indicates how good or bad a state is for the agent to be in, measured by the expected total reward for an agent starting from this state. Hence we introduce the **value function**, which depends on the policy the agent chooses its actions 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 [7]) as follows.

A *value based agent* starts off with a random value function and then finds a new (improved) value function in an iterative process, until reaching the optimal value function (Eq. 2.13). 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

"Reinforcement learning can be naturally integrated with artificial neural networks to obtain high-quality generalization" [3]. 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 θ as,

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

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.

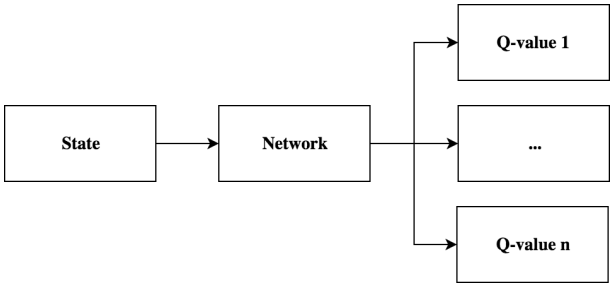


Figure 2.5: Neural network outputs Q-values

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 [34], as illustrated in Figure 2.6:

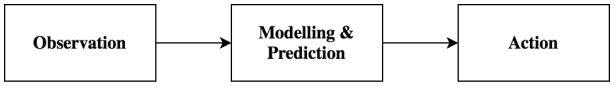


Figure 2.6: Deep Reinforcement learning end-to-end learning pipeline

3

Related Work

Compared to the execution problem, the literature for the order placement problem is sparse (as confirmed by Guo et al. [22]). 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.

3.1. Execution/Placement behaviour

Kearns et al. [36] 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.

Although this research does not optimize the placement of limit orders per se, it provides a method to measure the expected costs for (1) placing orders at certain limit levels or (2) submitting a market order. Therefore, this approach can serve as a measure to determine how well a method which does attempt to optimize the limit order placement performs. 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.

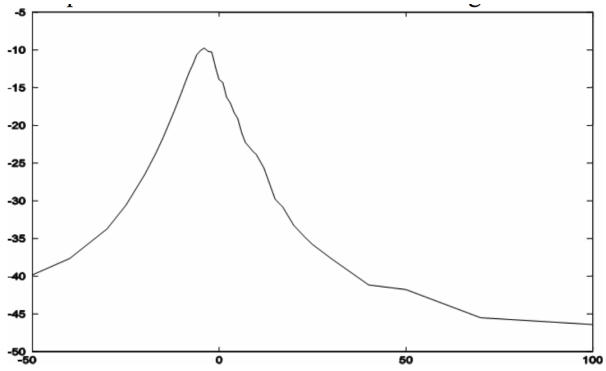


Figure 3.1: Taken from [36]. An illustration of the pricing strategy that produces the most favorable expected execution price.

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.

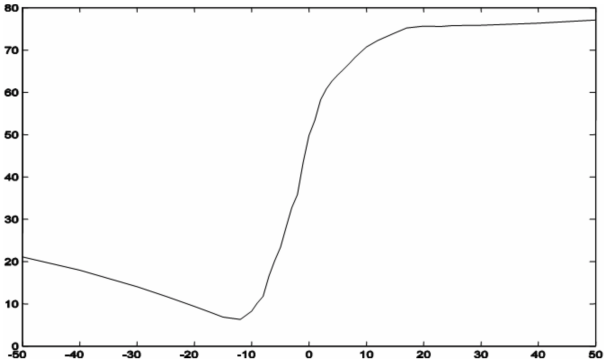


Figure 3.2: Taken from [36]. An illustration of the uncertainty of the expected execution price.

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.

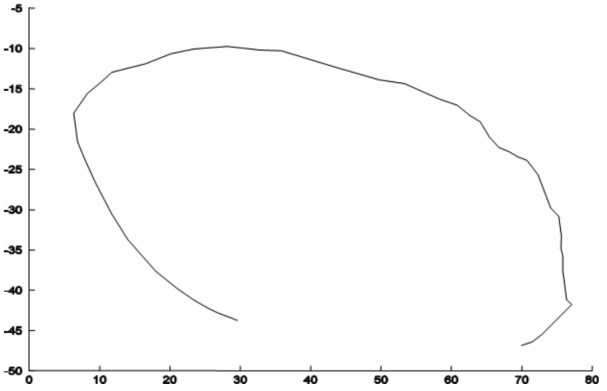


Figure 3.3: Taken from [36] An illustration of the trade-off between risk and return indicated by a efficient pricing frontier.

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 [33]). 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 [45]. 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 ($\Longrightarrow$ 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. [20] 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, it was suggested that the resulted predictions can 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 is mostly due to the fact that the zero movement prediction was accurate. An increased time delta resulted in significantly worse prediction accuracy.

A supervised learning approach, such as presented in this research, has multiple disadvantages. First, the model is incentivized based on the accuracy of bid and ask price predictions and therefore does not incorporate the outcome of order placements, at the predicted price levels, during training at all. Instead, with this setup the process of order placement would be attempted in a subsequent step where the predicted price determines at which price level to place the order. However, a multi-class setup would only provide knowledge whether to place a limit- or market order and does not specify the exact price level. Certainly, such an approach can easily be modified in order to become a regression problem where the price is defined specifically. However, the outcome of the order placement would still not have an impact on the loss function during training.

3.4. Reinforcement Learning approach

A large-scale empirical application of reinforcement learning to optimize trade execution has been presented by Kearns et al. [37]. 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 ($\frac{ask+bid}{2}$). 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 $t = 0$.

This work demonstrates the suitability of reinforcement learning for the limit order placement problem. The use of a reward function that involves the outcome of the proceeded placement itself allows to train a model specifically for this task. In addition, and unlike what was shown in Section 3.3, this architecture does not require a separation of the prediction and placement processes. Therefore, this research provides foundation for modelling and integrating the limit order placement problem into a reinforcement learning context with the use of a relatively complex environment and a simple agent. Furthermore, the results show the potential improvements that can be achieved by using market features.

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. As a result, 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` of the channel

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

`updateExchangeState`. The 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.

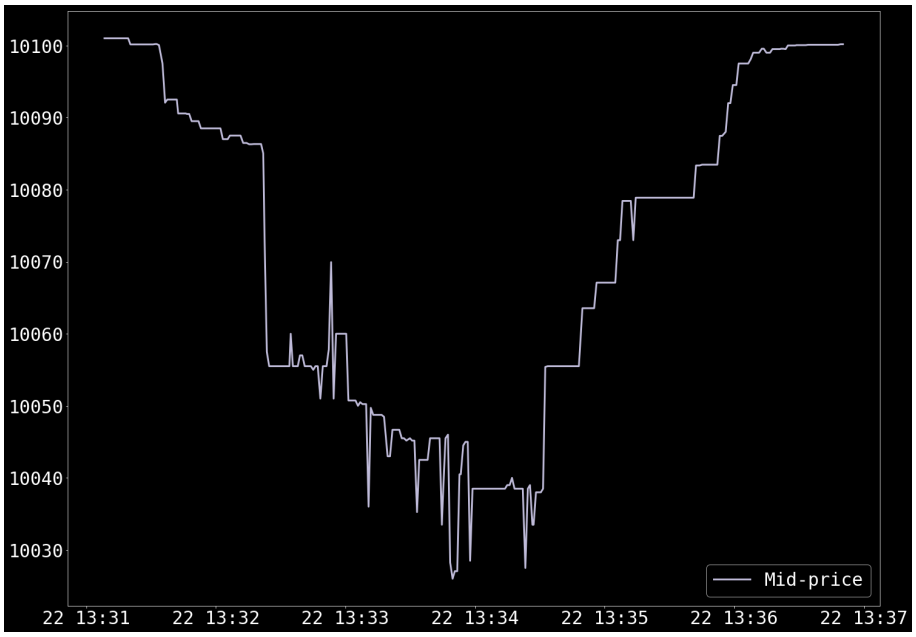
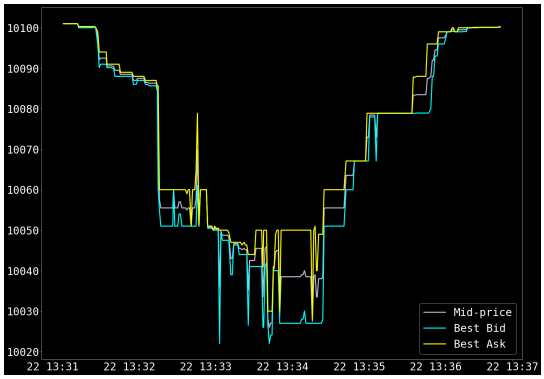
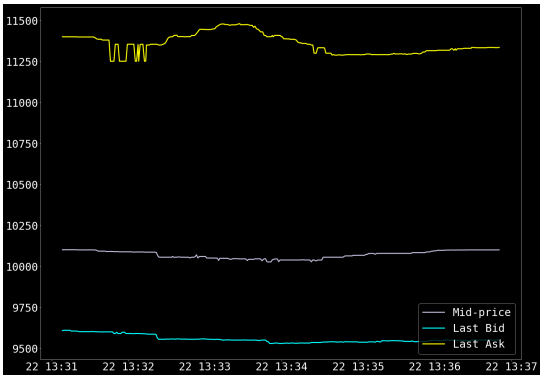


Figure 4.2: Price movement of sample data set

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.

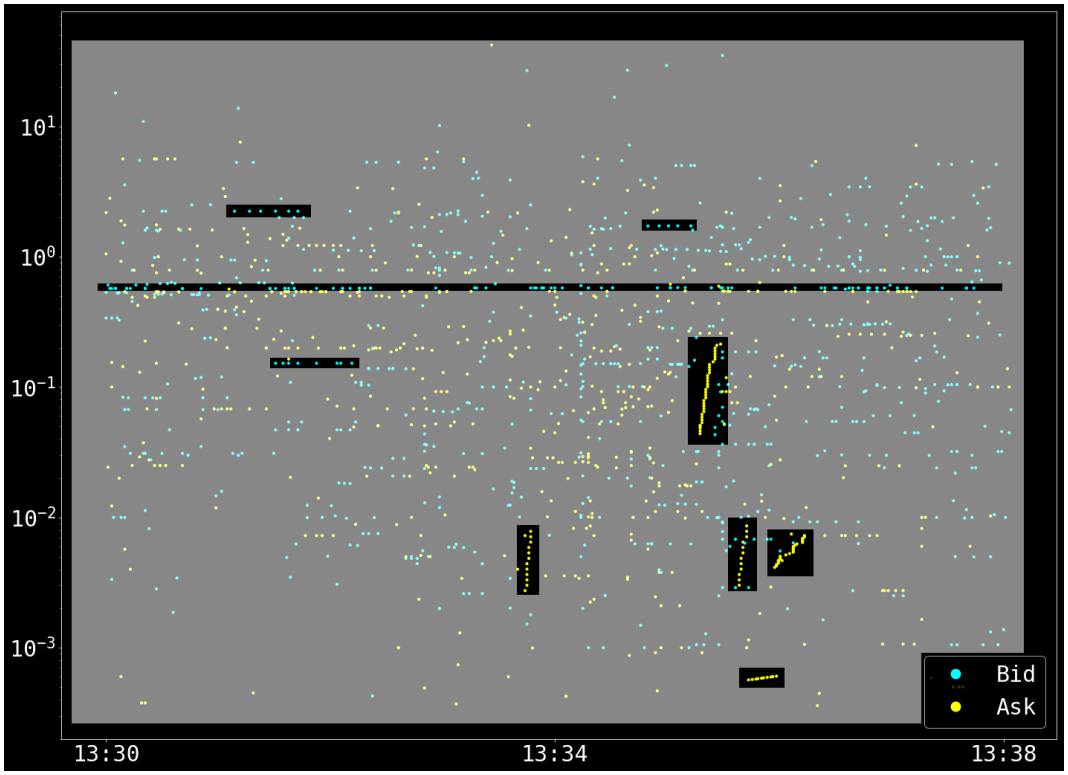


Figure 4.4: Volume map of created bid and ask orders.

4.2).

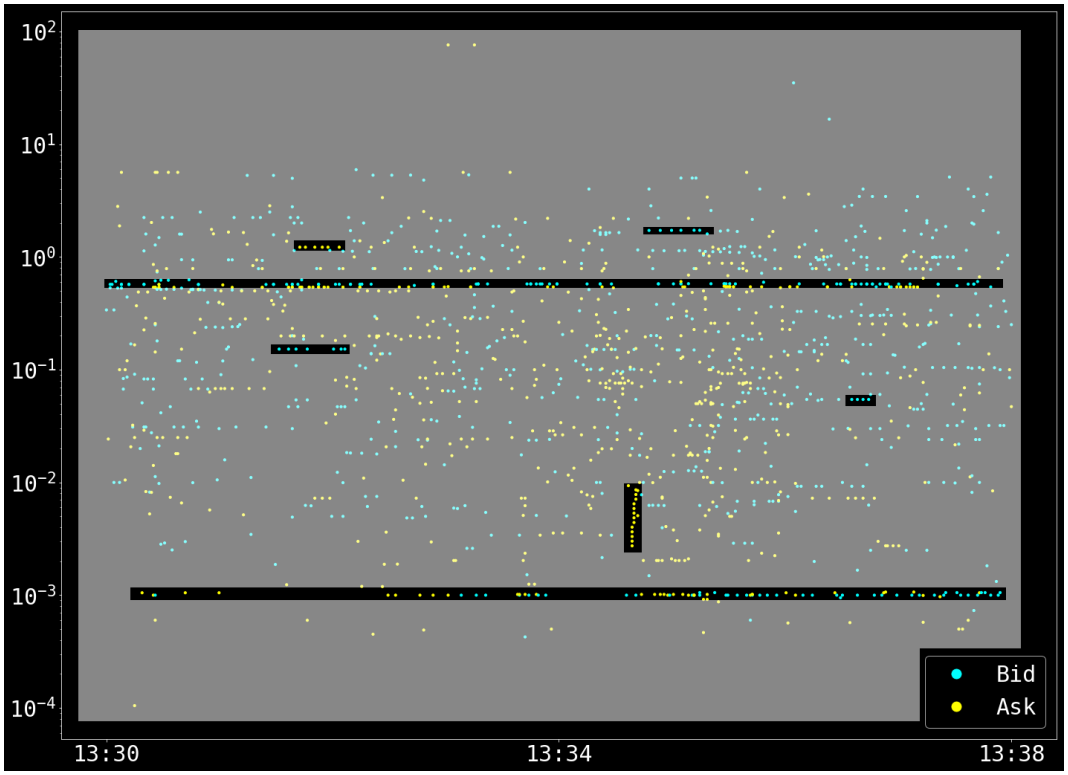


Figure 4.5: Volume map of cancelled bid and ask orders.

