Market Making in Limit Order Books

using Reinforcement Learning

Aadam Wiggers

Delft University of Technology

TU Delft

Market Making in Limit Order Books

using Reinforcement Learning

by

Aadam Wiggers

Student Number: 5855063

Thesis Committee

Supervisor:	Prof. Dr. Antonis Papapantoleon,	TU Delft EEMCS
Co-supervisor:	Assistant. Prof. Amin Kolarijani,	TU Delft ME
Committee Member:	Associate Prof. Joris Bierkens,	TU Delft EEMCS

TU Delft

Abstract

Market making, the act of providing liquidity to the market by simultaneously buying and selling, is a difficult problem to solve. The use of reinforcement learning to solve for market making is increasing, as academics and practitioners alike look for novel ways to approximate for optimal policies in increasingly complex markets. This thesis examines the use of reinforcement learning to solve the market making problem in limit order books. To this end, we provide the theoretical background on market making, modelling limit order books, and reinforcement learning. Furthermore, we implement and compare 'classical' algorithms, such as Q-learning and value and policy iteration, and compare their policies with newer techniques involving deep learning, namely deep Q-networks and double deep Q-networks. To train and compare these models, we use two models to simulate the dynamics of an order book. Experiment results and the ultimately learned policies are presented and discussed. We propose several ideas that could be worked on in the future.

Acknowledgements

I would like to express my profound gratitude to Professor Antonis Papapantoleon for his guidance and insightful commentary throughout the many update meetings that we had. Furthermore, I deeply thank Assistant Professor Amin Kolarijani for the many fruitful (and joyous) discussions we had and helping me figure out how to solve for the implementation struggles I faced, you have taught me a lot on the field of MDPs and RL. I sincerely thank Associate Professor Joris Bierkens for being willing to be a part of the thesis committee on such short notice. I thank the colleagues that I have met for the many interesting discussions on the practicalities of the financial markets, and how reinforcement learning fits in.

Thank you to all my family and friends for your unwavering support and encouragement throughout the years. Lastly, I would like to thank my partner for being there, and for the many times you have had to listen to my ramblings.

Aadam Wiggers
Amsterdam, October 2024

Disclaimer

The content of this thesis is for informational purposes only. It is not intended as investment research or investment advice, or a recommendation, offer, or solicitation for the purchase or sale of any security, financial instrument, financial product, or service, or to be used in any way for evaluating the merits of participating in any transaction. I, the author, do not accept any liability for any investment decisions based on statements of this thesis.

Contents

1 Introduction **1**

2 Theory **3**

2.1 Introduction . . . 3

2.1.1 Walrasian Auction . . . 3

2.1.2 Market-Makers . . . 4

2.1.3 Limit Order Books . . . 4

2.1.4 Models à la Avellaneda and Stoikov . . . 5

2.2 Modelling Limit Order Books . . . 6

2.2.1 Markov Chain Model . . . 6

2.2.2 Queue-Reactive Model . . . 9

2.3 Reinforcement Learning . . . 13

2.3.1 Markov Decision Processes . . . 14

2.3.2 Dynamic Programming . . . 20

2.3.3 Temporal-Difference Learning . . . 23

2.3.4 Deep Learning . . . 25

2.4 Reinforcement Learning in Limit Order Book Market Making . . . 28

2.4.1 Optimal Trading Strategies . . . 28

2.4.2 Previous Work . . . 31

2.4.3 Experimental Setup . . . 34

3 Experiments and Results **37**

3.1 Trading Environment . . . 37

3.2 Parameters . . . 37

3.2.1 Training Parameters . . . 37

3.2.2 Test Parameters . . . 40

3.3 Markov Chain Model . . . 40

3.3.1 Training . . . 40

3.3.2 Results . . . 48

3.4 Queue-Reactive Model . . . 52

3.4.1 Training . . . 52

3.4.2 Results . . . 57

4 Discussion **62**

5 Conclusion **64**

5.1 Future Work . . . 65

References **66**

A Appendix **69**

1 Introduction

Market makers are market participants that simultaneously provide a buy and sell order to provide liquidity to the market. They continuously do this and generate a profit by capturing the difference between these two prices, ideally having a buyer and a seller at the exact same time, i.e. a risk-free profit. The two main objectives of a market maker is to

1. Maximize profits.
2. Minimize inventory risk.

Earlier works to model and solve for an optimal solution started with Ho and Stoll [1]. They used stochastic dynamic programming to derive optimal bid/ask prices, given a 'true' price, that maximize the dealer's expected utility of wealth over a given time horizon. This was later extended by Avellaneda and Stoikov [2], who assumed that the agent is but one player in the market, and that the 'true' price is given by the mid-price.

Advances in technology have led to the electronic order book, which has led to a rise in algorithmic trading, notably written in the paper by Menkveld [3]. This has made it possible to automate market making. Outside of the markets, and more recently, with the rise of computational power, the world has seen a surge in the use of machine learning techniques to solve complex problems, such as Mnih et al. [4] who used deep reinforcement learning to play Atari games.

Specifically, the machine learning used is a class of algorithms called reinforcement learning (RL). RL is a type of machine learning that is concerned with how agents ought to take actions in an environment to maximize some notion of cumulative reward. The field stemmed from Markov decision processes, and has since seen a rise in popularity with the advent of deep learning.

Market making lends itself to be quite naturally modelled as a Markov decision process, and thus using RL is a natural way to try and solve for an optimal policy. This is the main motivation for this thesis. We aim to use RL to solve for the market making problem in limit order books. More specifically, the research questions we aim to answer are:

1. Can reinforcement learning methods approximate/find optimal solutions to the market making problem?
2. How do policies compare under different market dynamics?

3. How do different RL algorithms compare in terms of performance?

Naturally, the RL agent has to interact with the environment for it to learn, and is, unfortunately, not possible with historical data, as these are records of what has already happened. As a result, we simulate the limit order book. Several authors have proposed models to simulate limit order books, and we use two of these models in this thesis. The first is the Markov chain model from Hult and Kiessling [5], and the second is the queue-reactive model from Huang et al. [6].

The remaining part of this thesis is structured as follows. In Chapter 2, we provide a theoretical background on market making, modelling limit order books, and reinforcement learning. We also delve into how these three can be combined to solve the market making problem. In Chapter 3, we describe the experimental setup, the training of the agents, and present the results. In Chapter 4, we discuss the results and in Chapter 5, we conclude the thesis and propose directions for future work.

2
Theory

This chapter is divided into four sections and aims to introduce the reader to market-making, limit order books, reinforcement learning, and how these three can be combined to solve the market-making problem. The chapter starts with an introduction to market-making and limit order books, followed by a discussion on how limit order books can be modelled. The chapter then delves into the theory of reinforcement learning. Lastly, the chapter discusses how reinforcement learning can be used in the context of solving for market making in limit order books.

2.1. Introduction

In order to give context to this thesis, we first have to go over a brief history of the evolution of markets. This follows from the introduction found in Bouchaud, et al. [7].

What is a market?

Markets are attempts to solve the seemingly impossible problem of allowing trades between buyers, who want to buy at an ever-lower price, and sellers, who want to sell at an ever-higher price.

2.1.1. Walrasian Auction

One possible way to solve the above problem is to force a trader to commit to a trade once they have specified a price they are willing to buy at. This gives the setting for a market organisation called a *Walrasian auction*. Traders place bids, a want to buy the asset, and offers (also called asks), a want to sell the asset, at the maximum/minimum price that they are willing to buy/sell. These bids/offers are firm commitments. Nowadays, this commitment is called *liquidity provision*.

The auctioneer then collects these bids and offers into an order book. The order book describes the quantities that are available for buying and selling at each price level, as declared by the participants. In a Walrasian auction this order book is invisible to the traders. At some point in time, the auctioneer will announce a transaction price p such that the total volume exchanged at this price is maximized. What this means is that all the buyers above price p will be matched with all the sellers below price p, leaving the buyers below price p and the sellers above price p unmatched.

2.1.2. Market-Makers

One can imagine that the above is not a practical way to run a market, as there is no coordination between buyers and sellers. This leads to two scenarios one can end up in:

- All buyers and sellers are unreasonably greedy, so there is no overlap and no transaction price p exists.
- One side is unreasonably greedy, and the other is not. This leads to a sudden jump in the transaction price p, and a small transaction volume.

So, to fix this issue, markets adopted a special category of traders called *market-makers*. Market-makers maintain a fair and orderly market, in exchange for some privileges. They perform two main tasks:

- **Quoting:** At all times, a market-maker must provide a *bid-price* p_b, with a *bid-volume* V_b, and an *ask-price* p_a, with an *ask-volume* V_a (ask = offer). These may change, but in between changes these quotes are legally binding.
- **Clearing:** Once buyers and sellers submit orders that specify a price and volume, the market-maker decides on a transaction price p to maximise transaction volume.

This separates the market into two main participants: market-makers are *liquidity providers*, and the rest are *liquidity takers*. This forms the basis of which electronic markets nowadays operate, and this thesis will focus on the market-making aspect of trading.

2.1.3. Limit Order Books

Along with the rest of the world, markets too have moved into the electronic age. Instead of using a designated market-maker, standing in a pit, shouting orders, we now have computers that match orders. This is done using a continuous-time double auction mechanism through a *limit order book* (LOB). This is a system where traders can submit orders to buy or sell at a specified price. The order book is visible to all traders, and the exchange matches buyers and sellers whenever they agree on a price. LOB's do not require a Walrasian auctioneer, nor a designated market-maker, as the market is self-clearing.

Features of Limit Order Books

In essence, a limit order book is an open record of all the different buy and sell orders in the market. There is a price level for each price at which there is an order, and the minimum allowed increments between these price levels are called the *tick size*.

There are different degrees of information that can be presented to traders. The most basic form is a *level 1* limit order book, where traders can only see the best bid and ask prices. A *level 2* limit order book shows the best bid and ask prices, as well as the volumes available at these prices. An example of a level 2 limit order book is shown in Figure 2.1. Lastly, the most granular, a *level 3* limit order book shows all the orders in the order book, including the volumes and prices of all orders, as well as their position in the price level. An example of a level 3 limit order book is shown in Figure 2.2.

The rules by which the limit order book operates, i.e. who gets priority when orders are matched, is determined by the *matching engine* of the exchange. This is a set of rules that determine how orders are matched, and in what order. The most common matching engine

is the *price-time priority* matching engine, where orders are matched based on the price of the order, and the time at which the order was placed. This means that orders with the same price are matched in the order they were placed. This is also known as a first in first out rule. This thesis concerns itself with a *level 3* limit order book, and the price-time priority matching engine.

Types of Orders

For this thesis, we consider the three following types of orders:

- **Limit order:** An insertion of a new order in the limit order book. (A buy order strictly less than the best ask price, or a sell order strictly higher than the best bid price).
- **Market order:** A buy or sell order at the best available price.
- **Cancel order:** A cancellation of an already existing order in the limit order book.

Bid Qty	Price	Ask Qty
	102	425
	101	80
100	100	
1000	99	

Figure 2.1: Example of a level 2 limit order book. Volumes per price level are aggregated.

Buyers	Ttl Bid	Price	Ttl Ask	Sellers
		102	425	['maverick']
		101	80	['optiver', 'imc']
['optiver', 'imc']	100	100		
['maverick']	1000	99		

Figure 2.2: Example of a level 3 limit order book. One can see the priority of the buyers/sellers in the queue.

2.1.4. Models à la Avellaneda and Stoikov

The main type of models one comes across when studying market-making are models that are based on the seminal paper by Avellaneda and Stoikov [2]. The Avellaneda-Stoikov model is a simple model that captures the essence of market making. The model is based on the following assumptions:

- Mid prices are modelled by a stochastic process, exogenous to the market-maker's behaviour.
- The probability that the market maker buys/sell the bid/ask they are quoting depends on the distance between the quoted price and the mid price.

The authors then optimise to a utility function. One is then able to derive an optimal solution to how to skew the bid/ask prices a market-maker should quote around a midprice, depending on inventory and time to expiry, as done by Gueant et. al [8].

Essentially, the authors solve for the following problem (in the eyes of a market-maker):

- *Given a midprice, current inventory, and time to expiry: how far away do I place my bid/ask prices?*

The main difference, as highlighted by Gueant [9], between models à la Avellaneda and Stoikov and limit order book modelling is that the Avellaneda and Stoikov models do not take into account the discrete nature of prices. All limit order books have a minimal price increment called a tick size, and this is not assumed in the model. Furthermore, the very nature of limit order books as queuing systems are not taken into account. As discussed before, the trader that submits the best price gets executed first. Also, there is a priority if two traders give the same price, namely time. This is not something that Avellaneda and Stoikov take into account in their seminal paper.

This does not mean that models that stem from the Avellaneda and Stoikov paper are not used. They are mostly suitable to dealer based markets, where the limit order book queuing dynamics do not matter. An example of a dealer based market is the over-the-counter bond trading done by, for example, large banks.

2.2. Modelling Limit Order Books

There are countless ways to model limit order books. In this section, we will go over the theory behind two models that have been proposed in the literature. The first model is that of a continuous time Markov chain model, as proposed by Hult and Kiessling [5]. The second model is the queue-reactive model proposed by Huang et al. [6]. Both models are based on the idea that the order book can be represented by a Markov chain. The models differ mainly in the way the order book is updated, i.e. the functions by which orders are sent, as the queue-reactive model takes into account the queue size at each price level.

2.2.1. Markov Chain Model

Hult and Kiessling [5] present a model for a limit order book using a continuous time Markov chain. We will now go over the theory presented in their paper.

Modelling Framework

A continuous time Markov chain $X = (X_t)$ is used to model the limit order book. It is assumed that there are $d \in \mathbb{N}$ possible price levels in the order book, denoted by $\pi^1 < \cdots < \pi^d$. The Markov chain $X_t = (X_t^1, \dots, X_t^d)$ represents the volume at time $t > 0$ of buy orders (negative value) and sell orders (positive value) at each price level. Assume that $X_t^j \in \mathbb{Z}$ for each $j = 1, \dots, d$. The state space of the Markov chain is denoted by $\mathcal{S} \subset \mathbb{Z}^d$. The generator matrix of X is denoted $Q = (Q_{xy})$, where Q_{xy} is the transition intensity from state $x = (x^1, \dots, x^d)$ to state $y = (y^1, \dots, y^d)$. The matrix $P = (P_{xy})$ is the transition probability matrix of the jump chain of X. Let $X = (X_n)_{n=0}^{\infty}$, where n is the number of transitions from time 0.

For each state $x \in \mathcal{S}$ let

$$
\begin{aligned}
j_B = j_B(x) &= \max\{j : x^j < 0\} \ \textit{(highest bid level)}, \\
j_A = j_A(x) &= \min\{j : x^j > 0\} \ \textit{(lowest ask level)}.
\end{aligned}
$$

Assume that $x^d > 0$ for all $x \in \mathcal{S}$, i.e. there is always someone willing to sell at the highest possible price, and that $x^1 < 0$ for all $x \in \mathcal{S}$, i.e. there is always someone willing to buy at the

lowest possible price. Also assume that $j_B < j_A$. The *bid price* is defined to be $\pi_B = \pi^{j_B}$ and the *ask price* is defined to be $\pi_A = \pi^{j_A}$. Note that there are no limit orders between the bid and ask price, i.e. $x^j = 0$ for all $j_B < j < j_A$. The *spread* is the distance $j_A - j_B$.

We now look at all the possible transitions of the Markov chain X (possible ways that the order book can change). Let $e^j = (0, \ldots, 0, 1, 0, \ldots, 0)$ be the jth unit vector in $\mathbb{Z}^d$. The possible transitions are:

- **Limit buy order:** A limit buy order of size k at level j is an order to buy k units at price π^j. The order is placed last in the queue of orders at price π^j. The transition is $x \to x - ke^j$, where $j < j^A$ and $k \geq 1$. That is, a limit buy order can only be placed at a level lower than the best ask level j_A.
- **Limit sell order:** A limit sell order of size k at level j is an order to sell k units at price π^j. The order is placed last in the queue of orders at price π^j. The transition is $x \to x + ke^j$, where $j > j^B$ and $k \geq 1$. That is, a limit sell order can only be placed at a level higher than the best bid level j_B.
- **Market buy order:** A market buy order of size k is an order to buy k units at the best ask price. The transition is $x \to x - ke^{j_A}$, where $k \geq 1$. Note that if $k \geq x^{j_A}$, then the order will clear all the volume at that price level, resulting in a new lowest ask level.
- **Market sell order:** A market sell order of size k is an order to sell k units at the best bid price. The transition is $x \to x + ke^{j_B}$, where $k \geq 1$. Note that if $k \geq |x^{j_B}|$, then the order will clear all the volume at that price level, resulting in a new highest bid level.
- **Cancellation of buy order:** A cancellation of a buy order of size k at a level j is an order to instantly remove k limit buy orders at a level j from the order book. The transition is $x \to x + ke^j$, where $j \leq j^B$ and $1 \leq k \leq |x^j|$.
- **Cancellation of sell order:** A cancellation of a sell order of size k at a level j is an order to instantly remove k limit sell orders at a level j from the order book. The transition is $x \to x - ke^j$, where $j \geq j^A$ and $1 \leq k \leq x^j$.