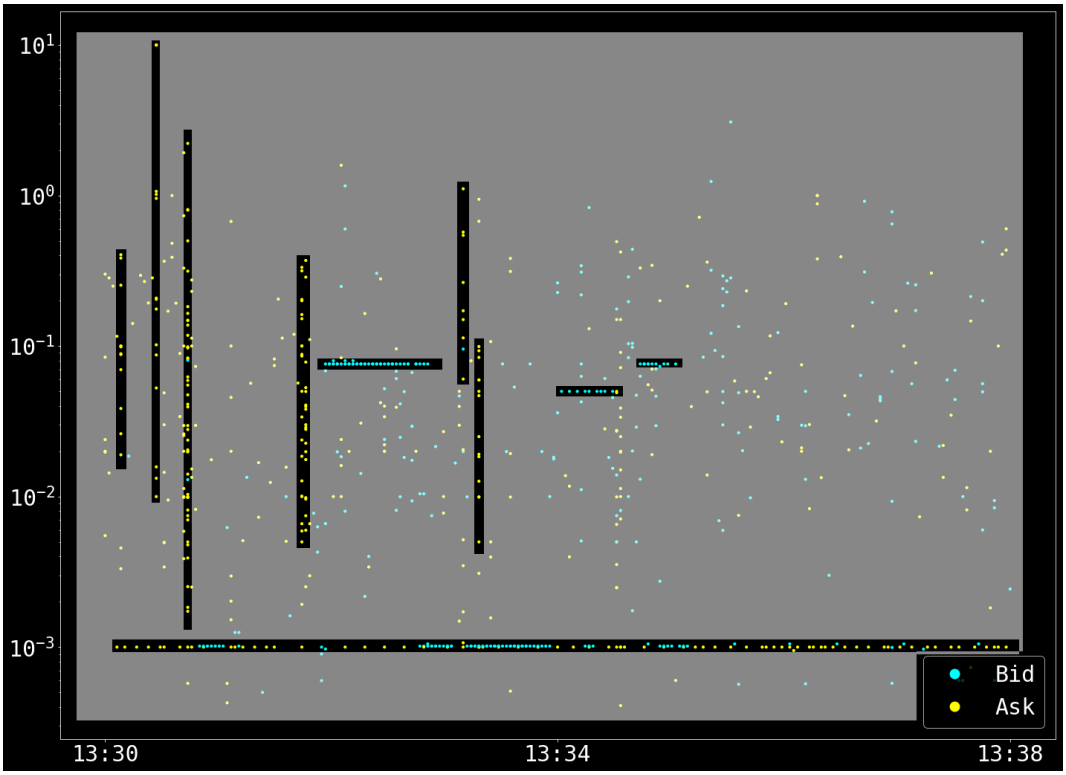


Figure 4.6: Volume map of trades initiated by a bid or ask order crossing the spread.

Figure 4.4 shows the volumes of the orders created. The volumes of most of the orders placed were less than 1.0 BTC (see y-axis 10^0 and below) 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[41]. 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.

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 Figure 4.4 before. Hence, it is likely that there was a trader following a strategy which involved creating and canceling orders in equal volumes[17]. 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 distribution of volumes of orders is an indicator of the 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 the volumes traded over time, in combination with the price at which the asset was traded.

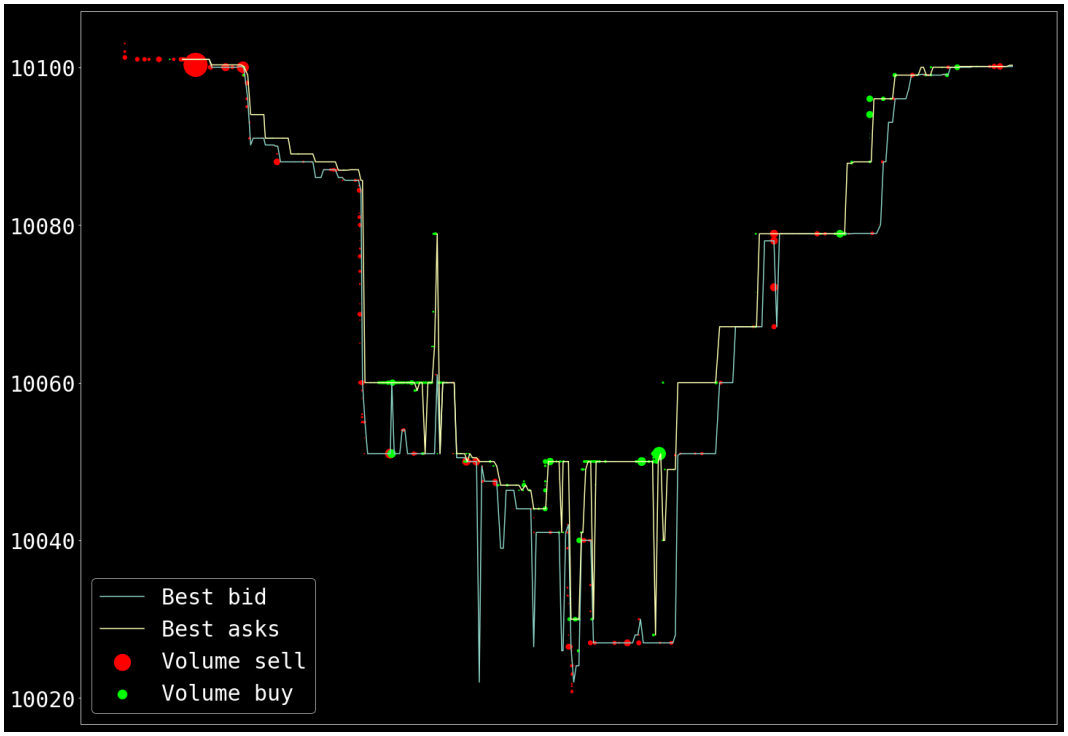


Figure 4.7: Relation of trade volume to price movement.

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 dots of a size that indicates the traded volume and position of each dot defines its respective 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 appeared consecutively, followed by one large sell transaction. After that, a series of buy orders with low volumes occurred, and this caused the minor spike. Interestingly, a rather large buy trade appeared shortly before the price started to rise again. Even though sell transactions took place 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. 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. 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 manually extracting features such as those shown in [20, 25, 37], the aim of this project is to acquire knowledge directly from raw inputs, similar to a proven, successful method in the gaming sector[34] and which was recently applied in the trading context[32].

4.4.1. Feature: price and size of historical orders

The first feature represents the order book (as defined in Eq. 2.5) and generated in Section 4.2. 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 $ts \leq t$. 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 have applied normalization to the prices (bp, ap) with respect to the best ask price for each state, that is ap_{i1}. Accordingly, we have also normalized the sizes (bs, as) by reference to the size provided at the best ask price as_{i1}. Lastly, we define $n = 40$ order book entries (maximum of what goes from collection) on the bid and ask side, and therefore make the most possible use of the available data.

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

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 determine whether or not trade events can provide a positive learning effect to the agent to optimize order placement, we will 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. In this feature, we take n trades into consideration, which occurred prior the time of the order placement. More precisely, the feature will be 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 its

previous historical trade. For the most recent historical trade, the time difference is measured from 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 served 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 movements were likely to be due to (1) an imbalance between bid and ask orders; (2) a certain 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 constructed 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 this 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, 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 and makes extensive use of the components previously described in Chapter 2. 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 [18] 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

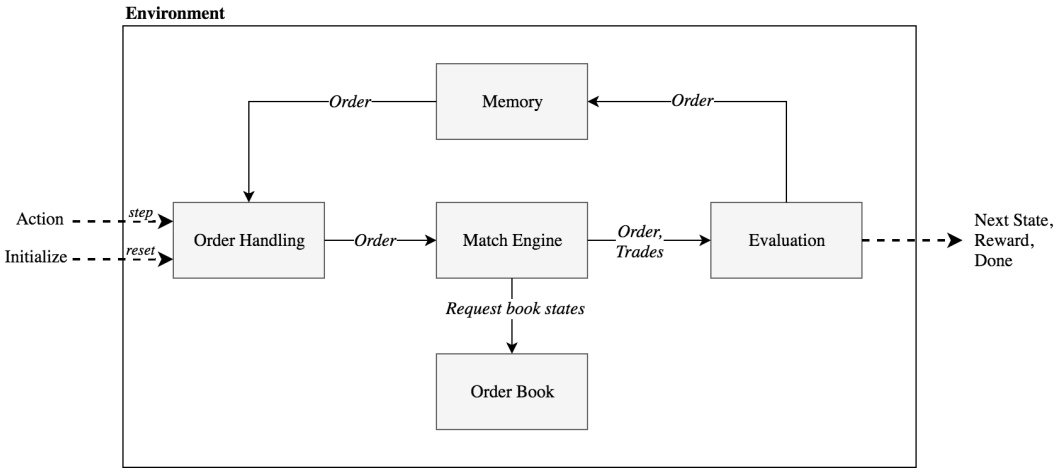


Figure 5.1: Overview of reinforcement learning order placement environment.

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 for the epoch, 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 run the initiated epoch (e.g. placement of the order). The agent explores the environment using the $step$ function which it uses to send the desired action in order to place the order at a certain price level. The first component of the environment to react to the 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 generated 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 horizon of the order in the *memory*. Additionally, the index of the last visited order book state is stored so that, in a subsequent step, the match engine will resume the matching from where it stopped last. If no trades result during the matching process, only the time consumed for the matching process 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 completed trades. If 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 enforce the final state and therefore complete 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 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.

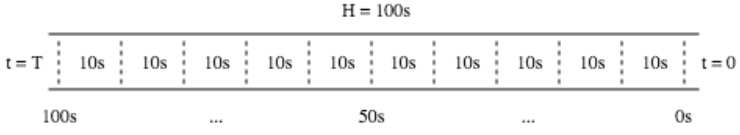


Figure 5.2: Segmented time horizon H with remaining time t.

The time horizon parameter H defines the amount of time allocated 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 commonly seen in a financial markets (see 2.1.2). Furthermore, in this environment, we intend the agents to take discrete steps, rather than continuous steps. The latter would not be applicable in real-world practices where each order has to be submitted to a remote exchange and thereby latency issues occur and API request limitations are applied. 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 $t = 0$ means 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 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* [37]. 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 dimensions 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

The action submitted by the reinforcement learning agent defines 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. More precisely, a_{min} and a_{max} define how deep and how high 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.

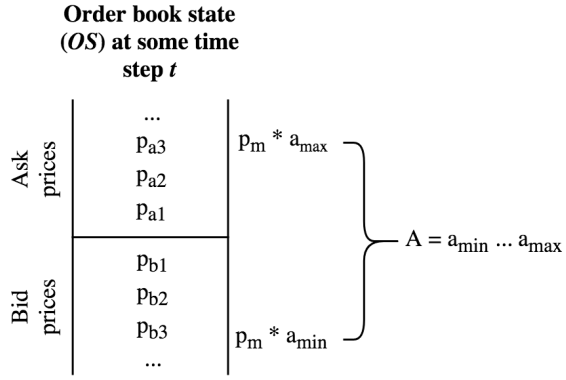


Figure 5.3: Actions represent an offset relative to the order price at which to place the order in some order book state OS_i at some time step t.

By default, the action step size $\Delta a = \$0.10$. For example, with $|A| = 5$ the total action space 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 order prices 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* for the 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, with respect to buying assets, the reward is defined as $r = p_{m^T} - p_{vwap}$ and for selling assets, $r = p_{vwap} - p_{m^T}$. If 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 if 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*[43]. 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 the immediate reward. The aim of the iterative value approach is to estimate the action-value function by using the Bellman equation as an iterative update,

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

Value iteration algorithms then converge to the *optimal action-value* function $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$. [42]

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 [21]. 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 make from the environment is defined by the discrete inventory unit i and time step t, that is, $s = (i,t)$.

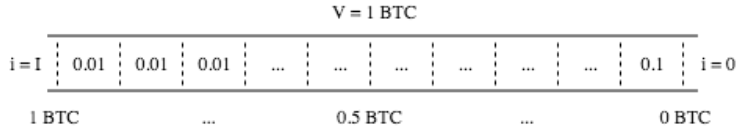


Figure 5.4: Inventory of V segmented shares with a remaining inventory i.

The change in the fractions of the remaining inventory that occurs during matching process would, however, still result in a vast state space. Therefore, the V shares are divided into discrete inventory units i of size Δi, as illustrated in Figure 5.4, and allow to approximate the order size 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

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

Finally, algorithm 1 describes the Q-Learning algorithm used in this work. An adaptation was made to a conventional Q-Learning algorithm[42] regarding the order of the steps the agents will follow; this adaptation therefore makes use of the same concept as presented in [37]. 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 to be transacted at the beginning of an epoch, for each inventory unit i and therefore provides immediate reward (see Section 5.1.5) 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. 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[35]. DQN is a deep reinforcement learning method that combines reinforcement learning with a class of artificial neural network known as

deep neural networks.

> Notably, recent advances in deep neural networks, in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data.[35]

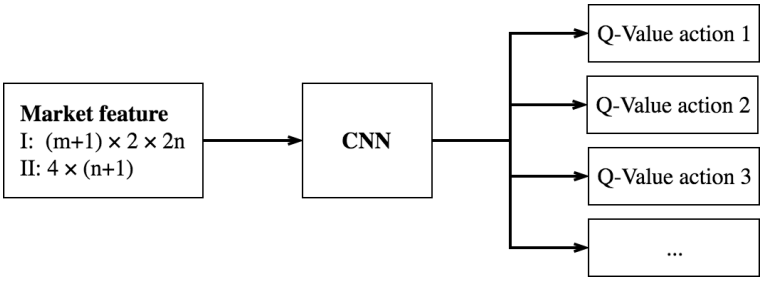


Figure 5.5: CNN forward pass outputs scalar number for each possible action when observing a particular state.

Figure 5.5 indicates that this model generates one output for each action $a \in A$, resulting in all possible Q values for a particular observed state that is either derived from feature I (Section 4.4.1) or feature II (Section 4.4.2). Furthermore, it is valid to note that this is not a classification problem but a regression instead, as the Q-value output is a scalar number.

We use one particularly successful architecture, the deep convolutional network (CNN)[2], which uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields, thereby exploiting the correlations present in trading activity. This model is in line with the architecture described in [34], except that the input shape changed for the data that we used. The input to the neural network is determined by the feature to be used. That is, for feature I (Section 4.4.1): $(m+1) \times 2 \times 2n$, whereas m is the number of order book states plus a window consisting of the inventory i and time left t, and n is the number of bid and ask levels. For feature II (Section 4.4.2), that is: $4 \times (n+1)$, whereas n is the number of historical trades appended to the row consisting of inventory i and time left t. The first and second hidden layers convolve filters that have stride 4 and stride 2 with respect to the input and apply a rectifier nonlinearity [26]. Thereby, we have to ensure that the depth of the filters is equal to the depth of the input. For feature I, we will choose a filter of size $(m+1) \times 5 \times 5$ and for feature II of size $1 \times 5 \times 5$. The final hidden layer is fully connected and consists of 200 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions is therefore defined by $|A|$, which is 201 in the default setup. We refer to convolutional networks trained with our approach as Deep Q-Networks (DQN-CNN).

Even though the CNN is the standard architecture for the widely-used DQN agent setup, and has been shown to perform well in various fields[34, 35], we will provide one additional architecture to be evaluated, namely, a simple multilayer perceptron (MLP)[24] with two hidden layers, each of which consists of 200 neurons. These layers are followed by the output layer which generates one output for each valid action. Therefore, the second architecture processes the samples in the same way as the aforementioned CNN approach, except that no convolution will be applied. As a result, not only will we have a means of comparison alongside an evaluation of the effectiveness of the state-of-the-art DQN agent setup, but we will also have additional confidence while concluding the effectiveness of deep reinforcement learning, when applied to the context of limit order placement. We refer to MLP trained with our approach as Deep Q-Networks (DQN-MLP).

The neural network treats the right-hand side of the above-mentioned Bellman equation (Eq. 5.1), with weights θ, as a target, that is, $r + \gamma \max_{a'} Q^*(s', a', \theta)$. We then minimize the mean squared error (MSE) with stochastic gradient descent,

$$L(\theta_i) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a', \theta_{i-1}) - Q(s, a, \theta_i)]^2 \tag{5.3}$$

Algorithm 2 shows the complete DQN algorithm. While the optimal q-value converges for the Q-Learning approach by using a look-up table, the DQN approach makes use of a non-linear function approximator. However, this 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-batches) 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. Furthermore, 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. Lastly, no preprocessing is being done in $\phi(.)$, as this is handled by the environment (see Section 5.1.3).

Algorithm 2 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode= $1, M$ **do**
    Initialize sequence $s_1 = o_{random}$ with observation and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for t=1,T **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and observation state o_{t+1}
        Set $s_{t+1} = s_t, a_t, o_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set

$$y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q^*(\phi_{j+1}, a', \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to Eq. 5.3

6

Evaluation procedure and discussion of results

In the previous Chapter, we built a reinforcement learning environment with the use of the components which were described earlier, in Chapter 2. The environment facilitates the simulation of order placement on historical order books of the type 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, thereby observing whether or not reinforcement learning is indeed capable of optimizing the placement of limit orders. Therefore, a comprehensive evaluation procedure will be introduced that measures the capabilities of the reinforcement learning agents. Throughout this process, real world historical order books will be drawn upon, as well as artificially created order books, where the latter define distinct 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 will be described. Finally, we will proceed evaluation steps with our agents. Accordingly, this chapter will seek to quantify the effectiveness of deep reinforcement learning and its use of the market features previously constructed.

6.1. Explanation of the evaluation procedure

This section explains the evaluation procedure that will be elaborated in the following sections of this chapter. From our analysis of the same, we will aim to formulate a statement that expresses the capabilities of optimizing limit order placement with reinforcement learning and the use of raw market data. The evaluation steps will proceed chronologically, as follows:

Empirical investigation: Section 6.3 investigates the reinforcement learning environment empirically by simulating an agent's 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 can expect the reinforcement learners to perform. More precisely, we determine the *expected returns* for (1) the optimally chosen limit order and (2) an immediate purchase or sale by using a market order.

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

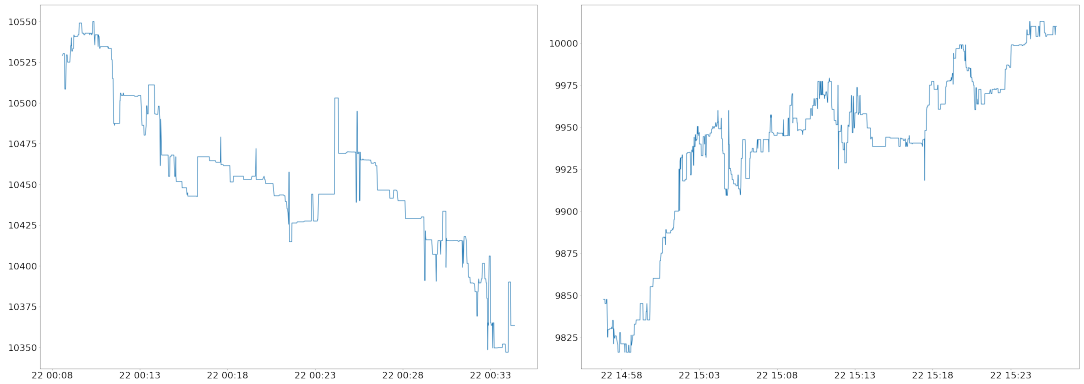
DQN agent policy: Section 6.5 applies market variables to the DQN agent. Hereby, we will make use of Feature I: the price and size of historical orders as described in Chapter 4 (Section 4.4.1); as well as Feature II: the price and size of historical trades (described in Section 4.4.2). As in the previous evaluation step, we will find the average rewards produced by the agent. More precisely, we determine the average rewards for 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. We will investigate the actions selected by the agent in order to determine the agent's limitations. In addition, we apply the agent to an environment which is equipped with an artificially generated order book and will enable us to determine to which extent the agent is able to learn from certain price trends. The results achieved in this evaluation step are (1) *the discovery* of situations where the DQN agent does not perform as expected and (2) the *average reward* achieved by using order books that follow an artificially created trend (downwards and sine curve).

With the results obtained throughout this evaluation, we will be able to determine and quantify the extent to which deep reinforcement learning can optimize limit order placement; we will also be able to 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 for the experiments in this chapter. One of the reasons for choosing two very different data sets is to determine the ability of the learners to react to a variety of market situations. In addition, as explained in the following section, the expected return of a learner for buying and selling assets heavily depends on the market price movements and therefore the behaviour is expected to be different for the data sets in use. More precisely, *data set I*, as shown in Figure 6.1a), is a downward trend (indicated by the bid/ask mid-price) and consists of 1132 order book states with an overall duration of 1681.8 seconds, resulting in 0.67 states per second. In contrast, *data set II*, as shown in Figure 6.1b, consists of 1469 order book states with an overall 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 canceled 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. Although more than two data sets would produce a more generalized outcome of this evaluation, this was computationally not feasible within this work.



(a) 30 minute downwards trend

(b) 30 minute upwards trend

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

As 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 the 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. In this way, the agent can simulate the placement of limit orders in such a way that, within the given time horizon H, the inventory can be either bought or sold. For each *epoch* an agent processes, 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.

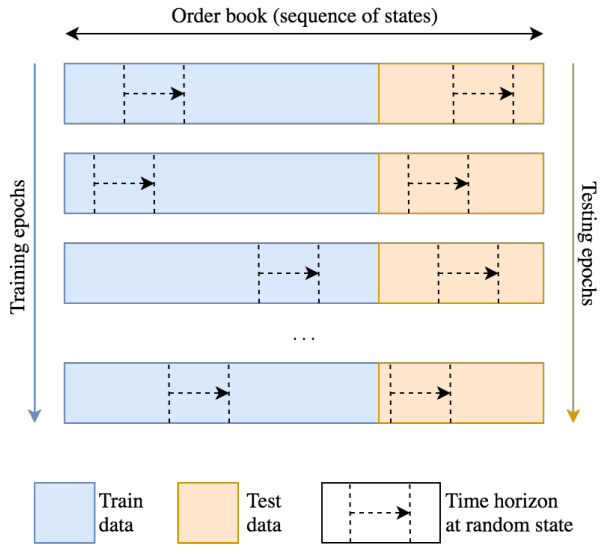


Figure 6.2: Order placement training and testing on an order book data set.

Figure 6.2 illustrates this process. A randomly-chosen order book state will define the beginning of the time horizon and the set of order book states that will fall into this window. This is very crucial since the states within this time horizon and the set of states, not only will 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 time stamp difference of Δt) is considered by the match engine, as explained in the previous chapter in Section 5.1.2. This process is identical for testing, except that the underlying data is different and the agent will not learn from the epochs to be proceeded during testing and instead will report the achieved rewards.

6.3. An empirical investigation of the reinforcement learning environment

In this section, the relationship between the limit order placement and the received return will be investigated. This method was demonstrated in the related work Section 3.1 and was taken to empirically evaluate the reinforcement learning environment. Therefore, we will simulate an agent that submits actions in order to buy and sell shares at every possible limit level and records the immediate returns it receives. A return is defined as the difference between the market price prior to the order placement and the volume- weighted average price (VWAP) paid or received, as stated in Eq. 5.1.5. Hence, we will gain an understanding of the estimated rewards of limit order placement in the historical data set in use. In addition, these results will set a benchmark for the reinforcement learners to come.

We will now describe the setup of this investigation. We will investigate the rewards of limit orders placed on progressively increasing time horizons, from 10 seconds to 100 seconds, and thereby observe the importance of the action chosen by the agents in order to buy or sell assets, in accordance with the length of the time horizon. For each time horizon, we will place (e.g. cross-validate) 100 orders of size 1.0 BTC at the beginning of this 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 will then be derived from the average of the received returns of these 100 orders. This process will be repeated across a range of 201 actions A that correspond to the limit levels $-100...100$ with step size $\Delta a = \$0.10$, resulting in orders priced in the range of $p_m - 10 \ldots p_m + 10$, whereas p_m is the market price before the order was placed. The limit levels will be chosen broadly in order to retrieve understanding about the outcome of a variety of possible actions. As a result, a total of 20,100 orders will be submitted for each time horizon defined. This procedure will be undertaken for both data sets I and II.

6.3.1. Order placement behavior on data set I

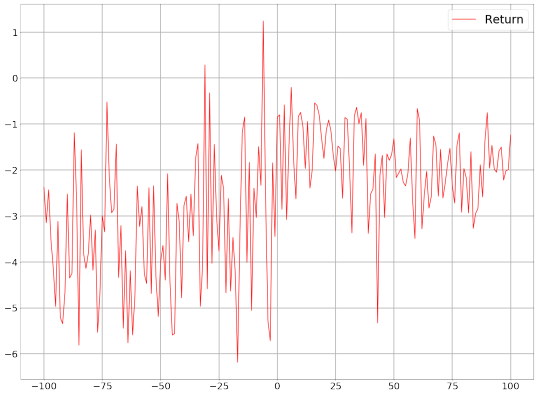
For data set I, where the market sees a downwards trend, the assumption is as follows: We expect buy orders to result in better returns when placed deep in the order book, in other words, on orders that have a highly negative limit level ($a < 0$). Since the price tends to fall, the assumption is that an agent is able to buy at a lower price as time passes. Therefore, the longer the time horizon, the lower the limit level that can still be chosen

in order to execute the full amount of shares. In contrast, we expect sell orders to provide better returns when the agent crosses the spread with a positive limit level ($a > 0$). The assumption is that, in a falling market, it is unlikely that market participants are willing to buy at 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, that is submitted if 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. The x-axis indicates the placement of the order at limit levels ranging from $a = -100$ to $a = +100$ and the y-axis indicates the average return received.

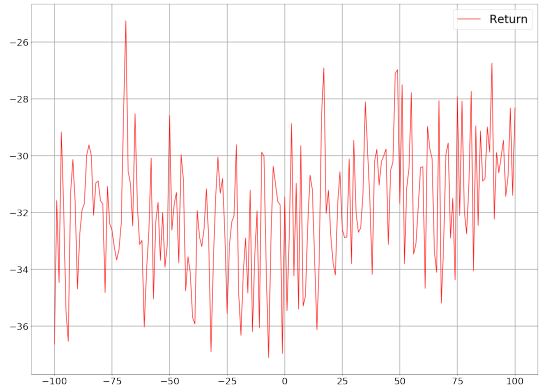
With a time horizon of only 10 seconds, the expected behavior is, however, proven wrong. For buy orders, shown in Figure 6.3a, the returns suggest that orders be placed close to the spread, but still on the opposing side ($a =\sim 5$). The spike at limit level $a =\sim -5$ indicates that the overall best return was produced at this level. However, this comes with the risk that the orders fail to execute, which is indicated by the downward dip 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 with $a =\sim +50$.

With an increased time horizon of a total of 30 seconds, as shown in Figures 6.3c and 6.3d, the expected behavior becomes more apparent. Positive returns can be achieved by posting buy orders deep in the order book. Therefore, we can expect that in this market situation, an agent would be able to partially execute the order at very low limit levels and, for the unexecuted part, a market order would follow. The densest range of positive returns can be seen around the limit levels just below the spread. Orders placed deeper in the book oftentimes result 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. This is a result of the agents' willingness to immediately buy at an increased price. The opposite effect occurs while selling assets. Market orders higher in the book 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, were rewarded better than the ones close to the spread. The most likely reasons it that a minority of orders were partially filled at this level during the cross-validation process.

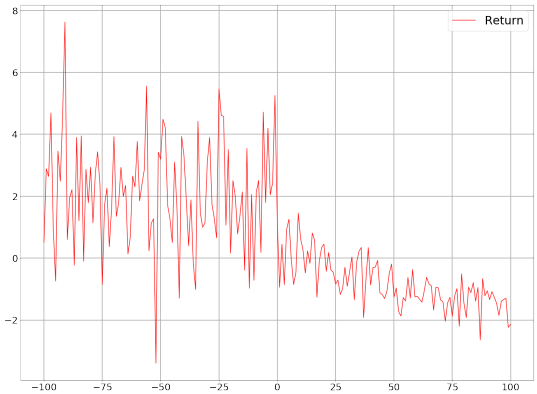
With time horizons of 60 and 100 seconds, the expected behavior of the orders is clearly apparent. Buy orders, as shown in Figures 6.3e and 6.3g, achieve highest returns when placed very deep in the order book. However, when placed at levels -100, the returns are slightly lower as a result of unexecuted orders which had to be completed with a following market orders. In addition, positive limit levels become stable at these price levels since there are more sellers in the market with the extended time horizon. Therefore, very highly placed orders result in the same return as limit orders posted only slightly above the spread. Furthermore, placing orders very deep in the book has the same effect as when placing them just below the spread; that is, there are no traders willing to buy at such a high price and therefore market orders follow once time has passed.



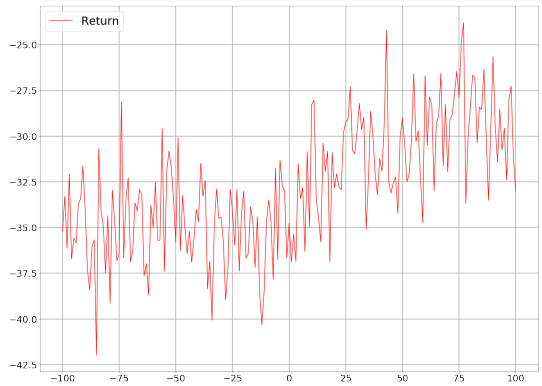
(a) Returns of buy orders within 10 seconds



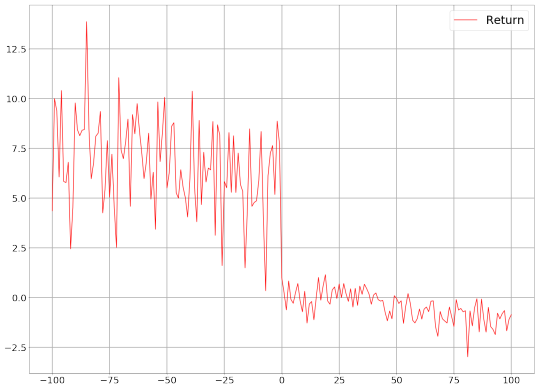
(b) Returns of sell orders within 10 seconds



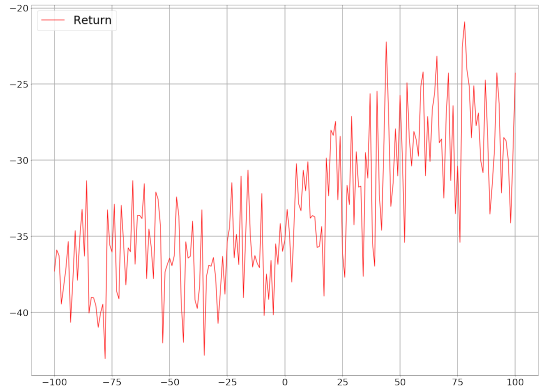
(c) Returns of buy orders within 30 seconds



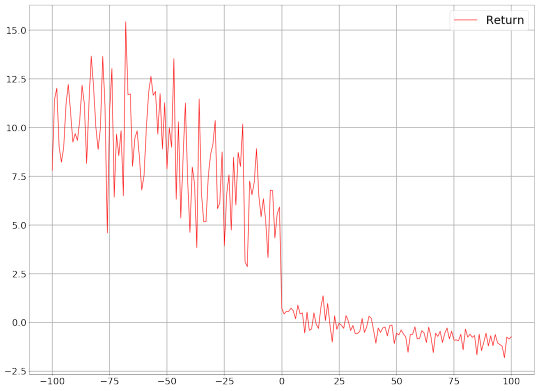
(d) Returns of sell orders within 30 seconds



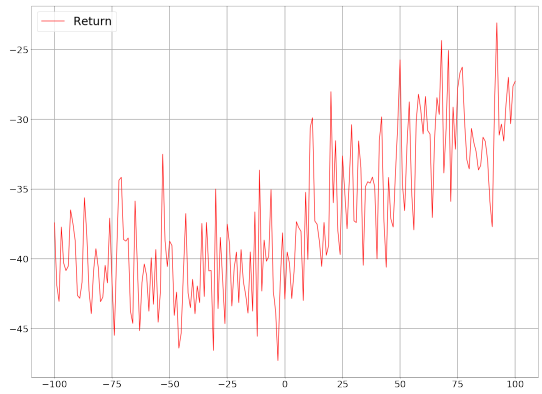
(e) Returns of buy orders within 60 seconds



(f) Returns of sell orders 60 seconds

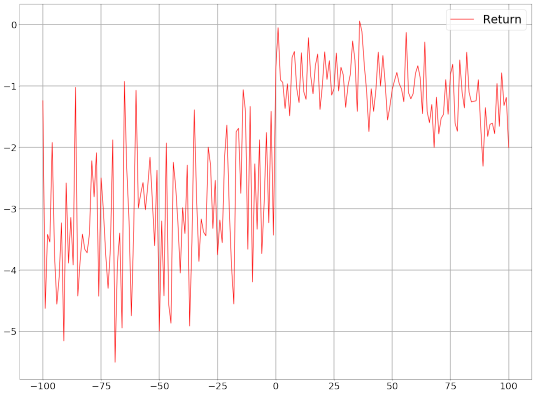


(g) Returns of buy orders 100 seconds

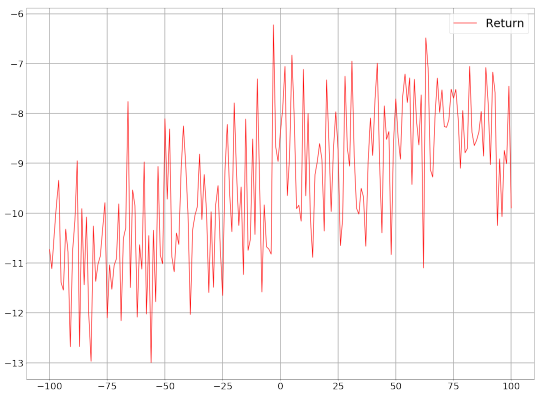


(h) Returns of sell orders 100 seconds

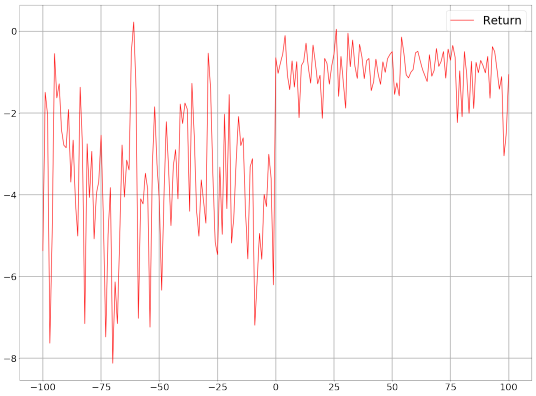
Figure 6.3: Returns of buy and sell orders executed within 10, 30, 60 and 100 seconds on data set I.