To summarise, the possible transitions are such that Q_{xy} is non-zero if and only if state y is of the form

$$y = \begin{cases} x + ke^j, & j \geq j^B(x), \quad k \geq 1, \\ x + ke^j, & j \leq j^B(x), \quad 1 \leq k \leq |x^j|. \\ x - ke^j, & j \leq j^A(x), \quad k \geq 1, \\ x - ke^j, & j \geq j^A(x), \quad 1 \leq k \leq x^j. \end{cases}$$

Now it remains to be seen how the model is parameterised.

Parameterisation

To ease calibration of the proposed model, the authors follow the framework of a zero intelligence model. This is a model where the transition probabilities are only dependent on the location of the best bid and ask, and is for the rest state independent. This is an extremely well studied model, and is known for its simplicity and ease of calibration. A deeper dive into the dynamics of such models, and how they can effectively capture the behaviour of real order books, can be found in the work of Farmer et al.[10]

Given two different states $x, y \in \mathcal{S}$, the transition intensity from x to y is denoted as Q_{xy}. The waiting time until the next transition is assumed to be exponentially distributed with parameter

$$-Q_x = \sum_{y \neq x} Q_{xy}.$$

The transition matrix of the jump chain, $P = (P_{xy})$, denotes the probability of transitioning from state x to state y in one jump. The transition matrix is obtained from Q by normalising the rows, i.e.

$$P_{xy} = \frac{Q_{xy}}{-Q_x}.$$

The model can now be completely determined by the initial state and the non-zero intensities for the transition rates of the Markov chain. The authors propose a simple model, namely that the limit, market, and cancellation orders arrive according to a Poisson distribution specified by the following parameters:

- Limit buy (sell) orders arrive at distance i levels from best ask (bid) at rate $\lambda_L^B(i)$ ($\lambda_L^S(i)$).
- Market buy (sell) orders arrive at rate λ_M^B (λ_M^S).
- The size of limit and market orders follow discrete exponential distributions with parameters α_L and α_M.
  - The distributions of the sizes of limit orders $(p_k)_{k \geq 1}$ and market orders $(q_k)_{k \geq 1}$ are given by
$$p_k = (e^{\alpha_L} - 1)\, e^{-\alpha_L k},\ q_k = (e^{\alpha_m} - 1)\, e^{-\alpha_M k}.$$
- Cancellation rates are $\lambda_C^B(i)$ and $\lambda_C^S(i)$. It is assumed that the size of the cancellation orders are 1.

One can calibrate the model to real data by estimating the parameters above, but we will use the authors' calibration for the sake of simplicity. The authors calibrate the model to a EUR/USD exchange rate traded on a foreign exchange and get the following parameters:

Levels (i)	1	2	3	4	5
$\lambda_l(buy)$	0.1330	0.1811	0.2085	0.1477	0.0541
$\lambda_l(sell)$	0.1442	0.1734	0.2404	0.1391	0.0584
$\lambda_c(buy)$	0.1287	0.1057	0.0541	0.0493	0.0408
$\lambda_c(sell)$	0.1308	0.1154	0.0531	0.0492	0.0437

$\lambda_m(buy)$	0.0467	α_l	0.5667
$\lambda_m(sell)$	0.0467	α_m	0.4955

Table 2.1: Calibrated parameters found in Hult and Kiessling [5].

To illustrate the model, we simulate a limit order book using the parameters above. With a starting price of 100, and tick size equal to 1, an example simulation run for a 1000 steps is shown in Figure 2.3.

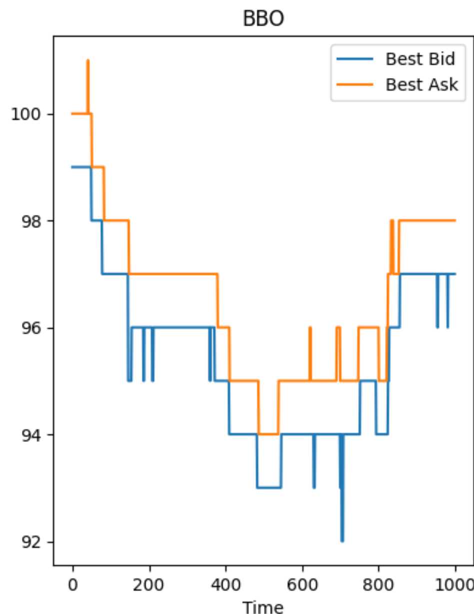

Figure 2.3: Simulation of a limit order book using the Markov chain model.

From what we can see, the model captures the dynamics of a price process in a limit order book relatively well, where we see prices and volumes (not pictured) fluctuate like a real market would. However, the model is quite simple and does not take into account several other variables, such as the queue size at each price level. Furthermore, the calibration of the model is done with a zero intelligence model. While being able to capture dynamics relatively well, it is not able to capture all empirical relationships observed in limit order books, something Farmer et al. [10] also mention in their work.

Thus, in order to study how well reinforcement learning algorithms will perform in more complex situations, we will now move on to the theory of a slightly more complex model, namely the queue-reactive model.

2.2.2. Queue-Reactive Model

The queue-reactive model was proposed by Huang et al. [6]. The model is based on analysis of high frequency data and is able to accommodate empirical properties of order book dynamics. The authors split the time interval of interest into periods in which a reference price remains constant. Within these periods, the limit order book is viewed as a Markov queuing system. The intensities of the order flows only depends on the current state of the order book. To take into account the whole period of interest, the authors present a stochastic mechanism to switch from one period of constant reference price to another.

Constant Reference Price

We will begin with outlining the dynamics of the LOB in a period of constant reference price. Let δ be the tick value of the limit order book. The limit order book is seen as a $2K$-dimensional vector, where K denotes the number of limits on each side. In the Markov chain model, the number of price levels was denoted as d. The reference price p_{ref} defines the center of the $2K$-dimensional vector, dividing the limit order book into two parts:

- **Bids:** $[Q_{-i} = p_{ref} - (i - 0.5)\delta]_{i=1}^{K}$,
- **Asks:** $[Q_i = p_{ref} + (i - 0.5)\delta]_{i=1}^{K}$,

where Q_j denotes the price level j. The number of orders at Q_j is denoted by q_j.

Furthermore, we assume that on both sides, the market sends limit, cancel, and market orders. We further assume that there is a constant order size at each limit, but do allow for different order sizes at different limits. The authors call this size at each limit Q_i the average event size (AES_i for short).

As a result, we now have a $2K$-dimensional process $X(t) = (q_{-K}(t), \ldots, q_{-1}(t), q_1(t), \ldots, q_K(t))$ that represents the limit order book. The limit order book is then modelled as a continuous-time Markov jump process, with jump size equal to one.

Estimation of Reference Price

As mentioned above, the reference price defines the center of the $2K$-dimensional vector. This definition is relevant for the positioning of the limits Q_i. Using the framework above, we must have that

$$p_{ref} = \frac{p_1 + p_{-1}}{2},$$

where p_1 and p_{-1} denote the best bid and best ask prices respectively. When the bid-ask spread is equal to one tick, p_{ref} is the midprice. However, when the bid-ask spread is larger than one tick, we can take several choices. The authors propose the following choices for odd or even spreads sizes (in ticks) larger than one:

- **Odd spread:**

$$p_{ref} = p_{mid} = \frac{p_{bb} + p_{ba}}{2},$$

  where p_{bb} and p_{ba} denote the best bid and best ask prices respectively.

- **Even spread:**

$$p_{ref} = p_{mid} + \frac{\text{tick}}{2} \text{ or } p_{ref} = p_{mid} - \frac{\text{tick}}{2},$$

  choosing one which is closes the the previous value of p_{ref}.

Model 1: Collection of Independent Queues

In the first iteration of the queue-reactive model, the authors assume independence between the flows arriving at different limits in the limit order book. Flows is a term used in the industry for the arrival of different types of orders. We assume that the intensities of the orders arriving are only functions of the queue size available at the limit Q_i. Furthermore, the different orders are taken to have independent intensities. Thus, we have:

- **Limit orders:** $\lambda_i^L(x)$,
- **Market orders:** $\lambda_i^M(x)$,
- **Cancel orders:** $\lambda_i^C(x)$.

We take the queue size at the limit q_i to be the input x. We also assume symmetry in the bid and ask sides, and thus we have that

$$\lambda_i^L(x) = \lambda_{-i}^L(x), \quad \lambda_i^M(x) = \lambda_{-i}^M(x), \quad \lambda_i^C(x) = \lambda_{-i}^C(x).$$

The methods the author uses for parameter estimation is outlined as follows:

1. Define an "event" ω, where an event is any modification of the queue size.
2. For queue Q_i, record:
   - The waiting time $\delta t_i(\omega)$ (in seconds) between ω and the preceding event at Q_i.
   - The type of event $\mathcal{T}_i(\omega)$, where
     - $\mathcal{T}_i(\omega) \in \mathcal{E}^+$ for a limit order insertion at Q_i.
     - $\mathcal{T}_i(\omega) \in \mathcal{E}^-$ for a limit order cancellation at Q_i.
     - $\mathcal{T}_i(\omega) \in \mathcal{E}^M$ for a market order at Q_i.
   - The queue size $q_i(\omega)$ before the event, which is approximated by an integer greater or equal to the volume at the queue divided by the AES_i.
3. When the reference price changes, restart the recording process.

Once the data $(\Delta t_i(\omega), \mathcal{T}_i(\omega), q_i(\omega))$ is collected, the authors estimate the intensities $\lambda_i^L(x), \lambda_i^M(x), \lambda_i^C(x)$ by maximum likelihood estimation:

$$\hat{\Lambda}_i(n) = \text{mean}(\Delta t_i(\omega)|q_i(\omega) = n)^{-1},$$
$$\hat{\Lambda}_i^L(n) = \hat{\Lambda}_i^n \frac{\#\{\mathcal{T}_i(\omega) \in \mathcal{E}^+, q_i(\omega) = n\}}{\#\{q_i(\omega) = n\}},$$
$$\hat{\Lambda}_i^M(n) = \hat{\Lambda}_i^n \frac{\#\{\mathcal{T}_i(\omega) \in \mathcal{E}^M, q_i(\omega) = n\}}{\#\{q_i(\omega) = n\}},$$
$$\hat{\Lambda}_i^C(n) = \hat{\Lambda}_i^n \frac{\#\{\mathcal{T}_i(\omega) \in \mathcal{E}^-, q_i(\omega) = n\}}{\#\{q_i(\omega) = n\}},$$

where 'mean' denotes the empirical mean, and # denotes the number of events.

Figure 2.4 shows the authors' estimated intensities for the stock France Telecom (now Orange), which we will also use for the simulator in this thesis.

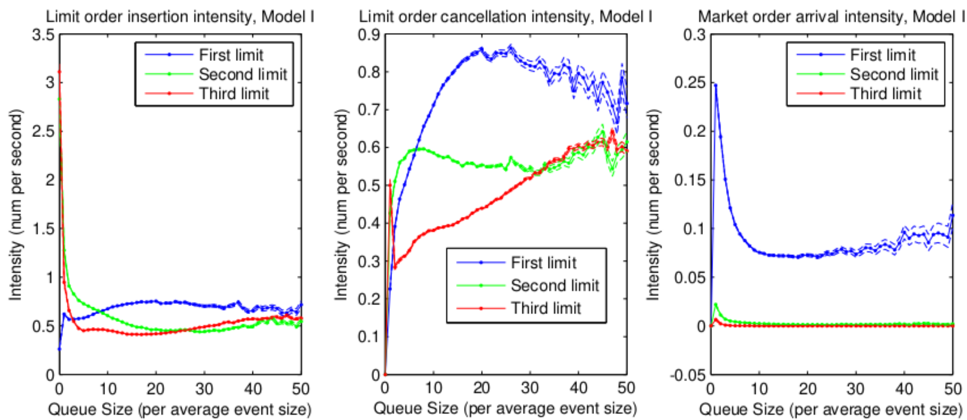

Figure 2.4: Intensities at $Q_{\pm i}, i = 1, 2, 3$, France Telecom. Estimated by the authors in [6].

We quickly analyse the intensities that we observe. We see that in the limit order insertions, that at the first limit, the intensity is almost constant with respect to the queue. However,

at the second and third limit, the intensity is a decreasing function of the queue size. This indicates that agents post orders at the second and third limit more when the queue size is small to seize order priority. For market orders, we observe a similar phenomenon in the first limit. This is probably due to the market participants rushing to execute when there is a lack of liquidity, but waiting for a better price when there is more orders resting. The second and third limit are almost zero, natural, as market orders are executed at the best price. For cancellations, we see that the intensity is a strongly increasing concave function with respect to the queue size. This is a large contrast to the linear approximations previously used, such as Cont et al. [11]. This could be due to market participants cancelling their orders when they see that the queue size is large, as they are not likely to get executed, i.e. having a lower queue priority. The second and third limit behave similarly, with even quicker rates of cancellations as the queue size increases.

Queue-Reactive Model

The largest difference between the model outline so far, and the Markov chain model proposed by Hult and Kiessling [5] is that the intensities for orders in the Huang et al. [6] paper is a function of the queue sizes at the order limits. In the Markov chain model, the authors use a zero-intelligence model, so there is no dependency on the state besides the distance of the levels from the best bid-ask.

In order to increase complexity in this model, the authors introduce a dynamic reference price. They assume that p_{ref} changes with some probability θ when some event modifies the midprice p_{mid}. Specifically, if p_{mid} increases/decreases, p_{ref} increases/decreases by δ with some probability θ, given that $q_{\pm 1} = 0$ at that moment (the best bid-ask queues are empty). Therefore, changes in p_{ref} are possibly triggered by:

- The insertion of a buy/sell limit order within the bid-ask spread when Q_1/Q_{-1} is empty at this moment.
  - I.e. a limit order becoming the best bid/ask.
- A cancellation of the last remaining best bid/ask order.
- A market order that consumes the last remaining best bid/ask order.

As a result, when p_{ref} changes, the value of q_i switches to the value of one of its neighbours (either q_{i+1} or q_{i-1} if p_{ref} increases or decreases respectively). Thus, one has to be careful in implementation, as the average event sizes are not the same for different queues.

Furthermore, to model exogenous information into the model, the authors assume that with a probability θ_{reinit}, the limit order book state is redrawn from its invariant distribution around the new reference price when p_{ref} changes. The authors justify this by considering that this can be seen as a flurry of activity from market participants to readjust their order flows around a new reference price.

Finally, we have that the market dynamics are now modelled by a $(2K+1)$-dimensional Markov process $\tilde{X}(t) := (X(t), p_{ref}(t))$, where $X(t)$ is the same as before.

Similar to the Markov chain model, an example simulation of 1000 steps for an order book is outlined in Figure 2.5.

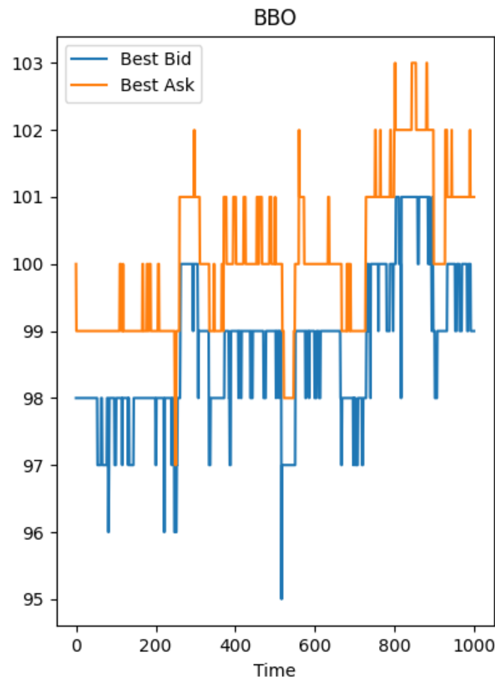

Figure 2.5: Simulation of a limit order book using the queue-reactive model.

Again, we see that the model captures the dynamics of a price process in a limit order book well. It is difficult to really tell if a model actually works well with only one visualisation, but we refer to the author's paper for a more in depth analysis as to why the queue-reactive model is a more realistic limit order book simulator. This is a sanity check that the price process at least behaves like a real market would. While not illustrated, the volumes also fluctuate in a realistic manner.