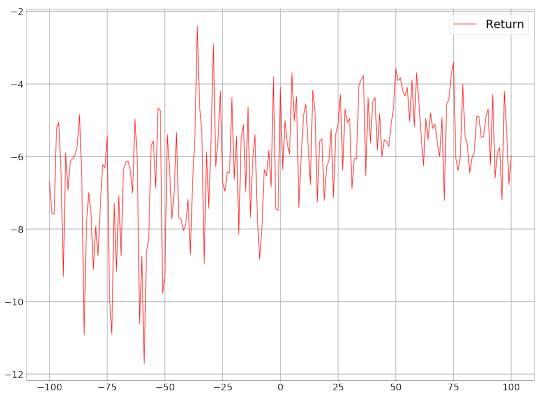
(a) Returns of buy orders within 10 seconds



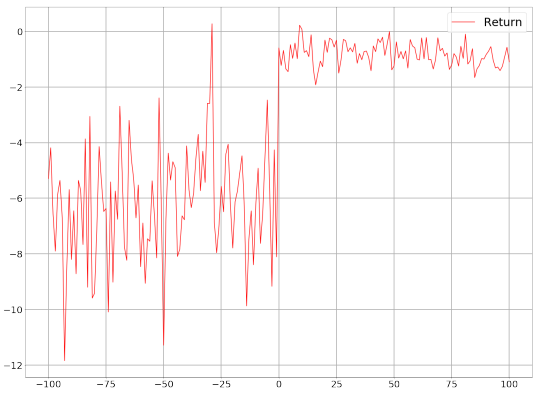
(b) Returns of sell orders within 10 seconds



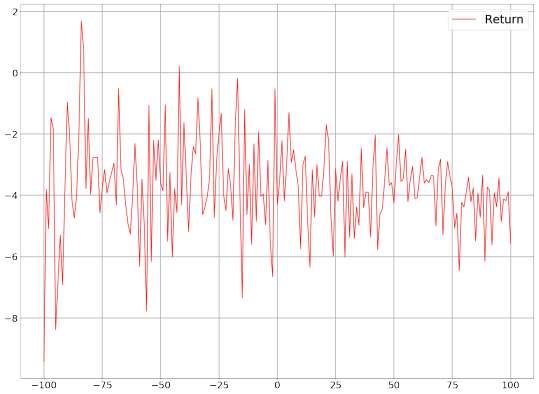
(c) Returns of buy orders within 30 seconds



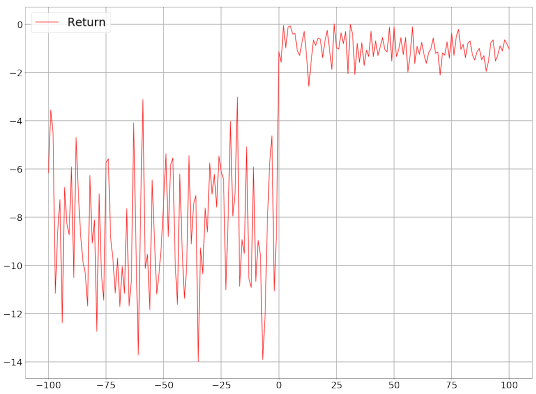
(d) Returns of sell orders within 30 seconds



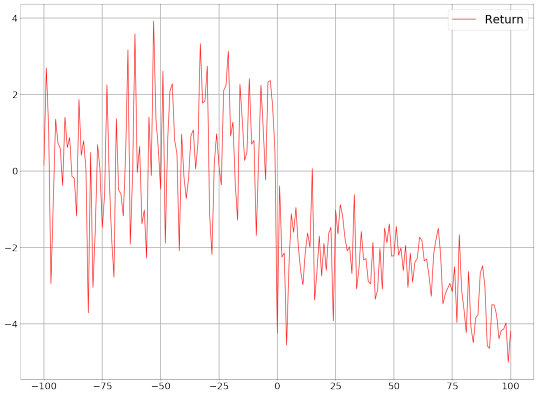
(e) Returns of buy orders within 60 seconds



(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 behavior on data set II

For data set II, which shows an upward price trend, the assumption is the opposite as stated 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 ($a > 0$). The assumption is that, as time passes and the market price rises, other traders become less willing to sell at the market price or lower. Therefore, the longer the time horizon given to the agent, the more critical it becomes to execute immediately; otherwise, shares would have to be bought at an increased market price. In contrast, better returns with sell orders are expected when placed deep in the book ($a < 0$), meaning that they are sold at a higher price. The assumption is that, as the price rises, market participants become more likely to buy assets at higher prices. Hence, the longer the time horizon, the deeper in the book the agent should place a limit sell order. We investigated these assumptions by performing the same experiment as in the previous section, with time horizons of 10, 30, 60 and 100 seconds, as shown in Figure 6.4.

The returns of buy orders filled within a time horizon of 10 seconds, as shown in Figure 6.4a, correlate with the above-mentioned 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 obtains the highest rewards when choosing a price for the order at market price $a = 0$. A highly negative limit level yields a return of approximately $3.00 less than when placing at the suggested market price.

With 30 seconds left to buy 1.0 BTC (Figure 6.4c), the orders placed with $a > 0$ (above the spread) become stable for any such limit level. This is likely due to the higher order pressure of data set II, as described in Section 6.2, as there are more market participants willing to sell. The returns for limit sell orders, as shown in Figure 6.4d, became more rewarding as the agent benefits from a slight increase in price within the given time horizon.

This pattern becomes clearly apparent when a time horizon of 60 and 100 seconds was given, as shown in Figures 6.4f and 6.4h respectively. With this 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 ($a < 0$). As time passes and the market price rises, market participants are willing to buy at an increased price and the agent is expected to be able to sell all assets at this increased price without the need for a further market order. In contrast, if the agent decides to sell the assets at decreasing prices, as indicated by the higher limit levels (when $a > 0$), a lower reward 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 an action of +100, as opposed to a negative action. Figures 6.4e and 6.4g show the expected results of an agent that buys assets within 60 and 100 seconds respectively. It is evident from the figures that the rise in the market price means that the expected damage can be minimized by crossing the spread and buying immediately. The proposition stated before remains valid: the agent should choose a price slightly 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

From the results of the empirical analysis performed on data sets I and II, the following observations can be made with respect to limit order placement: during a fall in the market price, the purchase prices of assets can be optimized by placing limit orders deep in the order book ($a < 0$) and the least loss is sustained when selling assets immediately at the market price ($a > 0$). In contrast, during a rising market, it is suggested that assets be purchased using a market order ($a > 0$) and sale prices can be optimized by placing the sell orders deep in the order book ($a < 0$). These effects become more apparent when a longer time horizon is given for an order. With regard to shorter time horizons, the order placement process tends to be either intercepted by short term fluctuations or low trading volumes. The latter implies that, during an upwards trend, market participants are willing to buy and sell shares at higher prices and on the contrary, during a downwards trend at lower prices. In this experiment, the orders placed had to be completely filled without the ability to take intermediate steps of canceling and replacing the order within the given time horizon. Therefore, it falls to the reinforcement learners to evaluate whether or not such amendments 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.

	$\mathbb{E}$[**Limit order**] **(optimal)**	$\mathbb{E}$[**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
Σ	-9.88	-30.53

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

Furthermore, we ran the same experiment by using different action step parameters Δa, as shown in Table 6.2. Thereby, we found that a coarser step size does not improve the expected return for an optimal placed limit order. As a result, we will henceforth use the setting $\Delta a = \$0.1$ for investigations undertaken with reinforcement learning agents.

	$\Delta a = \$0.1$	$\Delta a = \$0.2$	$\Delta a = \$0.5$	$\Delta a = \$1.0$
$\mathbb{E}$[**Limit order**] **(optimal)**	-9.88	-9.96	-13.52	-17.65

Table 6.2: Rewards derived from the empirical analysis with different action step parameters Δa.

6.4. Q-Learning without market variables

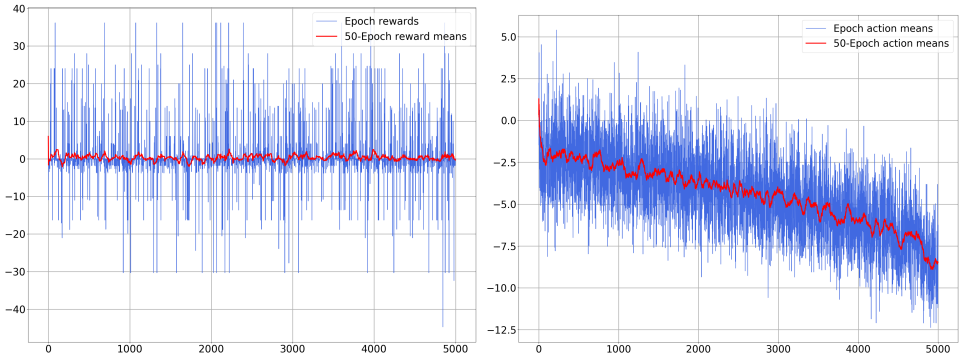
The previous section provided the expected rewards an agent receives when placing buy and sell orders using the reinforcement learning environment under the application of data set I and II. For each observation, a fixed time horizon was chosen for which an orders were placed in the order book, followed by a market orders in case the orders were not filled completely.

This section aims to investigate whether or not a Q-Learning agent, as described in Chapter 5 (Section 5.2), can improve with respect to the rewards received. Thereby, the agent is allowed to cancel its order after every 10 seconds (e.g. $\Delta t = 10$) and place a new order on the basis of the remaining inventory, until the time horizon of 100 seconds expires. Subsequently, and in the event that an order has not been filled completely, a market order is submitted for the remaining share to be bought or sold. For both data sets (I and II), an independent learning experiment was proceeded where the agent had to either buy or sell shares. In each experiment, the training was limited to 5000 epochs and 1000 orders were placed and evaluated on the test set (defined as "backtesting"). The Q-Learning agent was set up as follows: the learning rate selected $\alpha = 0.1$ is small due to the extensive number of steps the agent will take throughout the epochs. The discount factor $\gamma = 0.5$ is chosen to balance the agent's incentive to profit from immediate and future rewards. Initially, the exploration constant ϵ is set to 0.8 and a decay is applied such that the factor reduces to $\epsilon = 0.1$ 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, 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, in which the agent executed orders by choosing from the learned policy.

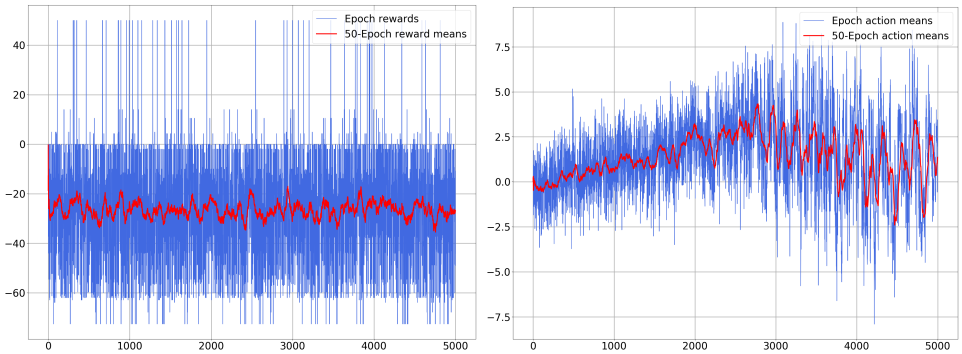
6.4.1. Results of training and testing on data sets I and II

Figure 6.5 shows the training on data set I. The average reward received 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 changing policy. This is 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 were then adjusted to the more negative side of the order book, such that after ~1500 epochs, the agent chose actions as low as -20 and then adjusted and stagnated at ~-15. During the backtest, 1000 orders were executed on the test data set, which resulted in an average reward of *-1.17*. Making a comparison between the results of the empirical analysis performed on the same data set, as shown in Figure 6.3, provides means for interpretation: The policy learned by the Q-Learning agent performed worse than the expected costs of a market order, which was $-1.12 when buying 1.0 BTC; this is shown in Section 6.3. The highly negative average actions the agent choose toward the end of the training indicates that the order might not oftentimes have been able to be filled within the time horizon and an expensive market order had to follow.



(a) Mean rewards per epoch (buy)

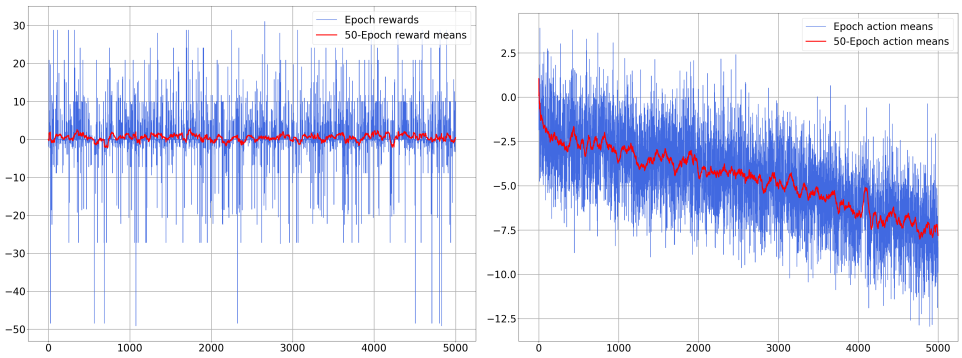
(b) Mean of actions per epoch (buy)



(c) Mean rewards per epoch (sell)

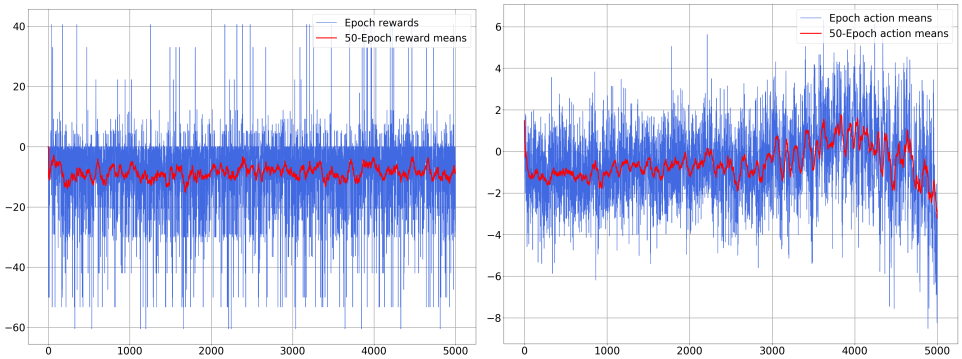
(d) Mean of actions per epoch (sell)

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.

The rewards received for the agents' tasks of selling the assets are much more volatile than for buying, as shown in Figure 6.5c, and no clear improvement can be seen. In addition, no significant adjustment was made by the agent regarding the mean of the actions chosen, as indicated in Figure 6.5d. The backtest resulted in the achievement of an average reward of -21.34 by the agent. The return received for placing market orders on the test set accounted to a negative reward of -27.70. Hence, the agent was able to save $6.36 when selling 1.0 BTC.

Figure 6.6 shows the experiment performed on data set II. The average reward received while training to buy the asset is shown in Figure 6.6a. Throughout the epochs, the agent was able to improve the mean reward by approximately 0.5, which was a similar result to that obtained with the previous data set (I), . 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 test data set II resulted in an average reward of *-1.04 –*, again worse than the average reward received with the training set. A market order on test data set to accounted to an average reward of -1.06, indicating that the agent's policy was saving $0.02 when buying 1.0 BTC. On the basis of the empirical analysis performed when buying on this data set, as shown in Figure 6.4, and comparing it to the rewards received by the agent, implies that the agent failed to execute orders with the limit orders placed and oftentimes market orders were followed.

As with the sell orders placed on data set I, the rewards received from the agent's tasks to sell the assets in 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 actions chosen, 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 use of an requires the payment of a premium of $3.02 when selling 1.0 BTC.

6.4.2. Conclusion of Q-Learning approach

	Q-Learner	$\mathbb{E}$**[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
Σ	-28.29	-30.53

Table 6.3: Summary of rewards for the Q-Learning agent and market orders.

The findings of this section are summarized in Table 6.3. We conclude that the Q-Learning agent was not able to constantly 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 trigger 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 findings were a result of the agent aiming for a reward that was too immediate, the same experiment was performed 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 rewards 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 has provided evidence, which we regard as sufficient, that the chosen actions resulted mostly in market orders. Furthermore, it is to be assumed that the absence of market variables makes it hard for any learner to determine an optimal policy. 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 favor of optimally placing limit orders.

6.5. Deep Q-Network with market features

In the previous section, the Q-Learning agent was trained on data sets I and II, and no significant optimization in terms of the buying and selling of assets was achieved. By considering the above-mentioned results of the empirical analysis of the limit order placement behavior, 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 has elapsed.

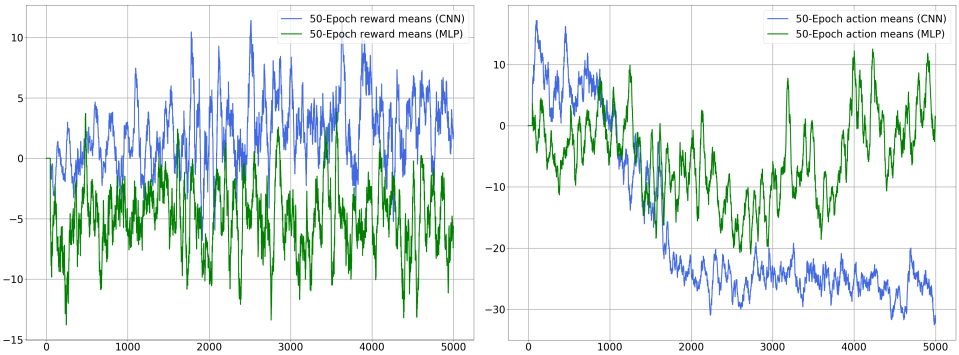
In this section, we aim to determine whether or not the DQN agent is capable of extracting information provided by the raw market features and therefore improve the limit order placement policy. The setup is the same as in the previous section, where the agents task is to buy and sell 1.0 BTC within 100 seconds with discrete step size $\Delta t = 10$ in both data sets I and II. Furthermore, both of the features constructed in Chapter 4, will be applied and investigated separately. The input shape of the model in use that approximates the action-value function, as described in Chapter 2 (Section 2.4.6), is determined by the feature in use and is described below. We evaluate the performance of the DQN agent under the application of both neural network architectures, the standard convolutional neural network and the multilayer perceptron, as presented in Section 5.3. For both of these DQN agent setups, we rely on the default hyperparameters worked out by Mnih et al. [35], as shown in Figure 6.7. Furthermore, we will train both agents with 5000 epochs and will test with 1000 epochs. Finally, we will conclude our findings and determine the capabilities of the DQN approach (and its use of the market features) by comparing it to the expected returns for a market order calculated earlier 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 can be obtained by tuning these hyperparameter values.

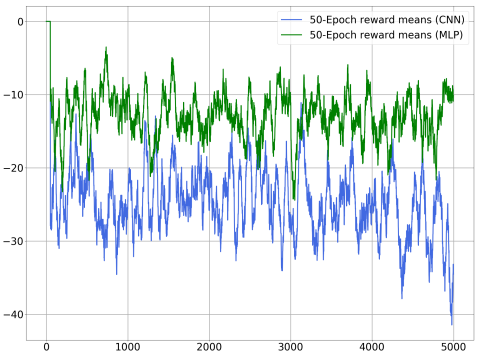
6.5.1. Application of historical order feature

The following evaluation of the DQN agent setups that consider Feature I, as described in Chapter 4 (Section 4.4.1). Furthermore, in this initial run we consider $m = 30$ historical order book states, alongside the maximum number of bid and ask levels $n = 40$. This results in a feature set size of $(60, 40, 2)$, as a consequence of the defined size $(2 * m, n, 2)$. When we included the two private variables by appending a vector $[inventory, time]$ at the beginning of this feature vector, the size of the feature set was $(61, 40, 2)$ and, as such, will serve as the input for the neural networks in use.

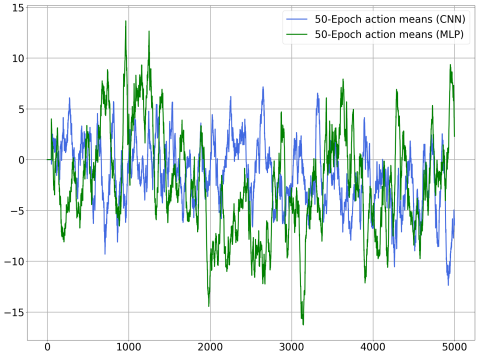


(a) Reward per epoch (buy)

(b) Mean of actions per epoch (buy)

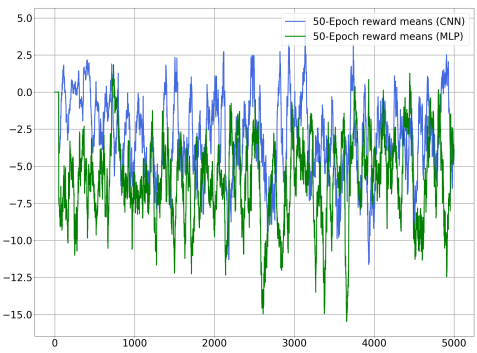


(c) Mean rewards per epoch (sell)

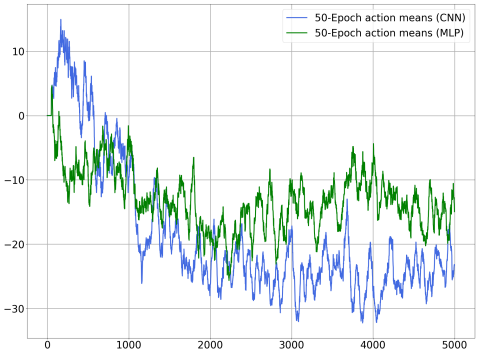


(d) Mean of actions per epoch (sell)

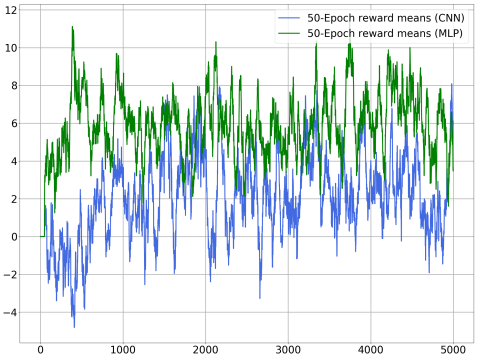
Figure 6.8: DQN agent rewards and mean of actions for buying and selling on training data set I using feature I.



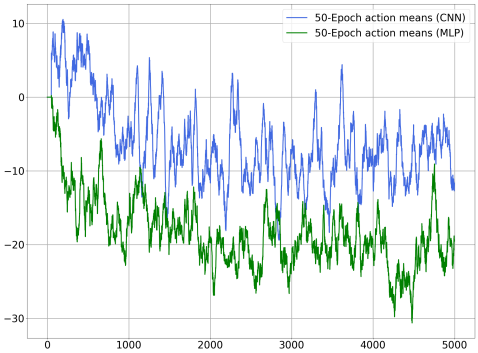
(a) Reward per epoch (buy)



(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 data set I. During the training process relating to the purchase of assets, the rewards obtained consistently improved over the course of the 5000 epochs (Figure 6.8a), throughout which the DQN-CNN agent steadily adjusted its actions to be more negative and the DQN-MLP agent adjusted them to become more positive (Figure 6.8b). It is evident that this adjustment, for both DQN agents, is not as linear as it was during the training of the Q-Learning agent, as shown in the previous section. However, the adjustment made by the DQN-CNN agent is appropriate since the market price was falling. The rewards obtained from selling are shown in Figure 6.8c and the average chosen action throughout the epochs is shown in Figure 6.8d. The rewards did not improve over the course of the training and both agents did not choose to adjust the average actions. Possibly, this was due to the constantly negative rewards obtained during the training under market conditions that were difficult for the sale of assets while the market price was falling. Figure 6.9 shows the agents learning processes using data set II. Over the course of 5000 epochs, the agents were not able to improve their reward and instead, stagnated at approximately $-2.50 and $5.00 rewards per epoch respectively, as shown in Figure 6.9a. Both agents adjusted the actions steadily such that the average limit level fell to approximately -20 at epoch 1000 and remained at this level, as shown in Figure 6.9b. Therefore, the rewards must have been determined by market orders followed after those which failed to fill the order with their levels chosen. The rewards for the agent that learned to sell assets are shown in Figure 6.6c. The rewards could not be improved during the training, even though the agent lowered the average action chosen to below -20, as shown in Figure 6.6d. In such rising market conditions, the rewards observed are a product of the market order, since negative actions will likely not result in a filled order. The rewards for selling assets in this rising market was improved and remained above $0.00, as shown in Figure 6.6c. Thereby, the average of the actions chosen for an epoch reduced within the first 1000 epochs and, thereafter, remained at approximately -10 (DQN-CNN) and -20 (DQN-MLP), which correlates with the rewards obtained .

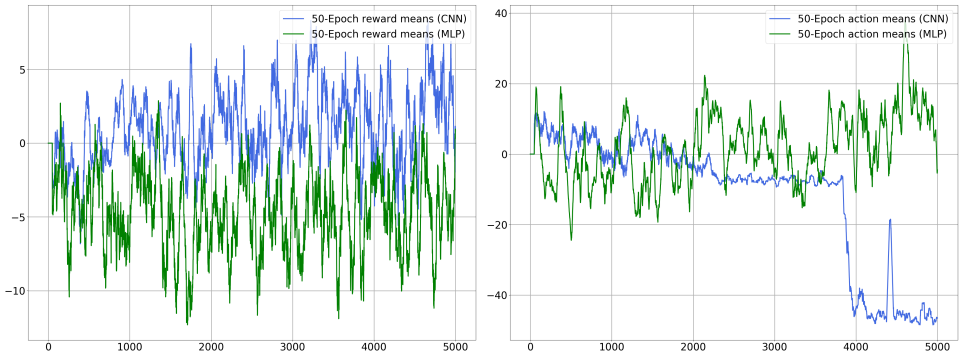
Table 6.4 summarizes the average rewards observed during the backtest using the DQN agents that made use of Feature I. Overall, both DQN agents were able to optimize limit order placement when the given market conditions became favorable to making a purchase or sale respectively. More precisely, significant improvements were made during the backtest of the DQN-CNN agent when the task was to buy assets and the market price was falling (data set I). Thereby, the agent was rewarded with 22.06, which is a significant improvement compared to the expected return of -0.05 for a market order. Likewise, during rising market conditions (data set I), both agents were able to improve by more than the expected market order return. However, when market conditions were not favorable to an intention to either buy or sell, the agents mostly performed not only worse than the expected return of a market order but also worse than the Q-Learning agent (see Table 6.3 above). That is, the DQN-CNN agent achieved rewards of -39.26 and -2.26 when selling in data set I and buying in data set II, which were both worse than the expected returns for market orders. The exception was when the DQN-MLP agent achieved -22.67 when attempting to make sales in data set I and therefore, it performed better than the expected market order return. However, the backtest that set the DQN-MLP agent to make purchases in data set II performed, with -7.40, much worse than the baseline provided by the expected market order return. Overall, the DQN-CNN agent performed the best and minor improvements were achieved by the DQN-MLP agent.

	DQN-CNN	**DQN-MLP**	$\mathbb{E}$**[Market Order]**
Buy (I)	22.06	-4.04	-0.05
Sell (I)	-39.26	-22.67	-27.70
Buy (II)	-2.26	-7.40	-1.06
Sell (II)	0.84	7.47	-1.72
Σ	-18.62	-26.64	-30.53

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

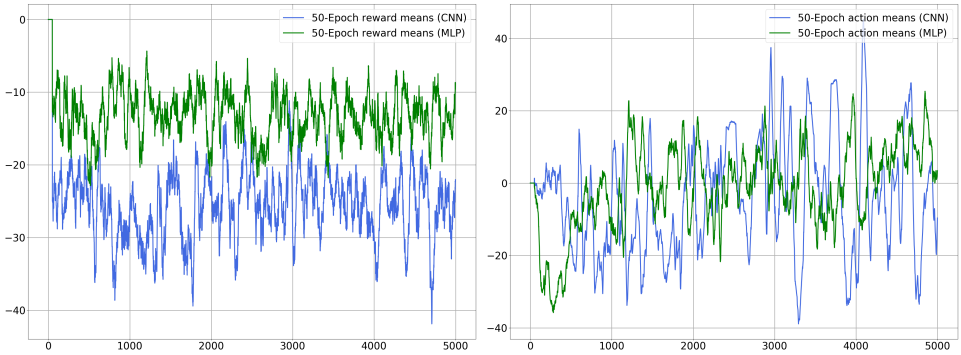
6.5.2. Application of historical trade feature

The following evaluation of the DQN agent setups that consider Feature II, as described in Chapter 4 (Section 4.4.2). In this initial run we consider $n = 30$ historical trades. This results in a feature set size of $(30,3)$, as a consequence of the defined size $(n,4)$. When we included the two private variables by appending a vector $[inventory, time, 0, 0]$ at the beginning of this feature vector, the size of the feature set was $(31,4)$ and, as such, will serve as the input for the neural networks in use.