It is important to highlight that using the parameters from Huang et al. [6] results in a model that is inherently more volatile compared to the Markov chain model's parameters from Hult and Kiessling [5]. This is due to the fact that the Queue-Reactive model was calibrated on France Telecom, a stock listed on the French stock exchange, while the Markov chain model was calibrated on the EUR/USD, which is significantly more stable. This distinction is a key factor when comparing the outcomes of the two models. The parameters from Huang et al. [6] were chosen to explore how agents perform in a more complex and volatile environment. Additionally, the queue-reactive model is more realistic, as it reflects the structure of most equity markets, which operate using a limit order book, making it a more relevant framework for this analysis.

2.3. Reinforcement Learning

A natural framework used to model and solve for the market-making problem in limit order books is the use of Markov decision processes, as done in Hult and Kiessling [5]. This stems quite naturally due to how we modelled the limit order books so far, namely as Markov chains. This will also be how this paper aims to solve the market-making problem in the two environments outlined in the previous section. To do so, we need to delve into the theory behind these methods.

In this section, we will outline the theory of Markov decision processes, dynamic programming and go into temporal-difference learning, sourced mainly from Sutton and Barto [12]. This is then finished with a dive into neural networks and deep reinforcement learning.

2.3.1. Markov Decision Processes

In this section, we introduce the theory of Markov decision processes, which is a framework used to model sequential decision-making problems. We discuss the agent-environment interface, how it relates to market making, and define goals and rewards. We then introduce the concept of value functions and policies, and how they can be used to solve for optimal policies in Markov decision processes.

Agent-Environment Interface

Markov decision processes (MDPs) are a framing of the problem of learning from interaction to achieve a goal. The learner/decision-maker is called the agent. The agent interacts with the environment, which comprises everything outside of the agent. These two interact continually, with the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. The agent receives rewards as feedback from the environment, which is the basis for the agent's learning. Below, an illustration of this agent-environment interaction 2.6.

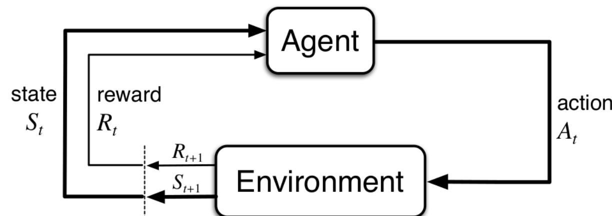

Figure 2.6: The agent-environment interaction in a Markov decision process. Taken from Sutton and Barto [12].

More specifically, the agent and environment interact at discrete time steps $t = 0, 1, 2, \dots$. At each time step t, the system is in the state $S_t \in \mathcal{S}$, on which the decision maker can choose an action $A_t \in \mathcal{A}(s)$, where $\mathcal{A}(s)$ is the set of actions available in state S_t. The agent receives a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and the system then moves to a new state S_{t+1}.

This gives rise to a sequence or trajectory that looks as follows:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, \dots \tag{2.1}$$

We concern ourselves with finite discrete MDPs, which is when the state and action spaces are finite and discrete. In this scenario, the random variables S_t and R_t have well defined probability distributions dependent only on the preceding state and action. That is, at a time t, for states $s, s' \in \mathcal{S}$, reward $r \in \mathcal{R}$, action $a \in \mathcal{A}$, there is a probability of those values occurring, given the preceding state and action:

$$p(s', r|s, a) := \mathbb{P}\left(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\right), \tag{2.2}$$

for all $s, s' \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}$. This function p defines the dynamics of the Markov decision process. Note that as a result p satisfies the following property, as it is the sum of conditional probabilities:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.3}$$

In a Markov decision process, the probabilities given by p completely characterizes the environment's dynamics. This implies that the values each state S_t and reward R_t depends only on the preceding state S_{t-1} and action A_{t-1}. This is the Markov property, which states that the future is independent of the past given the present.

As a result, we can define the state transition probabilities $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ as follows (this abuses notation slightly):

$$p(s'|s, a) := \mathbb{P}\left(S_{t+1} = s' | S_t = s, A_t = a\right) = \sum_{r \in \mathcal{R}} p(s', r | s, a). \tag{2.4}$$

We can also compute the expected rewards for a state-action pair (s, a) as a function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$:

$$r(s, a) := \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a), \tag{2.5}$$

and finally the expected rewards for a state-action-next state triplet (s, a, s') as a function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$:

$$r(s, a, s') := \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in \mathcal{R}} r p(s', r | s, a)}{p(s'|s, a)}. \tag{2.6}$$

This framework is abstract and very flexible. It can be used to model a wide range of sequential decision-making problems, and naturally can be applied to the market-making problem in limit order books. The market-maker becomes the agent, the simulated limit order book its environment, and the actions the market-maker can take are the different types of orders it can place. The rewards are then functions of profits or losses the market-maker incurs from these actions.

Goals, Rewards, Returns and Episodes

Now that we have defined the framework, we can introduce the concept of goals and rewards. The reward is simple, as it is the feedback from the environment to the agent. As mentioned, at each time step t, the reward is a number $R_t \in \mathcal{R}$.

The goal is more complex, as it is the objective the agent is trying to achieve. Informally, the goal is to maximize the rewards the agent receives. Thus, not maximising immediate reward, but cumulative reward in the long run. Sutton and Barto [12] states this informal idea as the *reward hypothesis*, which states that all goals can be described by the maximization of the expected cumulative reward. In the context of market-making, the goal could be to maximize profits, minimize losses, or some other objective over the trading day.

Formally, the sequence of rewards after time step t is denoted as

$$R_{t+1}, R_{t+2}, R_{t+3}, \dots . \tag{2.7}$$

In general, we seek to maximize the expected return, denoted G_t, which is a function of the rewards sequence. In its simplest case, we can define the return as the sum of rewards after time step t:

$$G_t := R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T, \tag{2.8}$$

where T is the final time step. This approach of a final time step makes sense in our application, where there is a natural notion of the final time step, i.e. the end of the trading day. Thus, the agent-environment interaction can be divided into subsequences called episodes. These episodes can be anything, a trading day, a trading week or in another setting a round of a game. These episodes end in a special state called the terminal state, where it is then followed by a reset to the initial state, independent of what happened in the previous episode.

We define a discount factor $\gamma \in [0, 1]$ to weigh the importance of future rewards. This gives rise to the discounted return:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k. \tag{2.9}$$

This approach makes the agent select actions so that the sum of discounted rewards is maximized. The discount factor γ determines the importance of future rewards. A $\gamma = 0$ makes the agent myopic, only caring about the immediate reward. A $\gamma = 1$ makes the agent care about all future rewards equally.

Important to note, we rewrite the goal in a more compact form, using recursion:

$$\begin{aligned} G_t &:= R_{t+1} + \gamma G_{t+1} \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{2.10}$$

We utilise this to in the following sections to define value functions and policies.

Policies and Value Functions

In many reinforcement learning algorithms, the central objective is to estimate *value functions*, which are a measure of how good it is for the agent to occupy a particular state (or to take a specific action in a given state). The concept of 'good' here is inherently tied to the goal that we defined previously. Naturally, the rewards an agent anticipates are dependent on its future actions, and thus these value functions are dependent on the agent's way of choosing actions, called a *policy*.

A policy is defined as a mapping from states to probabilities of choosing each action.

Definition 1 (Policy) *Suppose we have a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, and transition probabilities p. We define a policy π as a mapping from states to probabilities of selecting each action:*

$$\pi(a|s) := \mathbb{P}\left(A_t = a | S_t = s\right), \tag{2.11}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

The policy π can be deterministic or stochastic. A deterministic policy is one where the agent always chooses the same action in a given state. A stochastic policy is one where the agent chooses actions randomly, according to some probability distribution.

The *value function* of a state s under a policy π is the expected return when starting in state s and following policy π thereafter. We denote this as $v_\pi(s)$:

Definition 2 (State-Value Function) *For a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities p, and policy π, the value function $v_\pi(s)$ is defined as*

$$v_\pi(s) := \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \quad \forall s \in \mathcal{S}, \tag{2.12}$$

where $\mathbb{E}_\pi$ denotes the expectation under policy π.

Similarly, the value of *taking action a* in state s under a policy π, is the expected return starting in state s, taking action a, and following policy π thereafter. We denote this as $q_\pi(s, a)$:

Definition 3 (Action-Value Function) *For a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities p, and policy π, the action-value function $q_\pi(s, a)$ is defined as*

$$q_\pi(s, a) := \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right], \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{2.13}$$

where $\mathbb{E}_\pi$ denotes the expectation under policy π.

Now, we utilise the recursive relationship again in these definitions to define the value functions in terms of each other. For any policy π and any state s, we can get the relationship between

the value s and the value of the next state s',

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi \left[G_t | S_t = s\right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right], \end{aligned} \tag{2.14}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}$. The last equality is called the *Bellman equation* for v_π, and it expresses a relationship between the value of a state and the value of its successor states. Why do these recursive definitions, one might ask. The simple answer, to simplify formulas. One can now easily read the Bellman equation as a sum of rewards and the value of the next state, weighted by the probability of transitioning to that state, i.e. the expected value of the next state. More importantly, however, is that this recursive relationship can be used to actually solve for the value functions, as we will see in the next sections, specifically under dynamic programming. Note that the full derivation can be found in Appendix A.

Optimal Policies and Optimal Value Functions

We have now seen that solving a reinforcement learning problem is equivalent to finding the policy that maximises the reward in the long run. Since we are working with finite MDPs, we can define the optimal policy as follows. A policy π is better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states, i.e.

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s), \quad \forall s \in \mathcal{S}. \tag{2.15}$$

There always exists at least one policy that is better than or equal to all other policies. It may not be necessarily unique, and all the these policies are called the *optimal policy*, denoted π_*.

Definition 4 (Optimal Policy) *For a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities p, and discount factor γ, the optimal policy π^* is the policy that maximizes the value function for all states:*

$$\pi^* := \arg\max_\pi v_\pi(s), \quad \forall s \in \mathcal{S}. \tag{2.16}$$

By the relationship in 2.15, these optimal policies all share the same value function, called the *optimal value function*, denoted v_*:

Definition 5 (Optimal Value Function) *For a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities p, and discount factor γ, the optimal value function v_* is the value function that maximizes the expected return for all states:*

$$v_*(s) := \max_\pi v_\pi(s), \quad \forall s \in \mathcal{S}. \tag{2.17}$$

Similarly, optimal policies also share the same optimal action-value function q_*:

Definition 6 (Optimal Action-Value Function) *For a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities p, and discount factor γ, the optimal action-value function q_* is the action-value function that maximizes the expected return for all states and actions:*

$$q_*(s,a) := \max_{\pi} q_\pi(s,a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.18}$$

Thus, for the state-action pair (s,a), we can derive $q_*(s,a)$ in terms of the optimal value function v_*:

$$\begin{aligned} q_*(s,a) &= \max_{\pi} q_\pi(s,a) \\ &= \max_{\pi} \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \max_{\pi} \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \qquad \text{(by 2.10)} \\ &= \sum_{r,s'} p(s',r|s,a) \left[r + \gamma \max_{\pi} \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] \\ &= \sum_{r,s'} p(s',r|s,a) \left[r + \gamma v_*(s')\right] \\ &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \end{aligned} \tag{2.19}$$

Since v_* is still a value function for a policy, it must satisfy the Bellman equation in 2.14. The difference is now that since v_* is the optimal value function, we can write the Bellman equation without a reference to any policy, i.e. the Bellman optimality equation. Intuitively, this equation states that the value of a state under the optimal policy must equal the expected return for the best action from that state:

$$v_*(s) = \max_{a \in \mathcal{A}} q_{\pi_*}(s,a) \tag{2.20}$$
$$= \max_{a \in \mathcal{A}} \mathbb{E}_{\pi_*}[G_t|S_t = s, A_t = a]$$
$$= \max_{a \in \mathcal{A}} \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \qquad \text{(by 2.10)}$$
$$= \max_{a \in \mathcal{A}} \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{2.21}$$
$$= \max_{a \in \mathcal{A}} \sum_{r,s'} p(s',r|s,a) \left[r + \gamma v_*(s')\right]. \tag{2.22}$$

The last two equations are the two forms of the Bellman optimality equation for v_*. As a result, substituting Equation 2.20 into Equation 2.19, the Bellman optimality equation for q_* is

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a')|S_t = s, A_t = a] \tag{2.23}$$
$$= \sum_{r,s'} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a')\right]. \tag{2.24}$$

Once the optimal value function v^* is known, determining an optimal policy becomes straightforward. For each state, optimal actions are those that maximize the Bellman optimality equation, and any policy assigning non-zero probability only to these actions is optimal. When using the optimal action-value function q^*, choosing optimal actions is even easier, as it directly provides the best action for each state.

Explicitly solving the Bellman optimality equation offers one way to find an optimal policy, but this approach is not practical. This is because you have to do an exhaustive search, requiring complete knowledge of the environment's dynamics, a lot of computational resources, and that the Markov property holds. In real-world problems, these assumptions are often not true. For instance, some tasks involve an enormous number of states, like chess, making it computationally infeasible to solve for v^* or q^* exactly. As a result, reinforcement learning typically relies on approximate solutions.

2.3.2. Dynamic Programming

Dynamic programming (DP) is a method to solve for optimal policies in Markov decision processes. We continue with our assumption of having a finite MDP, as this is the problem statement we are interested in. DP methods is to use the value function to organise and structure the search for good policies. The general idea of DP is to turn Bellman equations into update rules for improving approximations of the value function.

Policy Evaluation and Improvement

We begin with how to compute the state-value function v_π for an arbitrary policy π. This is called *policy evaluation*. If we know all of the environment's dynamics, then the Bellman equation in 2.14 can be solved directly, as they become a simultaneous linear equations in $|\mathcal{S}|$ unknowns. However, in practice, iterative approximations are used.

Let v_0 be an initial estimate of v_π, chosen arbitrarily. Then, using the Bellman equation for v_π, we can derive an update rule for v_π, namely,

$$v_{k+1}(s) := \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s') \right], \quad \forall s \in \mathcal{S}. \tag{2.25}$$

Sutton and Barto [12] shows that this update rule converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π. This is called the *iterative policy evaluation* algorithm.

The reason to compute the value function of a policy is to find better policies. Suppose we have now the value function v_π for an arbitrary deterministic policy π. For some state s, we know how good it is to follow the current policy, but we ask ourselves if it might be better or worse to change the policy deterministically to an action $a \neq \pi(s)$. One way of doing so is to select a in s and then follow π thereafter. The value of behaving this way is given by the *state-action value function* $q_\pi(s,a)$, which is the expected return for taking action a in state s and then following policy π:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_\pi(s') \right].$$

To evaluate a change in policy, we introduce the *policy improvement theorem*.

Theorem 1 (Policy Improvement Theorem) *Let π and π' be deterministic policies for a Markov decision process with state space $\mathcal{S}$ and action space $\mathcal{A}$. If for all $s \in \mathcal{S}$,*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s),$$

then the policy π' is better than or equal to π. That is,

$$v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}.$$

If the inequality is strict for at least one state s in the first condition, then the inequality is strict for that state in the second condition.

Proof:
Proof found on page 78 in Sutton and Barto [12]. □

So, given a policy and its value function, we can now, with this theorem, evaluate a change in policy at a given state. We extend this to the entire state space, selecting at each state the action that appears best according to $q_\pi(s, a)$. This is called the *greedy* policy, given by

$$\pi'(s) := \arg\max_{a \in \mathcal{A}} q_\pi(s, a) \tag{2.26}$$

$$= \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a\right], \tag{2.27}$$

$$= \arg\max_{a \in \mathcal{A}} \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]. \tag{2.28}$$

The process of making a new policy that improves on the current policy is called *policy improvement*. Suppose that the new greedy policy π' is equal to the old policy π. Then, $v_{\pi'} = v_\pi$, and from 2.26, it follows that for all $s \in \mathcal{S}$:

$$v_{\pi'}(s) = \max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a\right] \tag{2.29}$$

$$= \max_{a \in \mathcal{A}} \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_{\pi'}(s')\right]. \tag{2.30}$$

But, this is the same as the Bellman optimality equation for $v_{\pi'}$, and thus $v_{\pi'} = v_*$, the optimal value function. So, both π and π' are optimal policies. Policy improvement thus must give us a strictly better policy, except for when the original policy is already optimal.

Policy Iteration

Suppose we have a policy π. We improve π using v_π to yield a better policy π'. We then evaluate π' to get $v_{\pi'}$. We can then improve π' to get π'', and so on. Each policy is guaranteed to be a strict improvement over the previous one, unless already optimal, and since we work with finite MDPs, it must converge to an optimal policy in a finite number of steps. This process of iteratively evaluating the policy and then improving the policy is called *policy iteration*. We write a computerised version of the algorithm in Algorithm 1.