(a) Reward per epoch (buy)

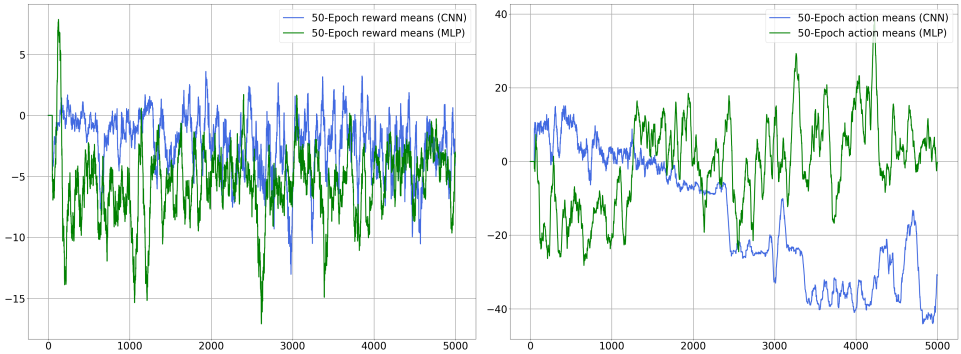
(b) Mean of actions per epoch (buy)



(c) Mean rewards per epoch (sell)

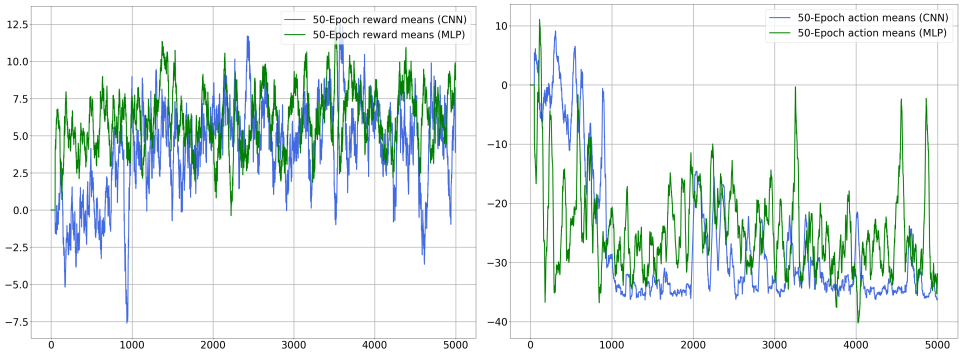
(d) Mean of actions per epoch (sell)

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 processes of the DQN agents when buying and selling in data set I. As with the agents that were provided by Feature I, the rewards received when buying were slightly improved over the course of the 5000 epochs, which is shown in Figure 6.10a. Interestingly, the average action chosen per epoch by the DQN-CNN agent was abruptly adjusted after 4000 epochs, while the DQN-MLP's choice remained between 0 and 20, on average. However, correlating improvements with respect to the rewards were not evident. The rewards received when selling remained just below -10 for the DQN-CNN agent and below -20 for the DQN-MLP agent, as shown in Figure 6.10c. The average of the actions chosen by the DQN-CNN agent, though volatile, were adjusted towards becoming positive, as shown in Figure 6.10d. Unfortunately, this is evidence that the average action does not provide enough insights into the actual decision-making process of the agent. Had the agent consistently chosen positive actions throughout an epoch, an improvement in the reward would have been visible in the aforementioned figure. Instead, the agent must have chosen negative actions in the initial steps and shifted towards very positive actions later in the epoch. 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) for either of the two agents. Interestingly, in this scenario, the DQN-MLP made a significantly better adjustment regarding the actions chosen (Figure 6.11b), indicating its willingness to make more immediate purchases. Rewards for selling in data set II increased during the training, as shown in Figure 6.11c and the average action chosen remained at around -30 after 1000 epochs, for both agents, as shown in Figure 6.11d.

Table 6.5 summarizes the average rewards received by the DQN agents during the backtest. As with the previous application of Feature I to the DQN agents, significant optimization could be achieved when market conditions became more favorable to buying or selling. For buying during falling market conditions (data set I), the DQN-CNN achieved a reward of 31.92 and the DQN-MLP a reward of 25.44, clear improvements compared to the expected market order return of -0.05. Likewise, both agents improved sales in data set II (0.15 and 3.80 reward respectively). Compared to when Feature I was applied, the agents under the application of Feature II achieved better results in difficult market conditions (selling in data set I and buying in data set II). Particularly, sales in data set II were better than the expected returns of market orders. Overall, as indicated by the positive sum of rewards achieved, both DQN agents were able to make purchases and sales of better than the market price. Consequently, the rewards obtained were much better than the expected market order return. The application of Feature II meant that the agent outperformed the DQN agent with Feature I (see Table 6.4 for comparison). Thereby, it is evident that the DQN-CNN setup has yielded greater improvements than the DQN-MLP setup.

	DQN-CNN	**DQN-MLP**	$\mathbb{E}$**[Market Order]**
Buy (I)	31.92	25.44	-0.05
Sell (I)	-25.15	-20.65	-27.70
Buy (II)	-3.56	-6.43	-1.06
Sell (II)	0.15	3.80	-1.72
Σ	3.36	2.16	-30.53

Table 6.5: Summary of rewards during backtest of DQN agent using Feature II (historical trades).

6.5.3. Comparison with different feature sizes

Both features constructed in this project allow for size-related adjustments that determine the information content provided to the agent. Until now, we have made use of one particular setting for each feature, that is, we have used $m_I = 30$ windows with length $n_I = 40$ for feature I and a vector length of $n_{II} = 30$ for feature II. With the use of the DQN-CNN agent, which has resulted in slightly better performance than the DQN-MLP agent, we will investigate whether or not a different size of the feature vector improve these results further. We will proceed with the same evaluation as described in the previous two sections, but with different feature vector settings. For feature I, we will investigate window sizes $m = 10, 20, 30, 40, 50$ with maximum vector length $n = 40$. Similarly, for feature II, we will investigate the length of the vector $n = 10, 20, 30, 40, 50$ (no window present for this feature).

Table 6.6 below shows the results from the backtest and confirms that the setting we ran earlier in this section performed the best. It is evident that a smaller window size for feature I and a smaller vector length for feature II ($m_I < 30$ and $n_{II} < 30$) both have a negative impact on the resulting performance, indicating

that the agent retrieved insufficient information. However, increasing the dimensionality of our feature space ($m_I > 30$ and $n_{II} > 30$) did not improve the agents' capabilities and therefore indicates difficulties arising from the curse of dimensionality[30].

#	**Feature I** $\Sigma[B1, S1, B2, BS2]$	**Feature II** $\Sigma[B1, S1, B2, BS2]$
10	-36.85	-12.36
20	-34.62	-13.68
30	-18.62	3.36
40	-21.59	-11.97
50	-23.37	-1.43

Table 6.6: Summary of rewards during backtest of DQN-CNN agent using different feature sizes.

6.6. Determining the limitations of the DQN agent

This section aims 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 conclude why the agent was not able to choose the most appropriate actions. In addition, artificial order books are being created which serve as new data sets to train and test the DQN agent on. With their aid, 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

Extensive investigations have shown that the DQN agent fails to place orders that lead to constant positive rewards for the following two reasons: 1) when the market situation does not allow the agent to do so and 2) when the agent submits an inappropriate action.

Our investigations have shown that, for the first category, it is not just upwards and downwards trends which make it difficult for the order placement to result in a positive reward, but also when the spread between the best bid and best ask price becomes large. Examples of wide spread scenarios are shown in Figures 6.12 and 6.13, where an agent attempted the placement of a buy and sell order respectively.

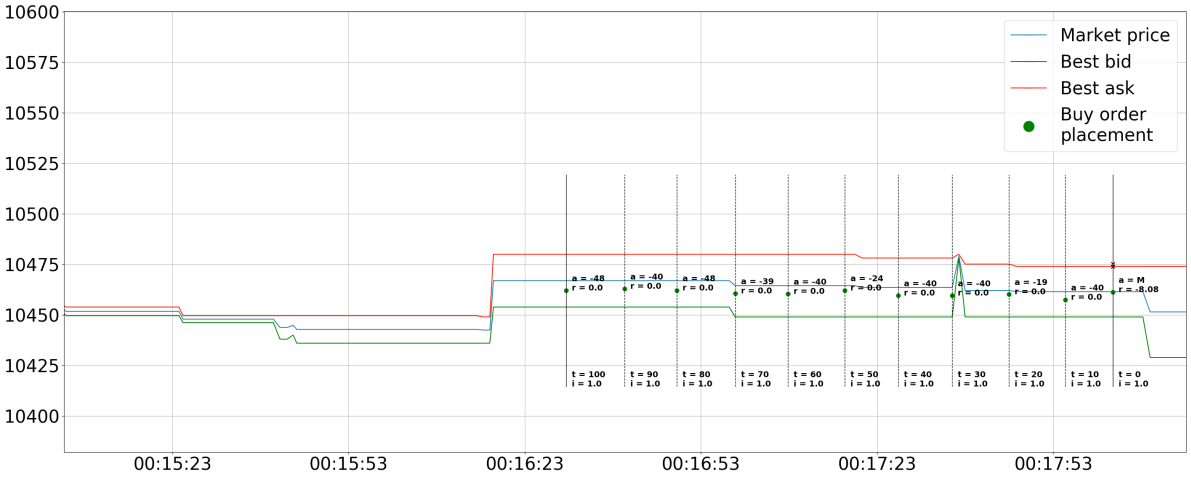


Figure 6.12: Wide spread between bid and ask prevents agent from buying.

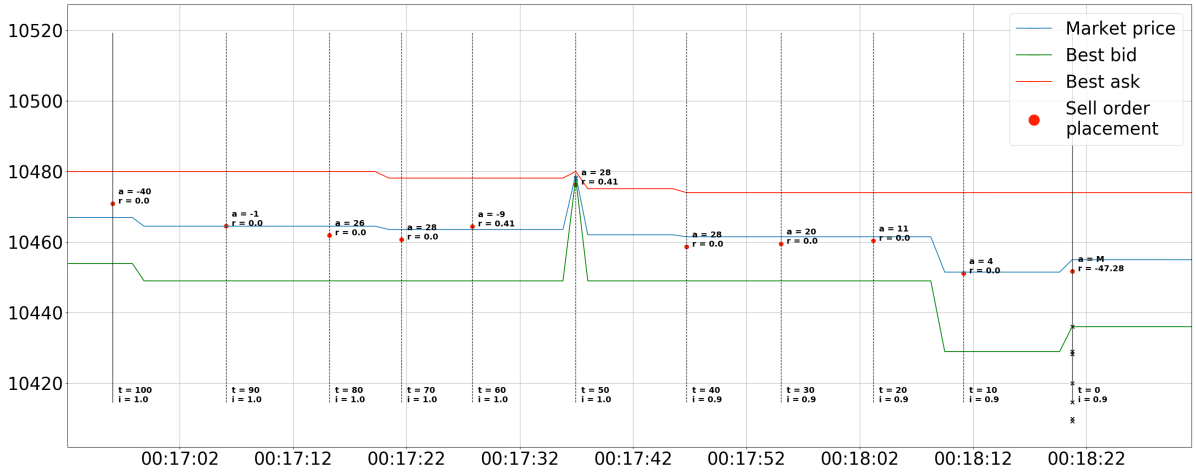


Figure 6.13: Wide spread between bid and ask prevents agent from selling.

Prior to the start of the buy order placement, the spread between the best bid and best ask price was very close, as indicated by the green and red lines. However, during the placement of this order, the spread widened and remained larger than $50.00 for almost the entire time horizon. Since the actions are segmented into discrete steps of $\Delta a = 0.10$, with a total of 101 steps, and a market price ranging from -$5.00 to +$5.00, 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 bought by using a market order at the end of the time horizon (a trade is 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 price level was close to the best bid price only once. As a result, a market order followed in which the agent sold the remaining 0.9 BTC at 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 at decreasing price levels, as indicated by the crosses in the figure. Two possible ways to overcome this limitation would be to either increase the number of steps or to increase the action step size; both approaches will be addressed in Chapter 7.

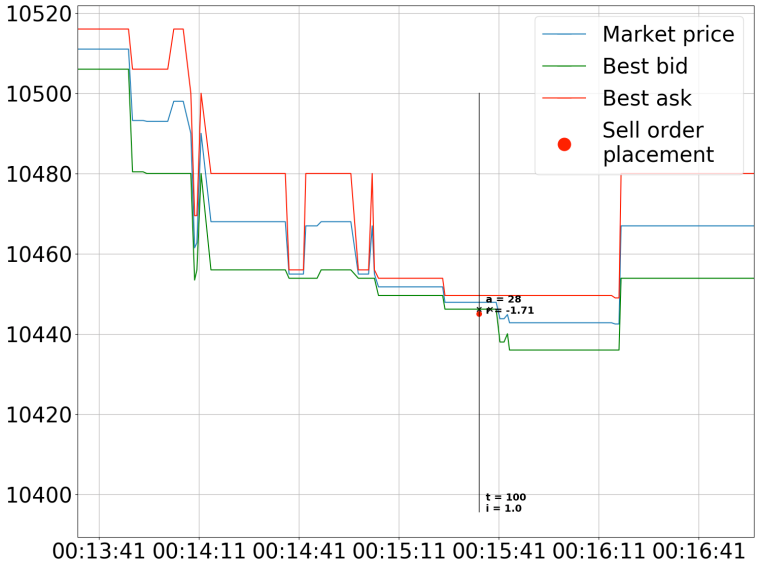


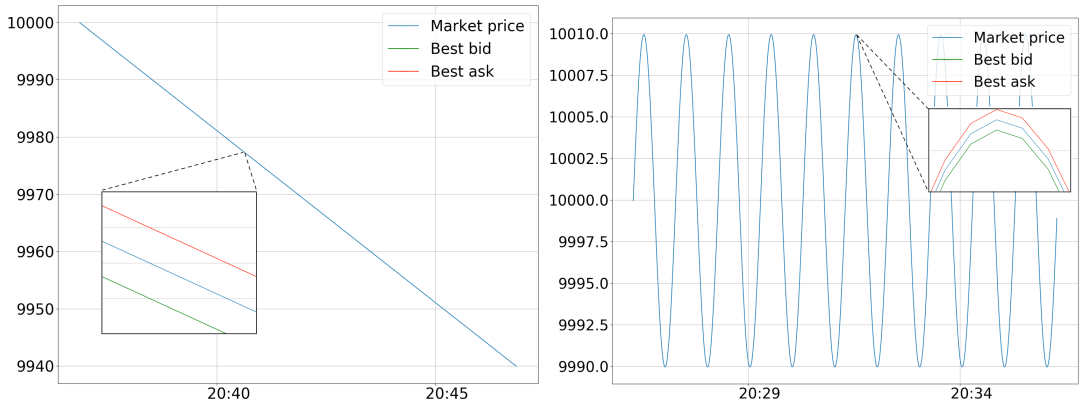
Figure 6.14: Wide spread between bid and ask prevents agent from selling.

An example of the second category, where the agent clearly failed to select an appropriate action, is shown in Figure 6.14. As is illustrated, the market price, and the best bid and ask price were declining before the initialization of the sell order placement. The agent then decided to cross the spread in the first step and choose action 28, which meant that 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

more patient and decided to place a limit order below the spread. Then, in a subsequent step, the order could have been filled by means of a negative action, which would have resulted in a positive reward. It is assumed that the agent failed to generalize the development of the order book with the provided feature set. Instead, the agent reacted to the previous, declining market situation, in which the better choice would certainly have been to immediately fill the sell order.

6.6.2. Capabilities evaluated using artificial limit order books

Introducing an artificially created order book allows us to determine the capabilities of a reinforcement learning agent in greater detail. We define the formation of the order book states over time and therefore can calculate the optimal order placement policy in advance. In doing this, we can investigate whether or not an agent is able to find the optimal placement policy. In addition, by constructing artificial order books, we were able to remove any short-term market fluctuations and therefore provide more consistent rewards to the agent during training. Figure 6.15 below shows the setup of two such order books. Both consist of 10 minutes worth of data with order book states changing every 1 second, resulting in a total of 600 data points that represent the order book states. Every such state consists of 25 levels on both the bid and ask side with a deviation of $0.10 for each level, as shown in the zoomed panes. In addition, on every such level, 1.0 BTC is listed on the bid and ask side, which ensures that an agent can fill the entire order at any chosen limit level. The change in the order book states is then determined by a function.



(a) Linear configuration of order book states with slope $a = -0.1$

(b) Order book states configured according to sine function with $f = 10$

Figure 6.15: Artificial order books with duration of 10 minutes

The first example, as shown in Figure 6.15a, is a linear function which sets the market price to start at $10,000 and end at $9,940. The price therefore falls by $0.1 every second. An agent should therefore realize that, when buying, the actions chosen should be negative ($a < 0$) so that, in each step, a trade is transacted at the end of the time horizon at a much lower price. The invariable reward after 100 seconds, for any state to start the order placement with, should therefore be $10.0. In contrast, the agent should realize that the loss when selling can be minimized by selecting a very positive action that results in an immediate sale. The invariable reward should in this case be -$0.10.

The second example, as shown in Figure 6.15b, applies a sine function accross the order book states. The frequency was set to 10, such that within one minute, at least one complete sine wave was generated with the amplitudes peaking at $10,010.0 and $9,990.0 respectively. An agent is expected to learn to place limit orders according to the low and high peaks for buy and sell orders respectively. The average reward for this example will not serve as an adequate measure of the policy since, depending on the state the agent starts at (that is the point in the amplitude), the optimal rewards can be different. Instead, we relied on the volume-weighted average price directly and therefore expect the agent to buy close to $9,990.0 and sell close to $10,010.0.

Both artificial order book constellations were trained and tested with the DQN agent. The configuration of the agent was chosen equivalent to the setup described in Section 6.5 and a total of 5000 epochs were run in training and 1000 epochs in testing. In the first example, which shows a clear downwards trend, the average reward during the backtest of buying assets resulted in $9.45 and, for selling, in $-0.10. These results indicate

that the agent was able to find a near-optimal policy. The second example shows the multivariate sine wave that represents the order book and the average volume-weighted average price for buying was $9,992.0 and, for selling, $10,007.0. This indicates that the agent was able to improve to an near-optimal policy with regards to the stationary development of the order book.

6.7. Conclusion of the evaluation

The evaluation procedure introduced and performed in this chapter showed that reinforcement learning techniques are indeed suitable for optimizing limit order placement for cryptocurrency markets. The findings are summarized in Table 6.7 First, we empirically estimated the potential improvements that can be achieved by placing orders at the the optimal limit level compared to an immediate sale or purchase using a market order. We showed that, when the development of the market price is favorable to limit order placement, namely when the price is falling while buying or when the price is rising while buying, then the reward for the optimal limit order is expected to be significantly higher. In situations where the market price is never favorable, then no improvement can be achieved and the optimal limit level is to cross the spread (a>0) which declares the order to become a market order that fulfills the demand to buy or sell immediately. Subsequently, the Q-Learning agent was trained and tested without the knowledge of market data but with private variables (inventory and time horizon) only. The average rewards achieved during a backtest have demonstrated that this agent was not able to take advantage of market price movements that were favorable to the order placement process. However, the agent was able to reduce the costs when the market price development was not favorable to either buying or selling assets. In fact, the performance of the Q-Learning agent was the best of all considered agents, when it came to reducing the occurrence of unavoidable losses. Subsequently, the DQN agent was evaluated under the independent application of two feature sets as well as two different types of action-value function approximators. It has been shown that the application of feature II (a sequence of historical trades) results in a better policy than the application of feature I (a window of historical order book states). Although the DQN was able to learn a policy that optimizes order placement when market conditions become favorable to buying or selling, the agent performed worse than the expected market order when conditions were not ideal. Nevertheless, the DQN has ultimately achieved a performance which demonstrates the capability of placing and filling limit orders to a better price than what is offered at the market. We have taken the better performing DQN setup, the DQN-CNN agent, and proceed further investigations. Thereby we have shown that low dimensionality of the feature vectors have negative impact on the learning process, and so does a feature with very large dimensionality. Furthermore, in order to understand why no policy could be learned that would constantly result in rewards better than the ones expected from using a market order, the actions taken by the DQN agent were analyzed. It has been shown that 1) a wide spread between the best bid and ask price and 2) provided input samples of the market features prevented the agent from choosing actions which would fill an order at a better price than the market price. Furthermore, artificially created order books were created and features derived therefrom. It was shown that, in a scenario where short-term market fluctuations are absent and liquidity on the buyer and seller side is constantly provided, a deep reinforcement learning agent is able to find a near-optimal placement policy.

	$\mathbb{E}$[**Market Order**]	$\mathbb{E}$[**Limit order**] **(optimal)**	**Q-Learning**	**DQN-CNN (Feature I)**	**DQN-CNN (Feature II)**
Buy (I)	-0.05	15.20	-1.17	22.06	31.92
Sell (I)	-27.70	-27.70	-21.34	-39.26	-25.15
Buy (II)	-1.06	-1.06	-1.04	-2.26	-3.56
Sell (II)	-1.72	3.38	-4.74	0.84	0.15
Σ	-30.53	-9.88	-28.29	-18.62	3.36

Table 6.7: Summary of expected and achieved average rewards from empirical evaluations and reinforcement learning applications.

7

General conclusions and discussion

This chapter discusses and revisits the research questions with respect to the corresponding contributions made and thereby mentions strengths and limitations of this work. Furthermore, a summary of the contributions made is provided. Finally, future work is suggested and a motivation is stated which supports the application of the techniques developed to be used in real world practices.

7.1. Summary of contributions

This work aims to take a step towards answering the important question of how one can optimize limit order placement on cryptocurrency exchanges using deep reinforcement learning. Previous work in this field, by Kearns et al., who studied the behavior of order placement and order execution[36] and developed a reinforcement learning strategy[37] for the purposes of optimization on traditional exchanges, has been extended and applied to the cryptocurrency field.

In this work, raw market data of the Bitcoin/US dollar trading pair was analyzed with regard to the activity of market participants. We have found that patterns emerge while traders create and cancel orders in financial markets. Based on these findings, two features where designed which incorporate these patterns, in the form of a window of 1) historical order book states and 2) historical trades. Furthermore, we developed a limit order book data structure and a simplified match engine; the latter emulates a local broker and can match orders using the historical order book. These mechanisms were incorporated into a reinforcement learning environment which enables the order placement problem to be transposed into the reinforcement learning context. This environment is configurable and therefore flexible enough to enable investigations by means of the previously constructed features as well as a variety of agents. In addition, two implementations of reinforcement learning agents were developed: a Q-Learning agent, which serves as the learner when no market variables are provided, and a Deep Q-Network agent which was developed to handle the features previously mentioned.

Prior to the experiments with the environment and using the agents developed, a comprehensive evaluation procedure was elaborated. This multi-step procedure involved an empirical investigation of the expected costs of limit and market orders and serves as a benchmark for the Q-Learning agent as well as the DQN agent. At the same time, the empirical investigation allowed us to improve the parameter setting of the reinforcement learning environment prior the more costly experiments that involved agents. The results of a backtest have shown that the Q-Learning agent was not able to take advantage of market price movements that were favorable to the order placement process. However, the Q-Learner was able to reduce the costs when the market price development was not favorable to either buying or selling assets. The DQN agent was backtested under the application of two different neural network architectures as well as two aforementioned feature settings. Thereby, we have found that the DQN-CNN agent (convolutional neural network), that is widely used in many fields of literature, is superior to the DQN-MLP agent (multilayer perceptron). Furthermore, we have achieved significantly better performance by using feature I (historical trades) compared to feature (II) historical order book states. Our experiments have shown that a feature vector containing the 30 most recent historical trades performed best among a range of settings tested. The final outcome was that the policy learned was able to optimize order placement when market conditions were favorable to making a purchase or sale. In addition, when conditions were not ideal, the agent still performed better than the

expected costs of a market order, improving the purchase and sale of 1.0 Bitcoin by $33.89. Moreover, we observed the decision-making process of the agent and found further limitations in our setup, which suggest that, with further tuning, our results could improve. Furthermore, it has been shown that the DQN agent is capable of finding a near-optimal policy on artificially generated market data.

Our work confirms that there is indeed a way in which deep reinforcement learning is capable of optimizing the placement of limit orders. However, under the application of real world market data, the agent can be exposed to severe conditions in the form of wide spreads or an absence of liquidity, both of which prevent the agent from applying an optimal policy. Therefore, and in order to constantly be able to perform placements that are better than or equal to what is offered on financial markets at any given time, the approaches presented in this work have to be extended.

7.2. Findings with regard to the research questions

In Chapter 1 (Section 1.2), the purpose of this thesis and the research objectives were stated. This section aims to revisit these research questions, conclude what has been achieved in respect of each of them, and then summarize and discuss the findings in greater detail.

7.2.1. 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?

Historical market event data was investigated in Chapter 4. Thereby, 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 movements were 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. Furthermore, hypotheses were formed based on these findings and two features were constructed that incorporate emerging data containing the patterns found. Namely, a window of historical limit order book states (feature I) and historical trade events (feature II).

The first attempt to find patterns in market data was made by investigating the relationship between trade activity (volume) as well as price change and volatility, as proposed by Karpoff et al. [29]. As traders submit limit orders to buy (sell), they impact the bid (ask) volumes of the limit order book. This gives us a view of traders' intentions and enables us to foresee the direction of the upcoming price changes. To quantify these intentions, we looked at the differences between the bid and ask volume, called order imbalance, as shown in Section 4.3.2 (Table 4.1). Even though some imbalance was clearly present, the high volatility in the Bitcoin/USD market prevented us from finding obvious an correlation with the direction of future price movements. This gave reasons to believe that the data had to be investigated at the raw market event level[28]. The visualization on a volume map of market events occurring over time enabled the detection and association of patterns to future price changes (Section 4.3.3). It was found that orders submitted and canceled, as well as the trades generated, had an impact on the future price movement. More precisely, Figure 4.4 and 4.5 showed examples of a patterns that we have classified, with references to [17, 41] as buy- or sell-walls and consecutive posting- or cancelling. Moreover, consecutive or impulsive trades occurred at the market were detected in Figure 4.6 and their impact on the market price was shown in Figure 4.7. However, in order to determine whether or not this information improves the optimization of limit order placement, a feature had to be constructed that enables an agent to attempt to learn a policy. Hand-crafting the features would have mean to measuring the change in volume and price with for each order and trade event with respect to the previous event occurred at the market, and thereby ensure to include important information such as the frequency at which the orders and trades were posted. Instead, we have chosen an approach that was demonstrated to be successful in other fields. Recent advances in deep reinforcement learning have demonstrated the ability to learn directly from pixel frames[34] and raw sensory data[35] occurring over time. We followed the same principles and constructed features which reflect (1) the order book states and (2) the trades that occurred over time. Feature I, as described in Section 4.4.1, is a vector that contains the most recent order book state and reflects the historical evolution of this state with a window m. Feature II, as described in Section 4.4.2, is a vector that lists the n most recent historical trades including the frequency of their occurrence. As a result, the features constructed will contain new information as soon as a new market event is published; at that point, a new sample can be added to the feature set without the need for a preprocessing step. One of the difficulties with the features constructed was determining the appropriate amount of historical data to

be considered. Attempts were made by measuring the entropy and correlation of order prices and sizes but ultimately, this did not give clear indications of how many historical order book states or trades should be considered. Subsequently, we evaluated the various feature settings during the learning process (see Section 6.5.3) and were able to find the desired settings. It was shown that for feature I the window size $m = 30$ performed best among the sizes 10, 20, 30, 40, and 50. Likewise, improvements with respect to the outcome of the placed orders could be achieved by considering up to $n = 30$ historical trades with feature II.

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

The challenging task of mapping the order placement problem into the reinforcement learning context demanded an investigation of components involved in the financial setting as well as in the reinforcement learning setting; this was described in Chapter 2. In Chapter 5, these components were combined and a reinforcement learning environment was built (see Figure 5.1). This environment configures a discrete action space that enables an agent to explore the state space that is determined by the underlying historical market data. The reward function used represents the outcome of the matching process of an order submitted. We were able to assign the most complex tasks to the environment and thereby drastically reduce the level of complexity of the agents. Two agents with the purposes of learning with and without market features were developed and will be discussed below.

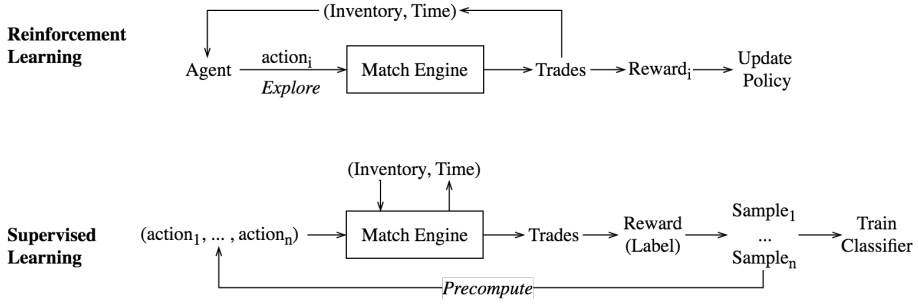


Figure 7.1: Illustration of suitability of reinforcement learning and supervised learning.

We will first reflect on why reinforcement learning is an appropriate approach to be taken in order to optimize the limit order placement problem. Literature reports the uses of supervised learning methods (Section 3.3) to forecast price movements and suggests the application of limit order placement as a separate task. We describe the reason for the separation of these tasks with the supporting Figure 7.1 above. The outcome of an order placement is always determined by the match engine that makes use of the underlying market data. In our case that is a local setup with historical market data; when applied in a real-world setup the match engine would be handled by the exchange where the live market data is determined by the ongoing trading activities. Therefore, multiple issues arise with a supervised learning approach. First of all, in order to respond to changing market conditions, the placement of a limit order involves sequential decisions making (cancelling and re-submitting the order). In supervised learning this implies to predict a sequence of actions (prices at which to place the order) by a classifier [11]. In our context, two problems would arise with such an approach: a match engine (or an exchange in the real world setup) is always laid out to process only one order at a time and therefore an additional scheduling mechanism would be required to handle the submission of a sequence of orders. The second difficulty is related to the fact that the rewards (labels) are not known prior the submission of an order. Therefore, one would have to generate a sample data set by first precomputing a number of sequences of actions, and second, processing their outcome using the match engine. This process is not only computationally expensive but would also require a method to reduce the vast number of combinations possible. In contrast, with reinforcement learning, it is possible to model an end-to-end learning process that is known to be successful in other domains[15, 34, 35]. As shown in the Figure above, this process becomes not only simpler but also uses the given functionality of a match engine and therefore makes it applicable in a real world environment. The steps involved are as follows: one order is submitted at a time by the agent. The order is then updated according to the action chosen as well as the time and inventory remaining, with each step taken. As the agent proceeds with its exploration and exploitation, it is able to reduce the combinations of actions to be taken by a great extent and learns a policy (i.e. an order placement strategy) by receiving immediate and future rewards for each step taken.

A successful application of reinforcement learning in the context of limit order placement is known in literature[37] and was introduced earlier, in Section 3.4. While one of the goals of our work was to make use of raw market data, the authors' approach made use of hand-crafted features; therefore, the only solution available to us was to use an action-value function approximator (see Section 2.4.6). This in turn demanded the application of our own reinforcement learning framework.

Many difficulties arose during the construction of the reinforcement learning environment and important findings were made. The order book (Section 2.1) and the match engine (Section 2.2) proved to be the key components which ultimately enabled a local broker to be emulated and therefore also to facilitate the simulation of limit order placement using historical order books. The difficulties that arose during the development of these components were related to performance and the inevitable approximations required during the matching process. The vast number of market events led the matching process to be become unbearably slow. Improvements were made by using caching and indexing mechanisms but slow speed remains a limitation of this project. The run-time for an agent to train over 5000 epochs took more than 4 hours. Ultimately, this limited the scope of the evaluation that was undertaken. Undoubtedly, improvements could be made by using techniques such as those proposed in [16]. That is, exploiting parallel processing capabilities of modern multi-core CPUs when constructing a limit order book data structure. However, this work was beyond the scope of this project as the focus was placed on the general principle of how to map the order placement problem into the reinforcement learning context, rather than building an efficient matching engine. A more serious limitation when matching orders using historical order books is the absence of market participants, who could have 1) entered or 2) left the market upon placing an order during the simulation. Participants who would enter the market would probably be favorable to the order submitted 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. Therefore, the matching process is strictly speaking an estimation of what would have happened in the past. However, since only up to 1.0 BTC were used for buying and selling, which is insignificant in terms of market impact[23], we considered the approximation of the matching process as sufficiently correct. Otherwise, the only way to overcome this limitation would have been to train with and test on live market data and make real purchases and sales.