Algorithm 1 Policy Iteration

Input: Tolerance TOL, transition matrix P, state space $\mathcal{S}$, action space $\mathcal{A}$, discount factor γ, max iterations N.

Output: Nearly optimal value function V_* and nearly optimal policy π_*.

1: Let $V_0(s) \in \mathbb{R}$ arbitrary for all $s \in \mathcal{S}$.
2: Let $\pi_0(s) \in \mathcal{A}$ arbitrary for all $s \in \mathcal{S}$.
3: Let $n = 1$, $d > TOL$, and `stable_policy` == True.
4: **while** `stable_policy` == True **do**
5: Policy Evaluation:
6: **while** $d > TOL$ or $n > N$ **do**
7: Set

$$V_n(s) = \sum_{s',r} p(s', r|s, \pi(s)) \left[r + \gamma V_{n-1}(s')\right], \quad \forall s \in \mathcal{S}.$$

8: Let $d = \max_{s\in\mathcal{S}} |V_{n-1}(s) - V_n(s)|$
9: $n = n + 1$.
10: **end while**
11: Policy Improvement:

$$\pi_n(s) = \arg\max_{a\in\mathcal{A}} \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_n(s')\right], \quad \forall s \in \mathcal{S}.$$

12: **if** $\pi_n(s) \neq \pi_{n-1}(s)$ **then**
13: `stable_policy` == False
14: **end if**
15: **end while**
16: **return** π_n and V_n.

Sutton and Barto [12] mention that in practice, policy iteration often converges in surprisingly few iterations. Notably, this is something that we also see in our implementation of the algorithm for our use case.

Value Iteration

A major drawback to policy iteration is that it requires a full policy evaluation at every iteration. This can be computationally expensive, especially when the state space is large. What if we only did policy evaluation iteratively? Then, convergence exactly to the v_π only occurs in the limit, and we might be able to stop short of that. The policy evaluation step of policy iteration can be truncated after just one update through the state space, and this is called *value iteration*. The simple update operation in value iteration is given by

$$V_{n+1}(s) := \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_n(s')\right], \quad \forall s \in \mathcal{S}. \tag{2.31}$$

We rewrite the full algorithm for computer implementation in Algorithm 2.

Algorithm 2 Value Iteration

Input: Tolerance TOL, transition matrix P, state space $\mathcal{S}$, action space $\mathcal{A}$, initial value V_0.
Output: Nearly optimal policy π.

1: Let $\pi_0(s) \in \mathcal{A}$ arbitrary for all $s \in \mathcal{S}$.
2: Let $n = 1$ and $d > TOL$.
3: **while** $d > TOL$ **do**
4: Set
$$V_n(s) = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_{n-1}(s')\right].$$
5: Let $d = \max_{s\in\mathcal{S}} |V_{n-1}(s) - V_n(s)|$
6: $n = n + 1$.
7: **end while**
8: Define $\pi : \mathcal{S} \mapsto \mathcal{A}$ as a maximizer to
$$\sum_{s',r} p(s', r|s, a) \left[r + \gamma V_n(s')\right].$$

As we can see the algorithm is much simpler than policy iteration, and it is more computationally efficient in one iteration. We will see in our implementation that value iteration still converges to the same policy as policy iteration, but does so in more iterations, albeit at a lower computational cost per iteration.

2.3.3. Temporal-Difference Learning

A large drawback to dynamic programming methods is that they require a model of the environment, i.e. the transition probabilities p. In many cases, such as ours, this is not available, and we must rely on experience to learn the value function. Temporal-difference (TD) learning is a model-free method that learns from experience, and it is a combination of Monte Carlo methods and dynamic programming methods. TD learning differs over Monte Carlo methods in not requiring the full episode to complete before updating the value function. Instead, it updates the value function after each time step. This makes it more efficient than Monte Carlo methods, as it can learn online, and it does not require the full episode to complete.

We do not go into the full details of TD learning, but we will introduce and explain the Q-learning algorithm as it is the most widely used TD learning algorithm, and the one we will use in our implementation.

Q-Learning

One of the major breakthroughs in reinforcement learning was the development by Watkins in 1989 [13] of an off-policy TD control algorithm called Q-learning. Q-learning is a model-free algorithm that learns the optimal action-value function q_* directly, without requiring a model of the environment and independent of the policy being followed. It is defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right], \tag{2.32}$$

where α is the step-size parameter, and the update is done after each transition $(S_t, A_t, R_{t+1}, S_{t+1})$.

The Q-learning algorithm is shown in Algorithm 3.

Algorithm 3 Q-Learning

Input: Step-size parameter α, discount factor γ, initial state S_0.
Output: Nearly optimal action-value function Q_*.

1: Let $Q(s, a)$ be an array indexed by state and action, arbitrarily initialized.
2: **while** episode not terminated **do**
3: Choose A from S using policy derived from Q (e.g. ϵ-greedy).
4: Take action A, observe R, S'.
5: $Q(S, A) \leftarrow Q(S, A) + \alpha\, [R + \gamma \max_a Q(S', a) - Q(S, A)]$.
6: $S \leftarrow S'$.
7: **end while**

Notable, the Q-learning algorithm is incredibly simple. It is able to be applied online, with minimal computation, and through experience generated from an interaction with an environment, it learns a close approximation to the optimal action-value function, without requiring a model of the environment.

While powerful, Q-learning is not without its own flaws. Namely, due to using a maximum operator in the update rule, it can be prone to overestimation of action values. This is called *maximization bias*.

We quickly show the double Q-learning algorithm as it is a simple extension to Q-learning that can reduce the overestimation bias by using two Q values to estimate each other's Q value, more detail is found in van Hasselt [14]. The algorithm is shown in Algorithm 4.

Algorithm 4 Double Q-Learning

Input: Step-size parameter α, discount factor γ, initial state S_0.
Output: Nearly optimal action-value function Q_*.

1: Let $Q_1(s, a)$ and $Q_2(s, a)$ be arrays indexed by state and action, arbitrarily initialized.
2: **while** episode not terminated **do**
3: Choose A from S using policy derived from $Q_1 + Q_2$ (e.g. ϵ-greedy).
4: Take action A, observe R, S'.
5: With 0.5 probability, update Q_1:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left[R + \gamma Q_2(S', \arg\max_a Q_1(S', a)) - Q_1(S, A) \right].$$

6: With 0.5 probability, update Q_2:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left[R + \gamma Q_1(S', \arg\max_a Q_2(S', a)) - Q_2(S, A) \right].$$

7: $S \leftarrow S'$.
8: **end while**

2.3.4. Deep Learning

With the rise in computational power, we have seen artificial neural networks (NNs) become a popular tool in many applications. They are widely used for nonlinear function approximation, in part supported by universal approximation theorems, with the arbitrary width and sigmoid activation function case proven by Cybenko [15] and the multilayer feed-forward network case by Hornik, et al. [16].

Following Hastie, et al. [17], we dive a bit deeper into the most widely used 'vanilla' neural net, also called the single hidden layer back-propagation network or single layer perceptron. For a further in-depth understanding of neural networks and the different architectures one can utilise, we refer to Hastie, et al. [17] and Goodfellow, et al. [18].

A neural network is a parameterised family of functions $f : \mathbb{R}^d \mapsto \mathbb{R}$. Typically, they are represented by a network diagram, such as the one in Figure 2.7. The output layer has k target measurements $Y_k, k = 1, \ldots, K$. The input layer has p input measurements $X_p, p = 1, \ldots, P$. Derived features $Z_m, m = 1, \ldots, M$ are created from linear combinations of the inputs X_p, and then the target Y_k is modeled as a function of linear combinations of Z_m,

$$\begin{aligned} Y_k &= \sum_{m=1}^{M} w_{km}^{(2)} Z_m + b_k^{(2)} \\ Z_m &= \sigma\left(\sum_{p=1}^{P} w_{mp}^{(1)} X_p + b_m^{(1)}\right), \end{aligned} \tag{2.33}$$

where $\sigma(\cdot)$ is called the activation function, and $w_{ij}^{(l)}$ and $b_j^{(l)}$ are the weights and biases of the network. The weights and biases are usually notationally collected in a parameter vector

$$\theta = \left(w_{11}^{(1)}, \ldots, w_{1P}^{(1)}, b_1^{(1)}, \ldots, w_{1M}^{(1)}, \ldots, w_{1M}^{(2)}, \ldots, b_K^{(2)}\right).$$

We do this for ease of notation.

The activation function σ is typically a non-linear and monotonically non-decreasing function. Examples include

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$,
- **Hyperbolic tangent:** $\sigma(x) = \tanh(x)$,
- **Rectified linear unit (ReLU):** $\sigma(x) = \max(0, x)$.

The weights and biases are learned through a process called backpropagation, where the error of the network is calculated and propagated back through the network to update the weights and biases. The error is measured by a loss function, such as the mean squared error, and the weights and biases are updated typically through stochastic gradient descent.

This thesis uses a ReLu activation function, mean squared error loss as its loss function, and optimises with the well known adaptive moment estimation (Adam) algorithm, a form of a stochastic gradient descent algorithm developed by Kingma and Ba [19]. We will re-iterate this in the implementation section.

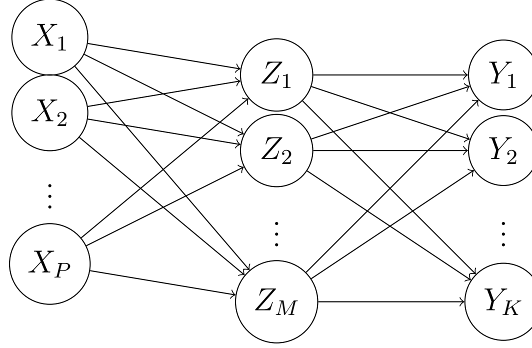

Figure 2.7: Schematic of a single hidden layer, feed-forward neural network. This network has P input nodes, M hidden nodes, and K output nodes.

Deep Reinforcement Learning

Most problems are too large to learn all the action values in all states separately, in a table à la classical Q-learning. Thus, we can represent an approximate value function as a parametrized form with weight vectors $\theta \in \mathbb{R}^d$. We write, $\hat{v}(s,\theta) \approx v_\pi(s)$, and $Q(s,a;\theta) \approx Q^*(s,a)$. There are a multitude of ways to approximate the value function, such as linear function approximation, kernel-based function approximation, and neural networks. The latter has become a popular method, and is the subject of this section, deep reinforcement learning.

Deep Q-Networks

We start with parameterised Q learning. By parameterising $Q(s,a;\theta_t)$, the standard Q-learning update rule after taking action A_t in state S_t and observing reward R_{t+1} resulting in state S_{t+1} in Equation 2.32 becomes

$$\theta_{t+1} = \theta_t + \alpha \left[Y_t^Q - Q(S_t, A_t; \theta_t) \right] \nabla_\theta Q(S_t, A_t; \theta_t), \tag{2.34}$$

where α is the scalar step size, and the target Y_t^Q is defined as

$$Y_t^Q = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta_t). \tag{2.35}$$

Now, a deep Q-network (DQN), developed by Mnih, et al. [4] is when a multi-layered neural network with parameters θ is used to approximate the action-value function $Q(s,a;\theta)$. For an n dimensional state space, and an m dimensional action space, the network is a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$. The Q network is trained by minimising a sequence of loss functions $L_t(\theta_t)$ that changes at each iteration t,

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'} \left[\left(Y_i^Q - Q(s,a;\theta_t) \right)^2 \right], \tag{2.36}$$

where Y_t^Q is the target network. Unlike the target network defined in Equation 2.35, Mnih et al. [4] use a separate target network with parameters θ^-, which is the same as the online network, *except* that its parameters are updated only every τ steps, where $\theta_t^- = \theta_t$. The target network by DQN is then

$$Y_t^Q = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta_t^-). \tag{2.37}$$

The second major innovation of DQN is the use of an experience replay buffer, first developed by L. Lin [20]. The experience replay buffer is a large memory of past experiences $\mathcal{D} = e_1, \dots, e_N$, where $e_t = (s_t, a_t, r_t, s_{t+1})$. During the inner loop of the algorithm, the agent samples a random minibatch from this memory $\mathcal{D}$. The randomization of sampling from this experience replay buffer breaks the correlation between consecutive samples, and therefore reduces variance of the updates.

The DQN algorithm is shown in Algorithm 5.

Algorithm 5 Deep Q-Network

Input: Step-size parameter α, discount factor γ, initial state S_0, replay memory ($\mathcal{D}$) capacity N, target network update frequency τ.

Output: Nearly optimal action-value function Q_*.

1: Let θ and θ^- be the parameters of the online and target networks, respectively, arbitrarily initialized.
2: **for** episode in episodes **do**
3: Initialize S.
4: **while** episode not terminated **do**
5: Choose A from S using policy derived from Q (e.g. ϵ-greedy).
6: Take action A, observe R, S'.
7: Store (S, A, R, S') in $\mathcal{D}$.
8: Sample a random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$.
9: Set

$$Y_i^Q = \begin{cases} R_i & \text{if episode terminates at step } i+1, \\ R_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta^-) & \text{otherwise.} \end{cases}$$

10: Perform a gradient descent step on $(Y_i^Q - Q(s_i, a_i; \theta))^2$ with respect to the network parameters θ.
11: Every τ steps, update the target network parameters $\theta^- = \theta$.
12: $S \leftarrow S'$.
13: **end while**
14: **end for**

Double Deep Q-Networks

Similar to the problem of Q-learning, deep Q-networks can also suffer from overestimation bias. van Hasselt, et al. [21] showed that DQN suffers from substantial overestimations in some games in the Atari 2600 domain. This led to the development of the double DQN algorithm, which is a simple extension to DQN that can reduce the overestimation bias.

The idea behind double Q-learning initially is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation. The parameterised version of the target in Q-learning is rewritten as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t).$$

The double Q-learning error can then be written as

$$Y_t^{DQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t'). \tag{2.38}$$

Note that the selection of the action is still done with the online network, through the parameters θ_t. However, we use a second set of weights θ_t' to fairly evaluate the value of this policy.

We now combine the concepts of double Q-learning and DQN to get double DQN. The target network in DQN (Y_t^Q) provides a natural candidate for the second value function, despite not being fully decoupled. Therefore, the authors propose to use evaluate the greedy policy according to the online network, but use the target network to evaluate estimate its value. The update rule is the same as DQN (see Equation 2.34), but with the target network defined as

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-). \tag{2.39}$$

The difference between DQN and the double Q-learning Equation 2.38 is that the weights of the second network θ_t' are replaced with the weights of the target network θ_t^-. The target network stays unchanged from DQN, and remains a periodic copy of the online network, a subtle difference.

2.4. Reinforcement Learning in Limit Order Book Market Making

With the core principles of reinforcement learning and limit order books established, we now explore the application of RL in the context of market making. In this setting, agents learn to optimize market making strategies by interacting with the LOB, dynamically adjusting their orders to maximize long-term profits while minimizing risks. We will cover the problem of optimal market making in the MDP framework, discuss previous work on RL in market making, and end off with the design choices for our experiments.

2.4.1. Optimal Trading Strategies

Using the theory of Markov decision processes, we can construct optimal strategies for trading strategies. We use two examples covered in Hult and Kiessling [5] of optimal strategies that can be constructed using the theory of Markov decision processes. The first example is a strategy for buying one unit, and the second example is a strategy for market making.

Hult and Kiessling's Theory

We first need to dive a bit deeper into the way Hult and Kiessling write out their theory on optimal trading strategies. They also use Markov decision processes, but the biggest difference is that the split the action state $\mathcal{A}$ into two, a set of continuation actions $\mathcal{C}$ and a set of termination actions $\mathcal{T}$.

Let $(X_n)_{n=0}^{\infty}$ be a Markov chain in discrete time on a countable state space $\mathcal{S}$ with transition matrix P. Let $\mathcal{A}$ be the finite set of possible actions. Every action can be divided into two sets, the set of continuation actions $\mathcal{C}$ and the set of termination actions $\mathcal{T}$. The action space is then defined as

$$\mathcal{A} = \mathcal{C} \cup \mathcal{T}, \quad \mathcal{C} \cap \mathcal{T} = \emptyset.$$

The Markov chain terminates when a termination action is taken.

Every action is not necessarily available in every state. Let $\mathcal{A}(s)$ be the set of actions available in state s. The set of continuation actions is denoted $\mathcal{C}(s) = A(s) \cap \mathcal{C}$ and the set of termination actions is denoted $\mathcal{T}(s) = A(s) \cap \mathcal{T}$.

As a result, there are values of continuation $v_C(s, a)$ and termination $v_T(s, a)$ for each state-action pair (s, a). Hult and Kiessling [5] prove that under this framework, that policies under this framework still have existing optimal policies and have that the optimal expected value function is the unique solution to the Bellman equation.

This does not change much of the actual underlying theory of Markov decision processes, but it does make the notation a bit more complex. The authors still use the value iteration algorithm to find the optimal value function.

Keep or Cancel Strategy for Buying One Unit

We aim to buy a single unit at price $j_0 < j^A(X_0)$, where X_0 is the initial state of the order book X_n. After each market transition, the agent can choose to either keep the limit order or cancel it and submit a market buy order at the best ask level $j^A(X_n)$. We also assume that the agent has decided upon a maximum price level $J > j^A(X_0)$ they are willing to pay. If $j^A(X_i)$, for some time i, reaches J before the agent's order is executed, the agent cancels the bid order and places a market order at J to fulfill the trade. It is assumed there is always volume at J.

We assume J for two reasons, one is that the agent does not want to pay more than J for the asset as it is then comparatively overpriced. The second reason is that by introducing an upper bound, we can ensure that the numerical computation of a solution is made simpler, as we shrink the state space.

Denote Y_n the position of the limit order of the agent in the queue at level j_0. This represents the number of limit orders in front of the agent's order, including the agent's order, after n transitions. Then, $Y_0 = X_0^{j_0} - 1$ (Y_n is negative) and Y_n can only move up towards 0 whenever there is either a market order at level j_0 or a limit order at level j_0 is cancelled. The agent's order is executed when $Y_n = 0$.

The pair (X_n, Y_n) is a Markov chain with state space $\mathcal{S} \subset \mathbb{Z}^d \times \{\dots, -2, -1, 0\}$, i.e. the volume at the levels $\times$ the queue size. The transition matrix of the jump chain is denoted $P = P_{ss'}$. Let $s = (x, y) \in \mathcal{S}$. There are three possible cases:

- $y < 0$ and $j_A(x) < J$, i.e. the agent's order has not been executed and the stop-loss has not been reached. Then, the possible continuation action is waiting for the next market transition, denoted by $\mathcal{C}(s) = \{0\}$. The possible termination action an agent can take is to cancel the order and submitting a market order at $j^A(x)$, denoted by $\mathcal{T}(s) = \{-1\}$.
- $y < 0$ and $j_A(x) = J$, i.e. the agent's order has not been executed and the stop-loss has been reached. Then, the possible continuation action is $\mathcal{C}(s) = \emptyset$, as we hit our stop loss. The possible termination action is $\mathcal{T}(s) = \{-1\}$, representing cancelling the limit order and submitting a market order at J.
- $y = 0$. The Markov chain is terminated, as the agent's order has been executed. The continuation action is $\mathcal{C}(s) = \emptyset$ and the termination action is denoted by $\mathcal{T}(s) = \{-2\}$.

As a result, the termination costs are

$$
\begin{aligned}
v_T(s,-1) &= \pi^{j^A(x)}, \\
v_T(s,-2) &= \pi^{j_0}.
\end{aligned}
$$

We note that this notation differs from the π in the RL section, where π denoted a policy, and here we go back to the notation before where π is the price of the asset at level j.

In a state $s = (x, y)$, when $j_A(x) < J$, and following a stationary policy α, it is given by Lemma 4.1 in Hult and Kiessling [5] (found in Appendix A) that the expected value of the agent's order is given by

$$
V_\infty(s,\alpha) = \begin{cases} \sum_{s' \in \mathcal{S}} P_{ss'} V_\infty(s',\alpha), & \alpha(s) = 0 \\ \pi^J - \pi^{j^A(x)}, & \alpha(s) = -1 \\ \pi^J - \pi^{j_0}, & \alpha(s) = -2. \end{cases}
$$

The waiting value is zero, as the agent does not pay any fees for waiting. The value function iteration becomes

$$
\begin{aligned}
V_{n+1}(s) &= \max\left(\max_{a \in \mathcal{C}(s)} v_C(s,a) + \sum_{s' \in \mathcal{S}} P_{ss'}(a) V_{n-1}(s'), \max_{a \in \mathcal{T}(s)} v_T(s,a) \right) \\
&= \begin{cases} \max\left(0 + \sum_{s' \in \mathcal{S}} P_{ss'}(0) V_{n-1}(s'), \pi^{j^A(x)}\right), & \text{for } y > -, j^A < J \\ \pi^J - \pi^{j^A(x)}, & \text{for } y > 0, j^A = J \\ \pi^J - \pi^{j_0}, & \text{for } y = 0. \end{cases}
\end{aligned}
$$

One can then use value iteration to solve numerically for a value function $V_\infty(s)$ and a policy $\alpha(s)$ that maximizes the expected value of the agent's order. We have now shown how to use the flexible Markov decision process framework to construct an optimal strategy for buying one unit in the limit order book.

Optimal Market Making Strategy

We now find the optimal strategy for the instance that the agent places two limit orders (one ask one bid) and attempts to *make the spread*, i.e. market make. Let the bid be placed at level j_n^0 with queue position Y_n^0 and the ask be placed at level j_n^1 with queue position Y_n^1. Our extended Markov chain is now redefined as $(X_n, Y_n^0, Y_n^1, j_n^0, j_n^1)$. It follows that $Y_0^0 = X_0^{j_n^0} - 1$ and $Y_0^1 = X_0^{j_n^1} + 1$, where Y_n^0 is non-decreasing and Y_n^1 is non-increasing. Again, the agent's individual orders are executed if $Y_n^0 = 0$ or $Y_n^1 = 0$.

Furthermore, the agent has decided on a best buy level $J^{B^0} < j^A(X_0)$, a worst buy level $J^{B^1} > j^A(X_0)$, a best sell level $J^{A^1} > j^B(X_0)$ and a worst sell level $J^{A^0} < j^B(X_0)$. The state space is thus $\mathcal{S} \subset \mathbb{Z}^d \times \{\dots,-2,-1,0\} \times \{0,1,2,\dots\} \times \{J^{B^0},\dots,J^{B^1}-1\} \times \{J^{A^0}+1,\dots,J^{A^1}\}$. The possible actions in this strategy are defined as:

1. Before any of the orders are executed, the market maker can choose from
   - Waiting for the next transition.

- Cancel both orders and resubmit at new levels k^0 and k^1.
- Cancel one order and resubmit at new level k^0 or k^1.

2. After one of the orders is executed, the outstanding limit order is proceeded according to the buy(sell)-one-unit strategy. And the price level is also renewable after each market transition.

Let $V_\infty^B(x, y, j)$ denote the optimal expected buy price in state (x, y, j), with best buy level J^{B^0} and worst buy level J^{B^1}. Let $V_\infty^A(x, y, j)$ denote the optimal expected sell price in state (x, y, j), with best sell level J^{A^0} and worst sell level J^{A^1}. The optimal expected value is then given by

$$V_\infty(s) = \begin{cases} \max\left(\sum_{s' \in \mathcal{S}} P_{ss'} V_\infty(s'), \max V_\infty(s_{k^0 k^1}), 0\right), & \text{for } y^0 > 0, y^1 > 0 \\ \pi^{j^1} - V_\infty^B(x, y^0, j^0), & \text{for } y^0 > 0, y^1 = 0 \\ V_\infty^A(x, y^1, j^1) - \pi^{j^0}, & \text{for } y^0 = 0, y^1 > 0 \end{cases}$$

where $s_{k^0 k^1}$ is the state where the agent has cancelled both orders and resubmitted at levels k^0 and k^1.

Again, this can be solved for numerically using value iteration to find the optimal value function $V_\infty(s)$ and the optimal policy $\alpha(s)$ that maximizes the expected value of the agent's orders. We have now shown how to use the flexible Markov decision process framework to construct an optimal strategy for market making in the limit order book.

The state and action space that we end up using in this thesis differs from the one above, where inherently the order book is part in the state space. Furthermore, we simplify the action space. This is done to make the problem computationally feasible to solve, and will be discussed in the next section.

2.4.2. Previous Work

This section covers previous work on reinforcement learning in the market making problem. Most of the next section is based on Gašperov et al. [22], as they have done an extensive literature review on the topic, as well as the Master's thesis of KTH students Carlsson and Regnell [23].

State Space Representation

The design of the state space is one of the most important aspects of RL. It can determine how well the agent can learn the optimal policy. The state space can be represented in many ways, such as the order book, the queue position, the spread, the mid price, the volume, and the volatility. The state space can be continuous or discrete, and the choice of representation can have a significant impact on the performance of the RL agent.

Inventory

Inventory-based models are the most common approach to base the state space, with 91% of the articles Gašperov et al. [22] reviewed using this approach. The inventory is the number of shares the agent holds, and inherently market making boils down to an inventory management problem, and so it is not surprising that this is the most common approach. Why? As the agent's inventory increases, the agent's risk increases, and the agent inherently starts to take a larger and longer term position in the asset, the opposite of what a market maker wants to do.

The inventory can be represented in multiple ways. Chan and Shelton [24] let inventory at time t be $q_t \in \{q_-, \ldots, q_+\}$, where q_- is the maximum short inventory and q_+ is the maximum long inventory allowed. The inventory can also be binned, to ease computation complexity by reducing the state space. An example is done by Lim et al. [25], where the inventory is binned into seven states: small, medium, large inventories, in either long or short, and a zero inventory state.

Time
Another notable state space variable is time, which also plays an important part in Avellaneda and Stoikov's paper [2]. Carlsson and Regnell [23] point out to numerous papers that use either the time remaining or the time passed as a variable.

Price
A natural candidate is the price of the asset, as this is what the agent ultimately ends up trading. The price can be represented in many ways, such as the mid price, the spread, the best bid and ask, the volume at the best bid and ask, and the volatility, which is usually derived from the price. Spooner et al. [26] point out numerous measures.

LOB Data
Another common approach is to use the limit order book itself as part of the state space. This is the approach that Hult and Kiessling [5] have implicitly been using when solving for optimal market making. There are many features that one can pull from the LOB, such as the volume at each level and the queue position. One can also calculate the imbalances in volumes on bid/asks, which Spooner et al. [26] uses as a state variable.

Action Space Representation

The action space is also important, as here we can capture the dynamic of how we want the agent to react. Most of what we are concerned with, namely market making in a limit order book, has to do with where the agent actually ends up placing the best bid/asks at that time t.

Price Level
Naturally, the price level itself is a common action space variable. But, implementing all the minimum price increments could mean an action space of up to several thousand different actions for some assets. Thus, Hult and Kiessling [5] choose to have their actions at a bid depth (d_b) and ask depth (d_a) away from the best ask and bid respectively.

To explain again, taking $d_b = 1$ would be the first tick away from the best ask, $d_b = 2$ the second tick away from the best ask, and so on. The same goes for the ask side. This is a bit unintuitive, but is a simple way to allow for the agent to improve upon the already existing best bid/ask. An example would be if the best bid is at level 1, the best ask at level 3. Now, the agent can place a new bid at $d_b = 1$, implying that the agent places a bid at level 2, becoming the new best bid.

Naturally, this also has its drawbacks. If we had a book depth of 10, this will imply $10 \times 10 = 100$ different actions the agent can take, as we place both a bid and ask, with some orders being completely unrealistic, i.e. a bid and ask at depth 10. This is computationally infeasible, and so a popular approach is to select an action to be chosen among a predetermined set of bid and ask pairs. Spooner et al. [26] use a set of 10 actions, namely

Action ID	0	1	2	3	4	5	6	7	8
Bid	1	2	3	4	5	1	3	2	5
Ask	1	2	3	4	5	3	1	5	2
Action ID 9	Market Order with Size $-q$								

Table 2.2: Action space of 10 actions, used in Spooner et al. [26]. The bid and ask number represent the level in the limit order book. The last action is to clear the inventory with a market order.

Commonly, the volume that is placed at these levels are fixed, which is the approach we resort to using to make the problem computationally feasible.

Market Order

As shown above, Spooner et al. [26] also include a market order action, which is to clear the inventory. This is a common approach, as it is a way to ensure that the agent can clear the inventory if it is too large.

Reward Function

Reward functions are always some form of profit and loss, whether adjusted to inventory (and other factors) or not. This is logical, as the ultimate aim of the market maker is to make a profit. However, there are numerous ways to formulate PnL, and we go over two of them.

Markt-to-Market PnL

Mark to market PnL is the most common approach, where the agent's PnL is calculated at each time step, i.e. the agents previous inventory is multiplied by the change in where the previous price was and is now. This is one of the approaches taken by Spooner et al. [26], where the PnL is calculated as the difference between the agent's inventory and the mid price. Namely, the mark to maker PnL is defined as

$$r_t^{pnl} = (m_t - p_{t-1}) \cdot q_{t-1},$$

where m_t is the mid price at time t, p_{t-1} is the mid price at time $t-1$, q_{t-1} is the agent's inventory at time $t-1$, and $r_0^{pnl} = 0$.

A problem with this is that there is no penalty for inventory holding, the agent could theoretically be incentivised to hold a large inventory and take risk in where the price is moving. This is why the next approach is was developed.

Asymmetrically Dampened PnL

Spooner et al. [26] show in their paper that the basic formulation of mark to market PnL ignores the specific objective of a market maker. They propose a new reward function that is asymmetrically dampened, where the agent is penalized for holding inventory. The reward function is defined as

$$r_t^{pnl_A} = r_t^{pnl} - \max\{0, \eta \cdot |q_t| \Delta m_t\},$$

where η is a parameter that determines the penalty for holding inventory, q_t is the agent's inventory at time t, and Δm_t is the change in the mid price from time $t-1$ to time t. Intuitively, this reduces the reward the agent can gain from pure speculation.

2.4.3. Experimental Setup

We now turn our attention to which algorithms we decide to test, how this thesis chooses its environment choices from the selection above, and also why we chose them.

Algorithms

The ultimate list of algorithms used and considered in each environment is listed in the table below.

	Value-iteration	Policy-iteration	Q-learning	DQN	DDQN
Markov chain model	✓	✓	✓	✓	✓
Queue-reactive model	✗	✗	✓	✗	✓

Table 2.3: Algorithms used in experiments.

We do not consider value and policy iteration, and DQN for the queue-reactive model. Mostly because computation time is valuable, and we want to focus on the algorithms that are most likely to perform well in the environment.

State Space Representation

In this thesis, we take the state space as a mixture of inventory and time, namely $\mathcal{S} = \mathcal{T} \times \mathcal{Q}$. The inventory is binned into 3 intervals, $q \in \mathcal{Q} = \{-1, 0, 1\}$, where if the agent is long the asset, $q = 1$, if the agent is short the asset, $q = -1$, and if the agent is flat, $q = 0$. This is done to reduce the state space, but also to have some form of indication whether the agent is generally long or short the asset.

In our experiments, an episode lasts $T = 1000$ where each time step $dt = 1$. The time is binned into 5 intervals of length 200, namely $t \in \mathcal{T} = \{1, 2, 3, 4, 5\}$, with 1 being the first bucket of time, and 5 being the last time bucket of the episode. Intuitively, this was done to have some form of indication of how long the agent has to offload its risk before the end of the trading day.

Thus, the state space is then

$$\mathcal{S} = \mathcal{T} \times \mathcal{Q} = \{1, 2, 3, 4, 5\} \times \{-1, 0, 1\}.$$

As a small experiment, I decided to include a second state space representation only for DDQN. We call this DDQN full, as now the state space includes the 10 levels of the order book. This is done to see if the agent can learn a more optimal policy with more information. The state space becomes

- Time: $\mathcal{T} = \{1, 2, 3, 4, 5\}$.
- Inventory: $\mathcal{Q} = \{-5, -4, \ldots, 0, \ldots, 4, 5\}$.
- 10 levels of the order book.

Unfortunately, we could not consider the above state space representation for all the algorithms, as the computational complexity would be too high. Training the DDQN on the full state space took around 2 days, and so we left the most complicated state space representation for the relatively most complicated model. However, as we will show, this did not necessarily lead to better performance.

Action Space Representation

Unlike Spooner et al. [26], we do not consider a market order action and also do not consider a predetermined set of actions. I wanted to experiment with the agent's ability to learn the optimal policy by itself, and so we only limit the agent to placing a bid and ask at the best 3 levels on each side, i.e.

$$\mathcal{A} = D_b \times D_a = \{1, 2, 3\} \times \{1, 2, 3\},$$

where D_b and D_a are the set of order depths. We only take three possible levels to reduce the action space. Furthermore, I chose not to include a market order to clear its inventory, as I wanted to push the agent to rely on controlling inventory through the placements of bid and asks.