Due to the complexity of the limit order placement process it was necessary to equip the environment with a variety of configuration parameters, as defined in Section 5.1.2. While this provides the ability to define evaluation setups very precisely, it also introduces the inevitable obligation of limiting the scope of our approach. We have decided to define a discrete time step size $\Delta t = 10s$, which represents the duration of how long an order remains in the order book for each step taken by the agent and therefore limits the number of steps to be taken throughout one epoch to $\frac{H}{\Delta t} = 10$. With an empirical investigation (Section 6.3), we have shown that a placement of 10 seconds diverges from the expected rewards received from placing the order for a total of 100 seconds. Naturally, if the agent decided in each step to choose the same action, the result would be equivalent to a placement over 100 seconds. Therefore, the benefit of the time-step parameter is the agent's ability to intercept and change the action accordingly, in the event of a sudden change in the market data. Investigations were made with time-step parameters smaller than 10 seconds (3s, 5s) and no improvements could be seen. The only difference was an increased training time as the agent had to take more steps in every epoch. Another important parameter is the size of the action space, which we defined as consisting of 50 negative and 50 positive actions that result in a total action space size of $|A| = 101$. The empirical investigation in Section 6.3 has confirmed that this is indeed the range of actions which have most influence in the posting of orders over a time horizon of 100 seconds. However, the limitations found in Section 6.6 showed that there are rare occasions where the defined action space is too small and therefore the agent is unable to place the order at a price level close to the offers made by potential buyers or sellers. Increasing the already-large action space would certainly allow the agent to place orders at the appropriate price levels. However, experiments with $|A| = 201$ have shown that this comes with the cost of increased training as well as a greater potential for misplacing orders; all these factors meant that performance could not be improved. In addition, different settings for the action step size Δa, which would enable us to increase the price range at which orders are placed by maintaining equal action space size. However, this implies that the agent would make more coarse decisions in terms of the price level at which to place orders and therefore potentially optimal placements would be out of reach. Our investigations have shown (Table 6.2) that no improvements could be made by increasing the action steps and the optimal setting is $\Delta a = \$0.10$.

The use of the OpenAI Gym toolkit[18] allowed us to design our environment in a way such that multiple

agent could make use of the environment. The Q-Learning and DQN agents developed are similar in terms of the reward function as both underlie the principles of the Bellman equation (Eq. 5.1). However, much complexity is added to the DQN agent, such as the use of an *experience replay* and most importantly a *value function approximator*. The neural network used served to approximate the patterns within our data set that were found and described earlier. This enabled us to feed private and market variables to this agent. Furthermore, this agent provided great flexibility when constructing the feature vectors. In contrast, the Q-Learning agent was not found to be a useful choice in this regard. While the Q-Learning agent can be suitable when no market variables or approximations thereof are applied, it was found not to be an appropriate choice for the purpose of optimizing the limit order placement problem. Market features such as the ones constructed in Chapter 4 did not provide any benefit when applied to the Q-Learning agent. The reason for this is that Q-Learning makes use of a look-up table which records a combination of all observed states and chosen actions. Since the observation states are derived from market data at a given point in time, most of them are unique and thus learning time is expected to scale exponentially with the size of the state space[27, 44]. To overcome this limitation, market features would have to be approximated drastically, as demonstrated in [37].

Furthermore, we briefly investigated the architecture of the neural network and ultimately equipped the DQN agent with the widely-used convolutional neural network architecture[35], defined as DQN-CNN, as well as with a simpler multilayer perceptron with two hidden layers, defined as DQN-MLP. Our investigations (Section 6.5) showed that the results were similar and the DQN-CNN setup performed best among all tested. We found that having a second network architecture helpful as it provided supporting numbers within our results and further confirmed the effectiveness of deep reinforcement learning applied in our context. However, there is room for improvements with regard to parameter tuning, which we neglected in this work in favor of prioritizing the architectural aspects of the reinforcement learning setup; we therefore suggest this as future work.

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

No previous research has yet been applied to the cryptocurrency market and therefore there are no optimization performance numbers, to which the approach of this work can be compared. As a result, an evaluation procedure was elaborated as part of this thesis (Section 6.1). Therefore, two real-world market data sets were chosen as well as artificially generated limit order books. This multi-step procedure involved an empirical investigation of the expected costs of an optimally placed limit order as well as the costs of an immediate purchase or sale using a market order. Furthermore, the expected costs were taken as a benchmark for the Q-Learning agent as well as the DQN agent under the application of both aforementioned feature types. In addition, an evaluation mechanism was built into the reinforcement learning environment which enabled the investigation of the steps taken by an agent while running a training or testing epoch.

The way in which order placement was simulated relied significantly on suggestions related to backtesting made by Marcos Lopez de Prado [19]. The selection of a random order book state that serves as the starting point of an epoch drastically reduces the chances for possible over-fitting. With this in mind, we considered data sets which show different market price constellations and furthermore, developed a way to create artificial order books on which to train and test the agent. It is obvious that, only by considering an infinite amount of data sets can a completely generalized statement about the optimization capabilities be made; however, as mentioned earlier, this comes at the expense of time and computational resources. However, our novel approach to generating artificial data sets has proven to be of substantial value since the possible optimization factor is known prior to the validation of the model and processing is much faster.

The empirical investigation step executed prior to the training of a policy has provided baseline performance results. With this approach we were able to determine and improve the parameter setting of the reinforcement learning environment, without proceeding actual learning tasks, which made it very efficient. Furthermore, we were able to determine the expected costs for market and limit orders and the results have shown that the expected behavior for limit order placement is similar to what was proposed by Kearns et al. [36], who worked with traditional stocks. However, we have given more details of this behavior with respect to the given time horizon for the placements of orders. The authors of [36] stated that, overall, the optimal price at which to place an order is just below the spread price. This statement is true for the cryptocurrency market as well, however, only on condition that the time horizon chosen is either very small ($H \leq 10s$) or large ($H > 100s$). Otherwise, for example when $H = 100s$, we have demonstrated that the optimal price at which to place an order changes drastically depending on the fluctuations of the market price. Namely, when buying,

limit orders should be placed deep in the order book when market prices are falling, and high in the book when the price is rising. On the contrary, when selling, orders should be place high in the book when the market price is falling and low in the book when the price is rising. These insights not only provide knowledge of how to optimize the placement of orders but also generated the means to compare the performance achieved by reinforcement learning agents.

We further reflect that average rewards provide a much greater insight into the performance of reinforcement learners, as opposed to the average actions chosen. An average action, consisting of up to 10 values (equivalent to the maximum number of steps for one epoch) from a range of 101 total actions, certainly provides understanding of the average price levels at which the learners attempt to place orders. However, it is suggested that an observation and analysis of the specific sequence of actions chosen be undertaken in future work.

Another important method introduced during the evaluation was the possibility to record and visualize the decision-making process of the reinforcement learning agents, as shown in Section 6.6. Thereby we were able to observe in which market situations the agents did not respond with the most appropriate actions. This detected the limitations of the agent as well as the environment in greater detail.

Lastly, by considering the artificial data sets with linear and sine functions applied, near-optimal performance was achieved with the DQN agent and feature type I (window of historical order book states). This confirms that 1) reinforcement learning is indeed capable of optimizing limit order placement but 2) that there is potential to further improve the setup for real-world market data sets. Moreover, the increased optimization performance using a sine-shaped order book, compared to real-world market data, indicates that the learner had much difficulty finding a policy under non-stationary conditions and the C parameter (see Section 5.3) was only of minor assistance.

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

The aforementioned evaluation procedure was used to determine the efficiency of the constructed features that were applied to the DQN agent. The results given in Chapter 6 showed that, with the use of either of the two features, a policy can be learned that enables the optimization of order placement when market conditions are favorable to making a purchase or sale. However, under the application of either of the two features, the agent was not able to performed significantly better than the expected costs of a market order when conditions were not ideal. Moreover, it has been shown that the application of feature II (a sequence of historical trades) results in a better policy overall than the application of feature I (a window of historical order book states).

First, reinforcement learning without market features was tested by using the adapted Q-learning algorithm presented in Section 5.2. The similar approach presented by Kearns et al. [37], and tested on traditional stocks, achieved an optimization in the range of 27.16% to 35.5% for sell limit orders placed. We backtested both buying and selling scenarios on the cryptocurrency market Bitcoin/USD. Interestingly, we achieved a similar performance with the Q-Learner, when the market price moved against our favor while attempting to make a sale. That is, \$-27.70 was the expected return for selling 1.0 BTC and the agent reduced the return to \$-21.34, e.g. an improvement of 22.96%. However, when the market price was rising, which is in favor of selling, the agent was unable to outperform the expected market order costs of \$-1.72 and a loss of \$-4.74 was generated. In the buying process, the return improved from \$-1.06 expected, to \$-1.04, when the market price was rising; no improvements were achieved when the price moved against our favor–\$-1.17 compared to \$-0.05 expected. In general, it can be said that the strategy learned by the Q-learner is relatively close to the costs of a market order, with one exception where it was significantly more profitable. That suggests that the absence of market variables (our constructed features) allows the agent to learn the general principles of determining the price level at which to submit orders. Nevertheless, the agent was still not able to perform better than a market order.

The results found for our best setup of the DQN agent are an improvement over ones for the Q-Learning agent. During the empirical investigation, we stated that the sum of the expected market order returns was \$-30.53 and the sum of optimally placed orders was \$-9.88. The agent under the application of feature I (historical order book states) achieved a total reward of \$-18.62 and, under the application of feature II, a total reward of \$3.36. This was surprising since the structure of feature I is much more like the one used in [34] and better results were expected therefrom. However, this suggests that the information provided by historical trades is much more beneficial than information from historical order book states. Furthermore, we found

that the DQN agent had the most difficulties detecting changes in the market price trend, as shown in Section 6.6 (Figure 6.14). Thereby the agent expected rewards according to the input sample provided and therefore did not consider a possible reversal of the trend. However, during the evaluation of the limitation of the DQN agent (Section 6.6), we found no improvements to its performance when enlarging the window size of the feature provided to the agent and thereby covering long-term market movements. In addition, we increased the number of training epochs to 25,000 in the hope that the agent could improve its selection of actions in difficult market conditions, however, no improvements were found. At this point it is to be assumed that increasing the data sets and, at the same time, enlarging the feature vectors may improve the performance.

The next step taken involved the use of the artificial order books generated to which we applied the same feature vector setup as in the experiments undertaken with historical market data. Thereby, we found that the DQN agent was capable of finding a near-optimal policy for an order book that underlies a linear function as well as one that underlies a sine function. This suggests that the features consisting of non-stationary as well as noisy market data prevented the agent from achieving similar results.

7.3. Recommendations and future work

The most significant recommendation would be to run simulations of limit order placement with the setup provided in this work on a live market where it would make actual sales and purchases. Thereby, it would be possible to determine to what extent market participants that enter or leave the market, during the placement of an order, would affect the conclusions found in this work. By doing so, market fees could be considered and integrated into the reward function provided. This process could then be followed by an observation; the purpose of which would be to determine whether or not the agent adjusts its actions such that market orders that come with the more expensive *taker fees* are chosen less often.

An alternative to a live market evaluation would be to make use of an entire artificial market, such as the one proposed by Raberto et al. [40]. Such a setup would require multiple agents to continuously post buy and sell orders. In this process, some of the agents could act as naive traders and others could make use of the learning capabilities provided in this work. It would then be interesting to see which category of agents would achieve a better performance overall.

Furthermore, some limitations were discovered in this work and further investigations were therefore suggested. This includes the extension of the evaluation procedure in which the sequence of actions chosen by an agent is investigated. The aim of this is to understand what makes an agent choose for a certain action over another. Additionally, the model presented and used in the DQN agent provides hyper-parameters which are to be further optimized. A systematic grid search would therefore provide insights into whether or not the problem can be further optimized.

In addition, this work could be extended by taking a hybrid approach with the help of imitation learning. A statistical framework such as the one proposed in [45] (see Section 3.2) would act as the expert agent from which an initial policy is learned. Subsequently, our deep reinforcement learning mechanisms could be used to learn patterns from the market data that is favorable to limit order placement. Thereby, we suggest updating the policy only if the rewards are either significantly positive or negative, in order to prevent noise being learned.

Lastly, the setup built in this work can be used not only to learn order placement but also to learn a specific market making[38] strategy. This strategy is to place buy and sell orders simultaneously with the incentive of making a profit from the difference between the price paid for the buy order and the price received for the sell order.

7.4. Application in real world practices

This work involved a pipeline of tasks, each of which contributed in part towards answering the research questions stated in this thesis. The tools and components developed throughout this process were consecutively extended and merged. The result of this is a ready-to-use product, in the form of a flexible framework that can implement an intelligent order placement strategy according to the historical or live market data provided by the end user. The configuration parameters of the environment and agent provided enable further adjustments to be made according to the user's specific needs. Moreover, our approach is not limited to cryptocurrency markets but also supports any product that is traded with limit order books, as long as data are provided. This in turn makes our setup attractive for numerous real-world settings. Financial exchanges could use this framework in order to provide a new *order type* that allows their customers to buy or sell an inventory I of an asset within a time horizon of H seconds. Moreover, this setup is not restricted to centralized

exchanges but can also be applied to decentralized ones where each node of the exchange would employ its independent agent that learns from the corresponding order book residing in that node. Another target group for this framework is (institutional) traders or brokers which intend to optimize the way in which they place orders on the exchanges of their choice. In this case, exchange data serves as the source of the reinforcement learning environment and the agent's task is to act as an *intermediary* between the trader and exchange.

Bibliography

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

[2] Cs231n - convolutional neural networks. URL http://cs231n.github.io/convolutional-networks. [Online; accessed April 30, 2018].

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

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

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

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

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

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

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

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

[11] Gentle introduction to models for sequence prediction with recurrent neural networks. URL https://machinelearningmastery.com/models-sequence-prediction-recurrent-neural-networks/. [Online; accessed April 30, 2018].

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

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

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

[15] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[16] Raphaël P Barazzutti, Yaroslav Hayduk, Pascal Felber, and Etienne Rivière. Exploiting concurrency in domain-specific data structures: A concurrent order book and workload generator for online trading. In *International Conference on Networked Systems*, pages 16–31. Springer, 2016.

[17] Bruno Biais, Pierre Hillion, and Chester Spatt. An empirical analysis of the limit order book and the order flow in the paris bourse. *the Journal of Finance*, 50(5):1655–1689, 1995.

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

[19] Marcos Lopez de Prado. *Advances in financial machine learning*. John Wiley & Sons, 2018.

[20] 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.

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

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

[23] Nikolaus Hautsch and Ruihong Huang. The market impact of a limit order. *Journal of Economic Dynamics and Control*, 36(4):501–522, 2012.

[24] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

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

[26] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[27] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[28] David Kane, Andrew Liu, and Khanh Nguyen. Analyzing an electronic limit order book. *The R Journal*, 2 (64-68):1, 2011.

[29] Jonathan M Karpoff. The relation between price changes and trading volume: A survey. *Journal of Financial and quantitative Analysis*, 22(1):109–126, 1987.

[30] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of machine learning*, pages 257–258. Springer, 2011.

[31] 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.

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

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Maureen O'Hara and George S Oldfield. The microeconomics of market making. *Journal of Financial and Quantitative analysis*, 21(4):361–376, 1986.

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

[40] Marco Raberto, Silvano Cincotti, Christian Dose, Sergio M Focardi, and Michele Marchesi. Price formation in an artificial market: limit order book versus matching of supply and demand. In *Nonlinear Dynamics and Heterogeneous Interacting Agents*, pages 305–315. Springer, 2005.

[41] Jens-Uwe Schluetter, Harris C Brumfield, Robert A West, and Sagy Pundak Mintz. System and method for randomizing orders in an electronic trading environment, January 26 2010. US Patent 7,653,589.

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

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

[44] Steven D Whitehead. Complexity and cooperation in q-learning. In *Machine Learning Proceedings 1991*, pages 363–367. Elsevier, 1991.

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