Another thing to note, is that the agent has to constantly place new bid and asks at each time step. This is partly realistic, as in the real world the agent would have to constantly update its orders, but also not as this takes away the value of having a resting order in the queue, which increases as the queue position increases behind the agent's order. This is a simplification that was made to make the problem computationally feasible.

Reward Function

The reward function used in the training of the numerical methods above is the asymmetrically dampened PnL, as defined above.

Benchmark Strategies

For a baseline policy, we will use what is called at-the-touch market making, taken from Cartea et al. [27]. This is a simple strategy where the agent continuously places a bid and ask at the best bid and ask levels, respectively. If the max inventory is reached, it will only place orders on the opposite side of its inventory, i.e. if it is max long, only place sell orders (asks). We call this agent "Follow BBO".

Do note that this is a bit of an unfair comparison, as the at-the-touch strategy has a different action space than the RL agents. However, this is a strategy used in the literature and is a good benchmark to compare the RL agents to. Unfortunately, we could not get to actually visualising this policy in this thesis, as we would have to average out the actions over many samples, which takes a lot of time.

Performance Criteria

To ultimately evaluate the performance of the agents, we use multiple measures. We examine how long it takes for the agent to converge to a policy, the mean absolute inventory over time, the different policies the agent has learned, and the mean mark to market PnL over time.

Why use mark to market PnL? Because in the 'real world', this is ultimately what the agent, or an actual market maker, is trying to optimize. The asymmetrically dampened PnL is used in training to ensure that the agent learns not take on too much risk, but the mark to market PnL is used to evaluate the agent's performance.

Value Iteration and Policy Iteration

Using value and policy iteration inherently implies that one knows the probability transition matrix. I could not find the probability transition matrix from the Markov chain model. Thus,

I resorted to empirically approximating the probability transition matrix from the environment. This is done by taking an agent who would do a random action from the above action space and counting how many times the state transitions from on to another based on this. This is done for a large number of episodes, and then the probability transition matrix is calculated by dividing the number of times a transition happens by the total number of transitions. This is then put into the value iteration and policy iteration algorithms, and the optimal policy for that probability matrix found.

This is a form of a hybrid model-based reinforcement learning, as we are using the model of the environment to find the optimal policy, but not quite using the analytical formula for the probability transition matrix (if it exists).

3 Experiments and Results

With the foundational concepts of reinforcement learning and the mechanics of limit order books established, as well as how we will approach our version of solving the market making problem, we now turn our attention to the practical implementation of the trading environment. In this section, we will detail the process of building a simulated trading environment tailored to our experiments. We will then show the hyperparameters used. Finally, we will present and discuss the results of these experiments, namely under the Markov chain model and the queue-reactive model, analyzing the agent's performance and the impact of various factors on the market making policies.

3.1. Trading Environment

The implementation of the whole trading framework was done in Python. The level 3 limit order book was implemented from scratch, and uses a sorted dictionary of dictionaries to keep tabs on price levels and orders. The self made trading environment was built on top of the popular Gymnasium library [28], a maintained fork of OpenAI's Gym library, which is a toolkit for developing and comparing reinforcement learning algorithms. The machine learning package used for the deep reinforcement learning aspect was the PyTorch package [29].

The computer used for the experiments was a personal computer with an AMD Ryzen 5 5600x CPU, 32GB of RAM, and an NVIDIA RTX 980Ti GPU. Note that despite sounding impressive, the computer specifications used were not necessarily the greatest hardware for training deep reinforcement learning models, but it was borderline sufficient for the purposes of this research.

3.2. Parameters

We will now discuss the model's parameters used in the experiments. We will first discuss the trading environment parameters, followed by the training parameters for the agents, and finally the test parameters used to evaluate the agents' performance.

3.2.1. Training Parameters

We reveal the parameters chosen for training. This was chosen based on a mixture of computational time, previous work, and experimentation.

Environment

The training parameters used in the experiments are listed in the table below. The training parameters are consistent across all experiments, and these are the environment variables that the agents interact with.

Parameter	**Value**
n_episodes	10000
T	1000
dt	1
Tick size	1
S_0	100
q_0	0
Reward function	Asymmetrically Dampened PnL

Table 3.1: Training parameters used in experiments.

The dampening factor η used was equal to 0.5.

Agents

We outline the parameters used in each agent's training process. Many of these parameters were chosen based on ad-hoc experimentation and are not necessarily optimal. Some of these things, such as the learning rate of the optimizer in the neural network, are difficult to tune. We note that this is something that could be improved upon in future work.

Value Iteration

As noted before, the implemented value and policy iteration uses a probability transition matrix simulated from the environment. We use 100000 episodes with 1000 steps to calibrate the transition matrix.

The parameters used in the value iteration agent's training process are listed in the table below.

Parameter	**Value**
γ	0.9
ϵ	$1e-6$

Table 3.2: Value Iteration training parameters.

Policy Iteration

The parameters used in the policy iteration agent is the same as the value iteration agent.

Q-Learning

The parameters used in the Q-Learning agent's training process are listed in the table below.

Parameter	Value
γ	1
Learning rate	0.01
Exploration Rate	$1 \rightarrow 0.05$ with a decay of $\frac{1}{5000}$

Table 3.3: Q-Learning training parameters.

DQN

For DQN agents, we use a neural network with the following architecture:

- Input layer: 2 nodes
- Hidden layer 1: 8 nodes
- Activation function: ReLU
- Hidden layer 2: 16 nodes
- Activation function: ReLU
- Output layer: 9 nodes

One might be surprised at how small the neural network is. However, the state space is relatively small, and the action space is also small. Out of some ad-hoc experimentation, I noticed that larger neural networks did not necessarily perform better, and took longer to train.

The parameters used in the DQN agent's training process are listed in the table below.

Parameter	Value
Buffer size	1000
Gradient clip	1
Gradient repeat	1
γ	1
Learning rate	0.01
Batch size	64
Optimizer	Adam
Exploration Rate	$1 \to 0.05$ with a decay of $\frac{1}{5000}$

Table 3.4: DQN training parameters.

DDQN

For DDQN agents, we use a neural network with the same architecture as the DQN agent, the target network also has the same architecture. The parameters used in the DDQN agent's training process are listed in the table below.

Parameter	Value
Buffer size	1000
Gradient clip	1
Gradient repeat	1
γ	1
Learning rate	0.001
Batch size	64
Optimizer	Adam
Exploration Rate	$1 \to 0.05$ with a decay of $\frac{1}{5000}$
Target update interval	200
Target update type	Copy

Table 3.5: DDQN training parameters.

3.2.2. Test Parameters

To evaluate the performance of the agents, we use the test parameters listed in the table below. The test parameters are consistent across all experiments.

Parameter	**Value**
n_episodes	1000
T	1000
dt	1
Tick size	1
S_0	100
q_0	0
Reward function	Mark to Market PnL

Table 3.6: Test parameters used in experiments.

3.3. Markov Chain Model

The first experiment uses the Markov chain model that we discussed in Section 2.2.1. We have already discussed the training parameters in Section 3.2. We will now present the results of the experiments.

3.3.1. Training

We begin with the training of the agents in the Markov chain model. We will first present the results of the value iteration and policy iteration, followed by the Q-learning, DQN, DDQN and finish with DDQN Full.

Value Iteration

Beginning with value iteration, we use the empirical transition probability matrix and then run the value iteration algorithm. We first present the Bellman error below with respect to the number of iterations, found in Figure 3.1.

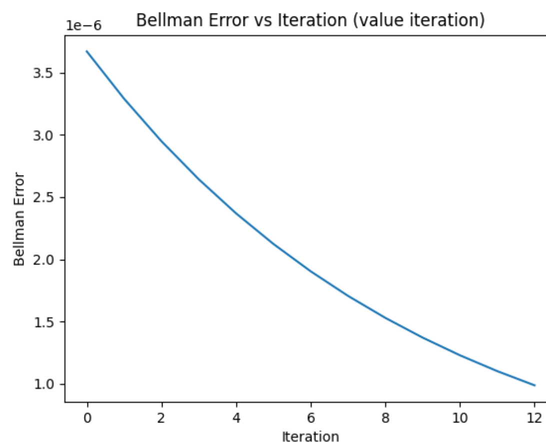

Figure 3.1: Bellman error for value iteration in the Markov chain model setting.

As we can see, the convergence happens after 12 iterations, and the error is less than $1e-6$. The policy that we obtain from this value iteration is shown below in Figure 3.2.

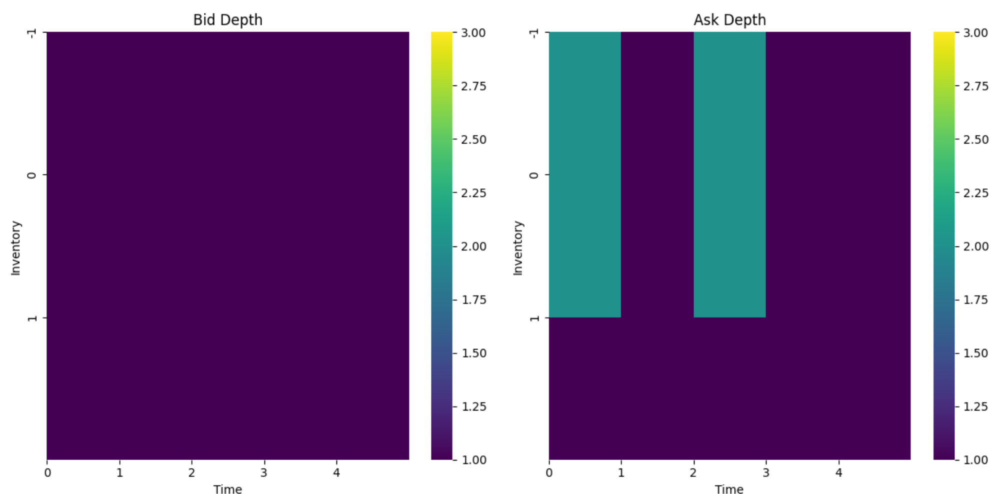

Figure 3.2: Final policy for value iteration in the Markov chain model setting.

We see that the bid depth and the ask depth are unidentical, not something that we initially would have expected, due to the symmetry of the model itself. However, this could be a problem that stems from using an empirical transition matrix. Another noticeable thing is the lack of diversity in actions, namely that in almost all states the agent prefers to take the action with $d_b = 1$ and $d_a = 1$, save for a few states where the ask depth taken is two. This is not what I initially expected, as the policy implies that the agent maximizes its reward and minimizes inventory by constantly trying to improve or join the best bid/ask. This stems likely due to the fact that the actual trading is maximized at the best bid/ask due to the dynamics of the limit order book. This, plus the fact that the orders come in symmetrically imply that being the best bid/ask at all times allows the agent to constantly capture the spread and as a result also minimise inventory.

Policy Iteration

We now present the results of the policy iteration algorithm. Again, we first present the Bellman error below (Figure 3.3).

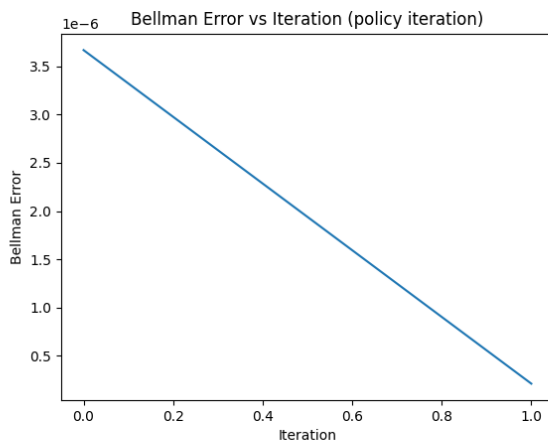

Figure 3.3: Bellman error for policy iteration in the Markov chain model setting.

Noticeably, the policy iteration converges after only a single iteration, compared to the 12 it took in value iteration. This is an observation that Sutton and Barto [12] mention, that policy iteration often converges remarkably quickly. The policy that we obtain from this policy iteration is shown below in Figure 3.4.

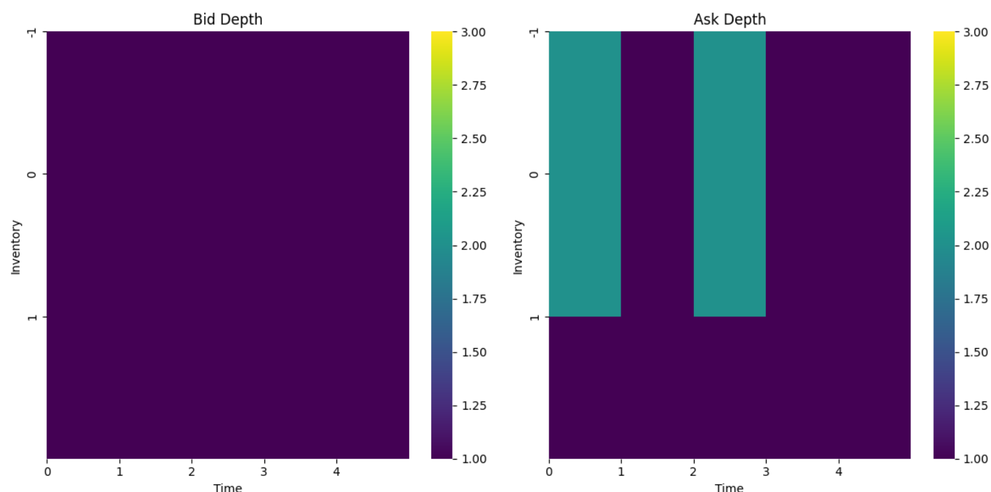

Figure 3.4: Final policy for policy iteration in the Markov chain model setting.

The policy is identical to the one obtained from value iteration, which is roughly expected as in theory the two algorithms should converge to the same policy. The same observations made for the value iteration policy apply here as well.

Q-Learning

We now begin with our first model free approach, Q-learning. We present the rewards against episode, a histogram of the mean absolute position, and the frequency of actions taken obtained by the agent in Figure 3.5.

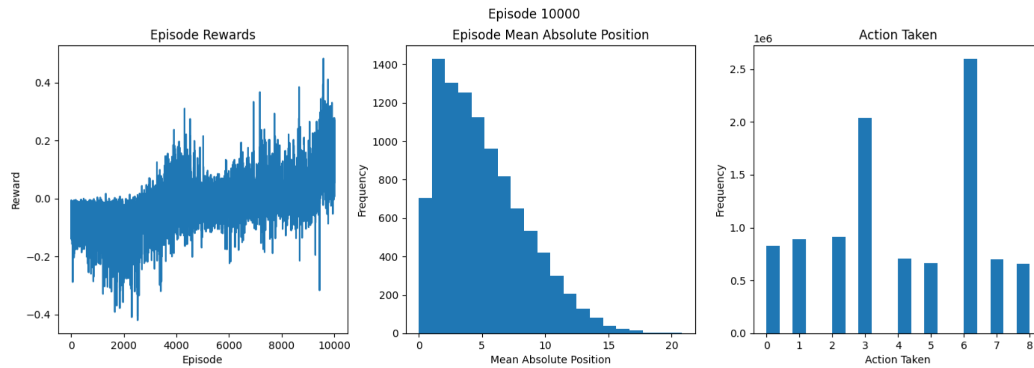

Figure 3.5: Rewards for Q-learning in the Markov chain model setting.

Initially, the rewards are slightly negative, but after episode 4000, it seems that the agent is learning to improve its policy as the rewards start to trend upwards. The histogram of the mean absolute position shows that the agent is able to maintain a position mostly under 5, which is a good sign. However, it does seem to have a relatively high frequency to shoot its position above 10, a sign that there is still a high number of cases the agent is not able to manage its inventory.

Below, we present the Q-values of the agent after 1 episode and 10000 episodes in Figures 3.6 and 3.7 respectively.

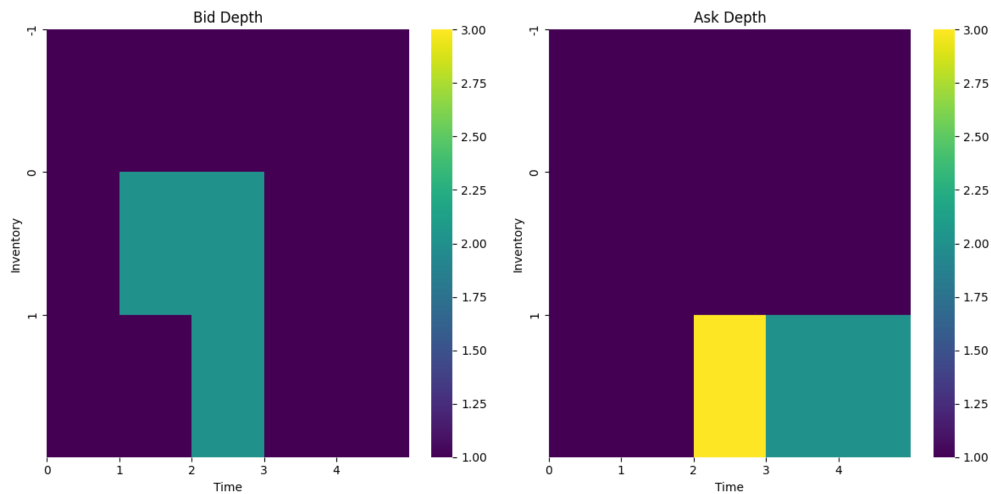

Figure 3.6: Policy Q-learning in the Markov chain model setting after 1 episode.

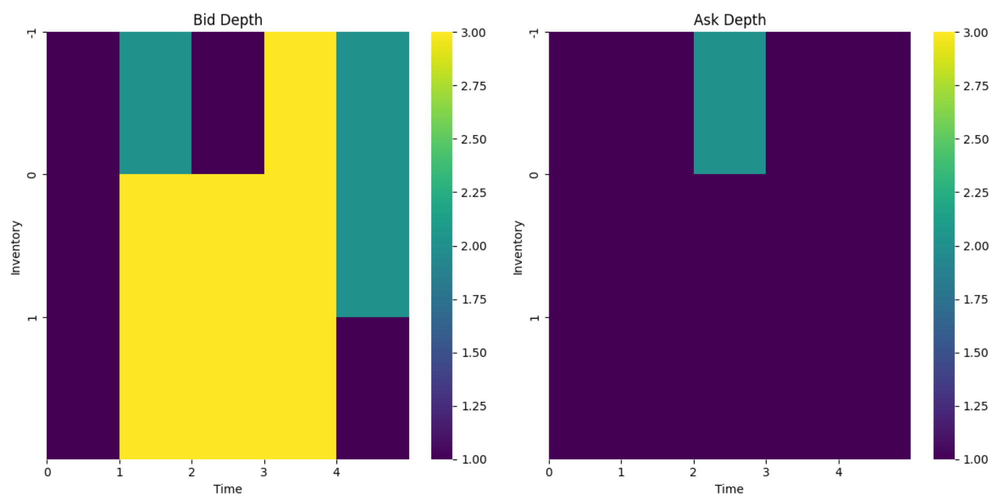

Figure 3.7: Policy Q-learning in the Markov chain model setting after 10000 episodes.

A positive note is that the policy seems to be changing as the agent steps through the episodes. We do now see that the asymmetry in bid and ask depths is more pronounced than the value/policy iteration policies. The bid depths seem to indicate that the agent rarely takes the action $d_b = 1$ and thus we can conclude that there in an imbalance, namely that this policy is more keen on selling than it is on buying the asset. We will discuss this more in depth in the discussion section, but the Q-learning reward graph is a particularly interesting oddity that seems to also come up in the queue reactive model setting.

Deep Q-Network

We now present the results of the DQN agent. We present the rewards against episode, a histogram of the mean absolute position, and the frequency of actions taken obtained by the agent in Figure 3.8.

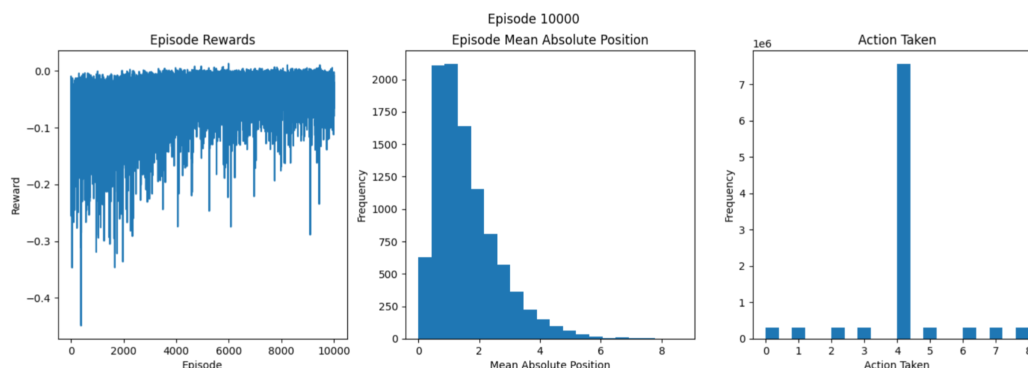

Figure 3.8: Rewards for DQN in the Markov chain model setting.

We first notice that almost all of the rewards are negative, this is something we will expand on in Chapter 4. At first, the rewards are particularly noisy, and it seems that over time the agent is able to reduce the variance of its rewards. Secondly, the histogram of the mean absolute position indicated that the agent is able to maintain a frequently low position of around 2, with outliers only gapping up to 7. This is a good sign that the agent is able to manage its

inventory well. Lastly, the frequency of actions taken shows that the agent takes a liking to one particular action, namely where the agent places its orders at bid depth and ask depth equal to two. We show the policies of the agent after 1 episode and 10000 episodes in Figures 3.9 and 3.10 respectively.

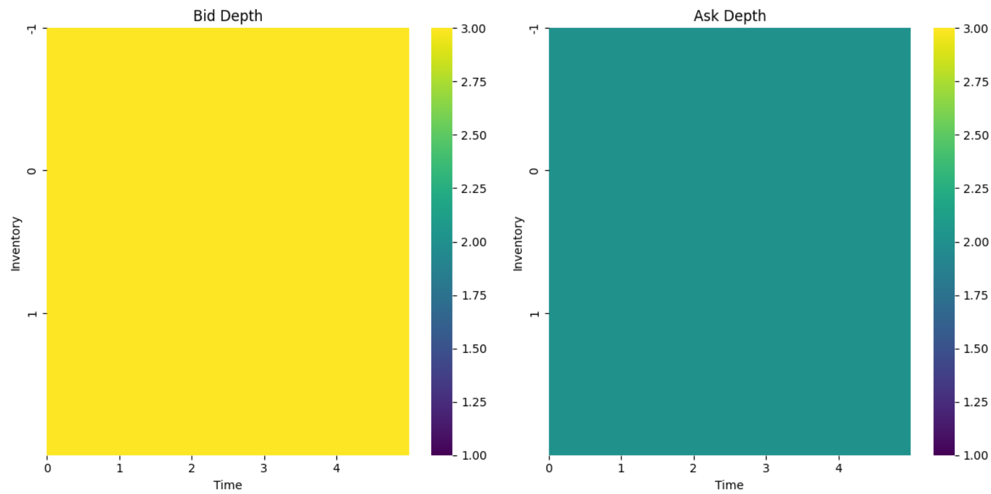

Figure 3.9: Policy DQN in the Markov chain model setting after 1 episode.

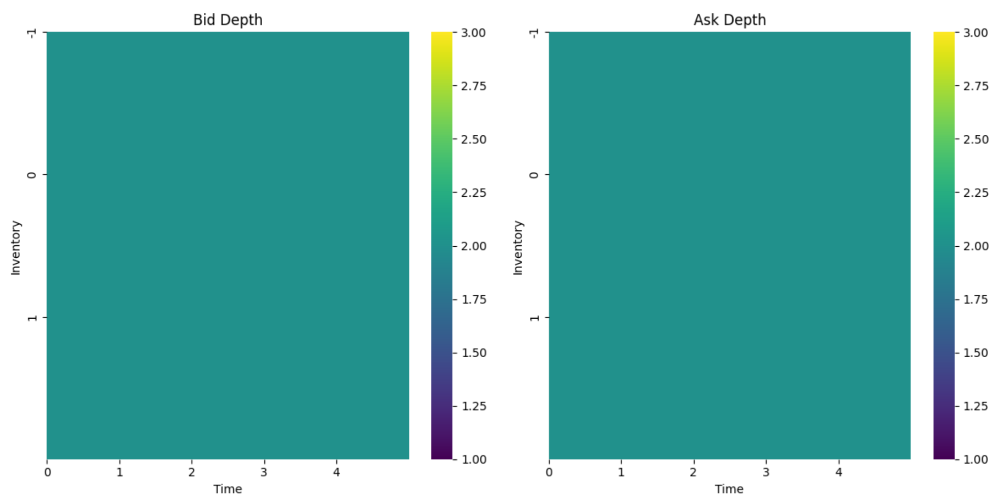

Figure 3.10: Policy DQN in the Markov chain model setting after 10000 episodes.

Here, we see that the policies do not change much from the first policy the agent comes up with. It does improve the bid depth to 2, from only posting a bid depth at 3. Furthermore, we see that the agent takes the same action in all states, namely posting a bid depth and ask depth at 2. We can conclude from the fact that the agent quickly settling on one policy and only doing that one onwards indicates that we are experiencing the very problem we were discussing with DQN in the theory section, namely that DQN has an overestimation bias of its action values.

Double Deep Q-Network

The overestimation bias in DQN is the very reason why we decide to implement the Double Deep Q-Network, of which we present the results below. We present the rewards against episode, a histogram of the mean absolute position, and the frequency of actions taken obtained by the agent in Figure 3.11.

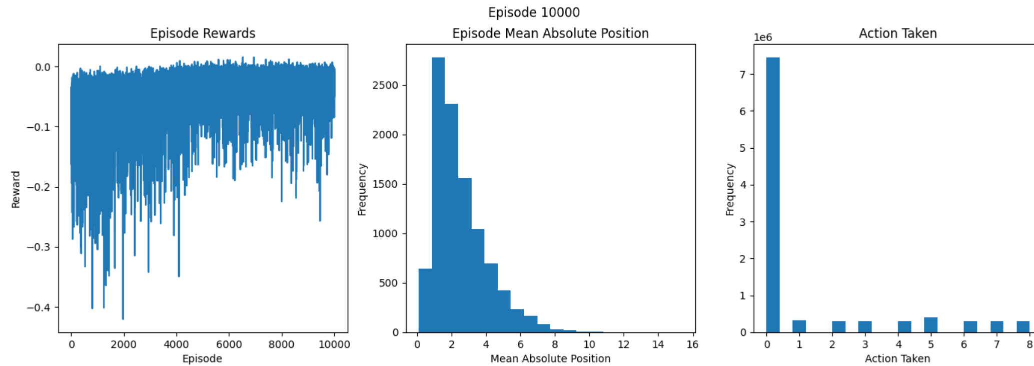

Figure 3.11: Rewards for DDQN in the Markov chain model setting.

The rewards, mean absolute position are comparable to DQN, but the frequency of actions indicate that the agent now settles on the 0th action, placing orders at a depth of 1 on both sides. We show the policies of the agent after 1 episode and 10000 episodes in Figures 3.12 and 3.13 respectively.

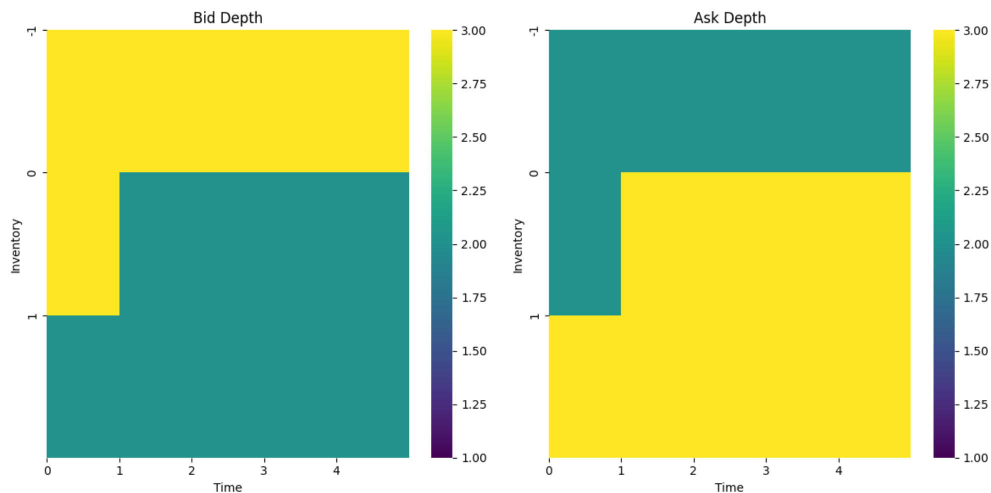

Figure 3.12: Policy DDQN in the Markov chain model setting after 1 episode.

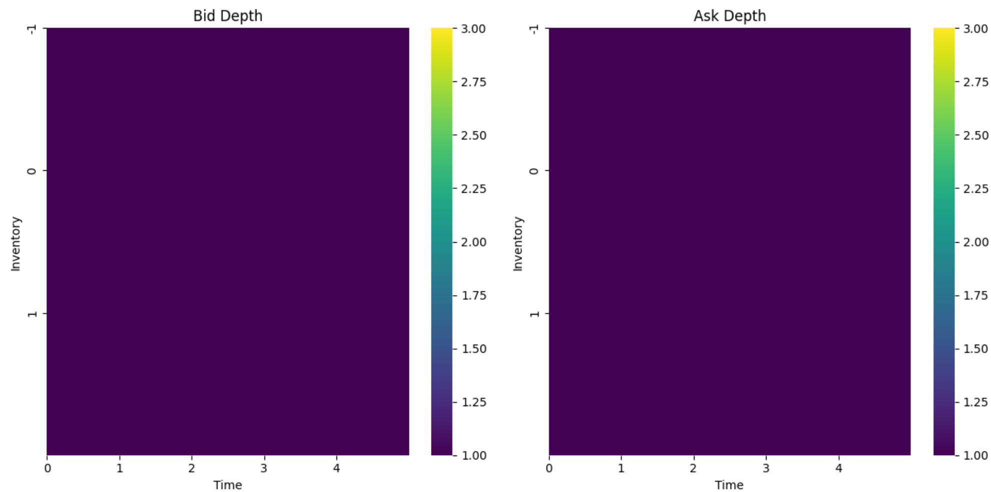

Figure 3.13: Policy DDQN in the Markov chain model setting after 10000 episodes.

We see that the final policy that the agent settles on is vastly different from its initial policy. The agent now prefers to place orders at a depth of 1 on both sides, which is comparable to the results of the value/policy iteration agents.

Double Deep Q-Network Full

To see if we can improve the performance of the agent by adding more state variables, we implement the DDQN Full agent. We present the rewards against episode, a histogram of the mean absolute position, and the frequency of actions taken obtained by the agent in Figure 3.14.

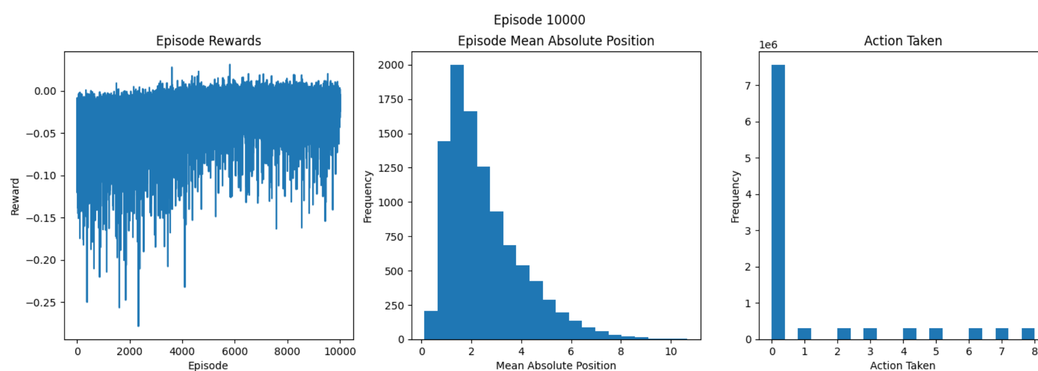

Figure 3.14: Rewards for DDQN Full in the Markov chain model setting.

Again, the rewards, mean absolute position are comparable to DQN and DDQN, with the action frequency being very similar to DDQN. The histogram of mean absolute positions has a slightly heavier tail than DDQN and DQN, but the mean is still around 2. We show the policies of the agent after 1 episode and 10000 episodes in Figures 3.15 and 3.16 respectively.

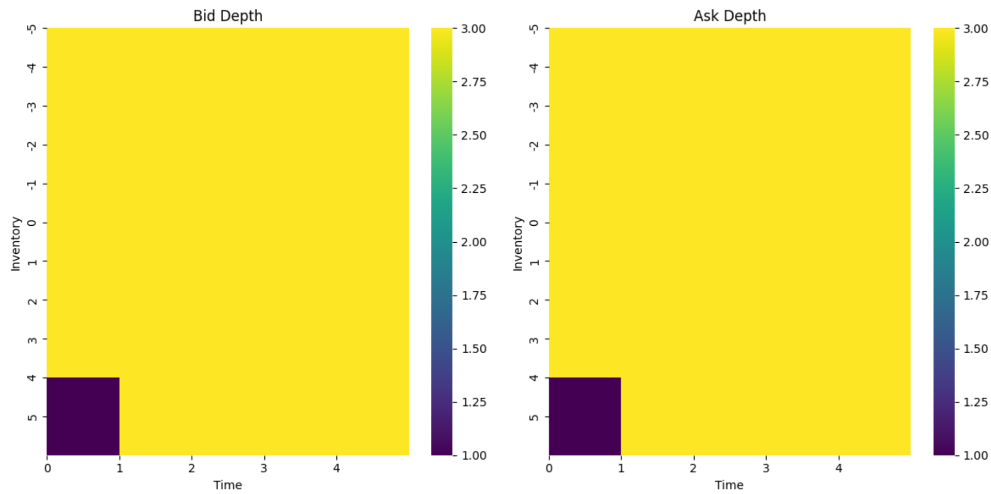

Figure 3.15: Policy DDQN Full in the Markov chain model setting after 1 episode.

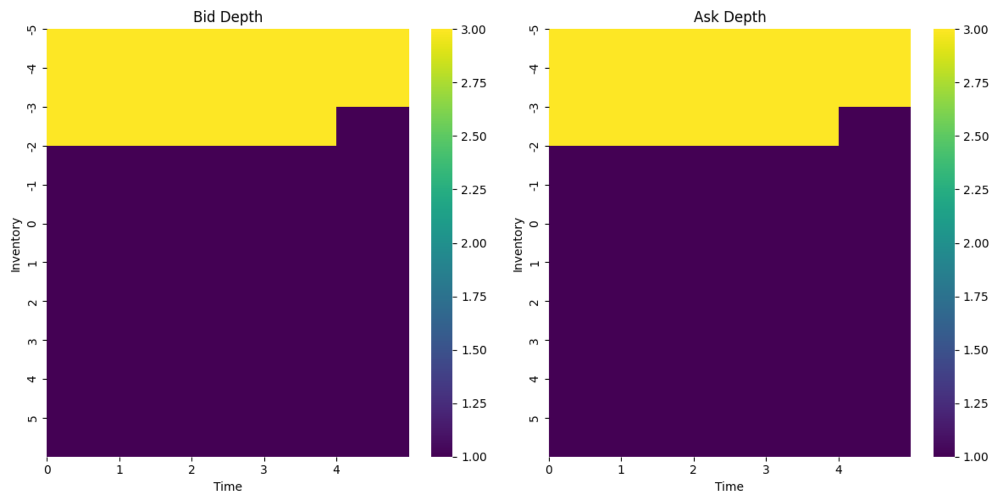

Figure 3.16: Policy DDQN Full in the Markov chain model setting after 10000 episodes.

We see that despite the bigger state space, the agent still settles on a similar policy to DDQN, namely having most of its actions at a depth of 1 on both sides. To note is that the as the inventory becomes more and more negative, the agent starts placing orders at a depth of 3 on both sides. This implies that the agent is placing orders far away from where the trading is happening, a way for the agent to avoid more trading.

Do note that due to the DDQN Full also having the limit order book as a state variable, visualising a policy is difficult, as which order book state do we choose to visualise the policy? There is no clear answer to this, and as such, we just use a symmetric order book around the mid price to visualise the policy.

3.3.2. Results

We now present the results of the agents in the Markov chain model. We present the rewards and the mean absolute position of the agents in the form of violin plots and histograms. We

also present the mean rewards and mean absolute positions of the agents in the form of tables.

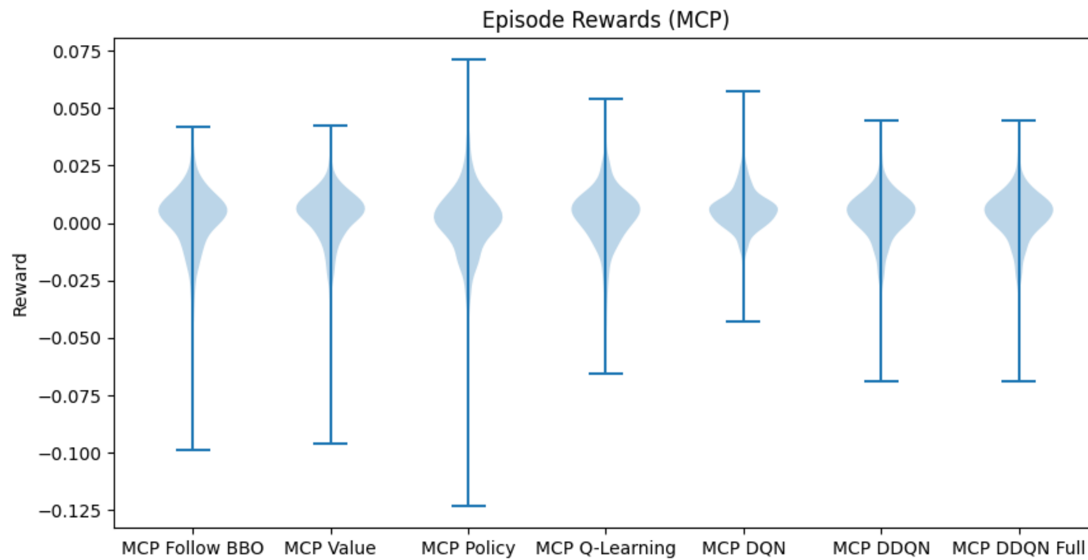

Figure 3.17: Violin plot of rewards of different methods in the Markov chain model setting.

Agent	Mean Reward	Standard Deviation
Follow BBO	0.0011	0.0151
Value Iteration	0.0026	0.0127
Policy Iteration	0.0007	0.0155
Q-Learning	0.0028	0.0134
DQN	0.0055	0.0100
DDQN	0.0024	0.0122
DDQN Full	0.0024	0.0121

Table 3.7: Mean rewards and standard deviation of different agents in the Markov chain model setting, rounded to 4 decimal places.

We see that going off Table 3.7, the DQN agent has the highest mean reward, followed by the Q-learning agent. The DQN agent also has the lowest standard deviation, indicating that the agent is able to consistently perform well. Strangely, the policy iteration agent has the lowest mean reward, despite having the same policy as the value iteration. This probably stems from the fact that the environment is inherently dynamic, a very small change in one order at the start can have a cascading effect on the rest of the orders, leading to very different simulations despite having the same seed. Noticeably, outside of policy iteration, all the other agents have a better mean reward than the benchmark Follow BBO agent, with a lower standard deviation as well. Out of all the non-benchmark policies, and the strange results of policy iteration, Q-learning has the highest standard deviation. Using the violin plot of Figure 3.17, we can see that the visualisation confirms what we see in the table, namely that the deep network agents have a higher mean reward and lower standard deviation than their non-deep network counterparts.

We now look at how good the agents are able to manage their inventory.

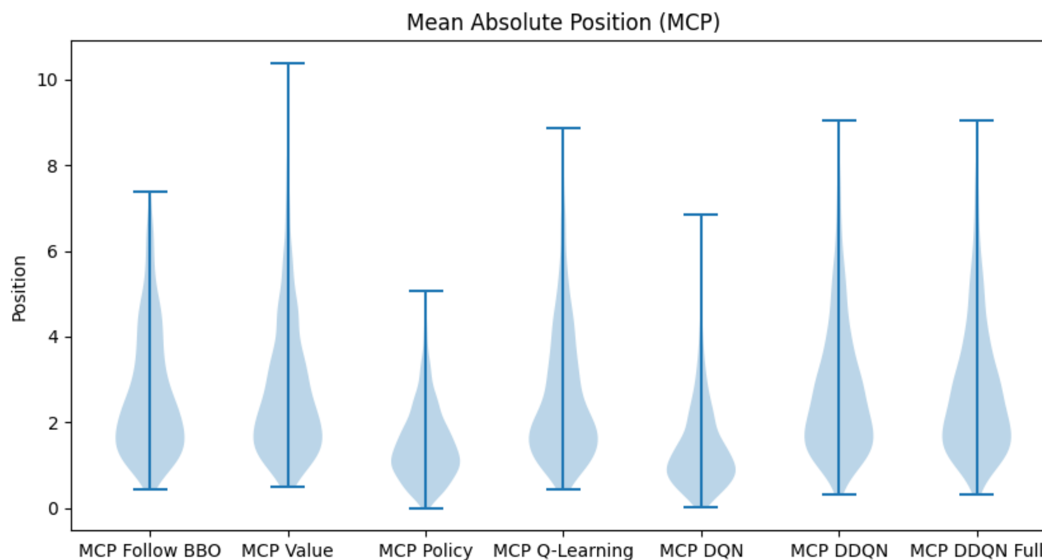

Figure 3.18: Violin plot of Mean Absolute Position of different methods in the Markov chain model setting.

Agent	Mean Position	Standard Deviation
Follow BBO	2.68	1.48
Value Iteration	2.78	1.60
Policy Iteration	1.51	0.87
Q-Learning	2.60	1.50
DQN	1.43	0.96
DDQN	2.77	1.57
DDQN Full	2.77	1.57

Table 3.8: Mean absolute positions and standard deviation of different agents in the Markov chain model setting, rounded to 2 decimal places.

Again, DQN outperforms, having the lowest mean position and second lowest standard deviation, followed by the policy iteration agent where the policy iteration has the lowest standard deviation. As mentioned before, the divergence in results of value iteration and policy iteration is quite an oddity. Ignoring this, we see that most agents outside of DQN have a slightly higher than average mean absolute position than the benchmark Follow BBO policy, with a slightly higher standard deviation as well. Outside of DQN, Q-learning has a slightly lower mean absolute position than the benchmark, but a higher standard deviation. There is no big difference between the deep network agents and the non-deep network agents, as the violin plot of Figure 3.18 shows, outside of DQN.

To finalise the results, we present the histograms of the rewards and mean absolute positions of the agents.

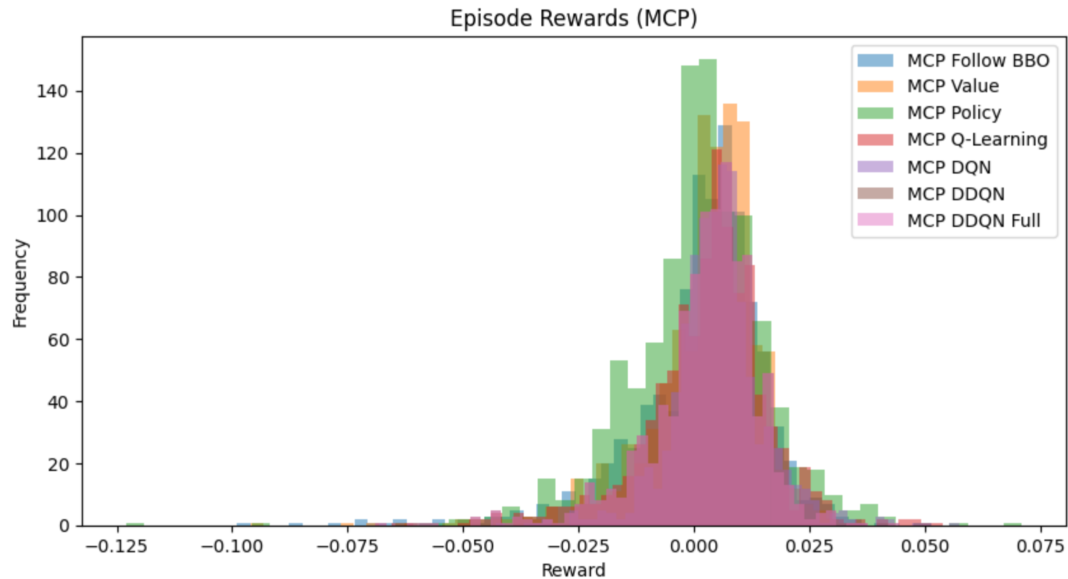

Figure 3.19: Histogram of rewards of different methods in the Markov chain model setting.

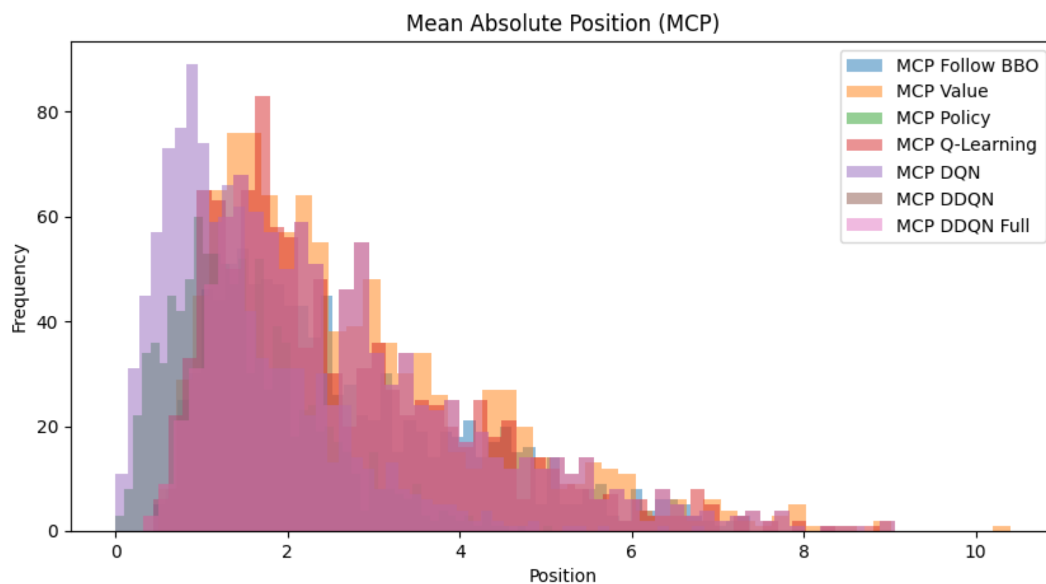

Figure 3.20: Histogram of Mean Absolute Position of different methods in the Markov chain model setting.

The histograms of the rewards and mean absolute positions show that the agents all have a very comparable distribution of rewards and mean absolute position. The differences might seem very marginal, but if we take into account the fact that in reality the agents might be trading in much larger volumes, and a lot more often, these differences might be amplified. Thus, the slightly lower mean of the mean absolute position in DQN, compared to the rest, might be a sign that the agent is able to manage its inventory better. Besides this, the rewards distribution is slightly heavier on the left tail. Most of these are caused by the benchmark

strategy and the policy iteration agent.

With all of this in mind, these results can conclude that DQN seems to be the best performer in the Markov chain model. The agent is able to consistently perform well, and manage its inventory better than the rest of the agents. The policy iteration agent is a very close second, but the strange results of the policy iteration agent make it hard to be assertive about this statement. If we take policy iteration to have the same results at value iteration, we can say that the model free approaches are marginally better than the model based approaches. The deep learning counterparts are also marginally better in the sense of having lower standard deviation in rewards, and with DQN also in the mean absolute position.

We can also conclude that adding more state variables to the agent does not necessarily improve the performance of the agent. The DDQN Full agent has a very similar performance to the DDQN agent, and the DQN agent. This is likely due to the fact that the state space is already quite small, and the action space is also quite small. The agent is able to learn a relatively good policy with the state space it has, and adding more state variables does not necessarily improve the performance of the agent.

The agents are all noticeably better at solving the market making problem than the benchmark Follow BBO agent.

3.4. Queue-Reactive Model

We now move onto the more complicated queue-reactive model that we discussed in Section 2.2.2. Again, we have already discussed the model's parameters in Section 3.2. We will now present the results of the experiments conducted in the queue-reactive model.

I would like to note down again that by using the parameters in the Huang et al. [6] paper, we inherently have a model that will be more volatile than the Markov chain model's parameters from Hult and Kiessling [5]. This is because the queue-reactive model was calibrated on France Telecom, a stock on the french stock exchange, and the Markov chain model was calibrated on the EUR/USD. The EUR/USD is a lot more stable and this is a significant factor to consider when comparing the results of the two models.

The reason why we chose to use the parameters from Huang et al. [6] is that we wanted to see how the agents would perform in a more complicated, and also a more volatile environment. Furthermore, the queue-reactive model is more realistic, as most equities are traded on exchanges with a limit order book, and as such, it is a more relevant application.

3.4.1. Training

Beginning with the training of the agents, we will first present the results of the training process of Q-learning, followed with DDQN, and ending with DDQN Full.

We jump straight into the DDQN from Q-learning as I initially hypothesised that DDQN would perform better, and due to the time it takes to train these agents, namely days, I decided to skip DQN.

Q-Learning

We start with the results of the Q-learning agent in the Queue-Reactive model. We present the rewards and the Q values of the agent after 6000 and 10000 episodes.

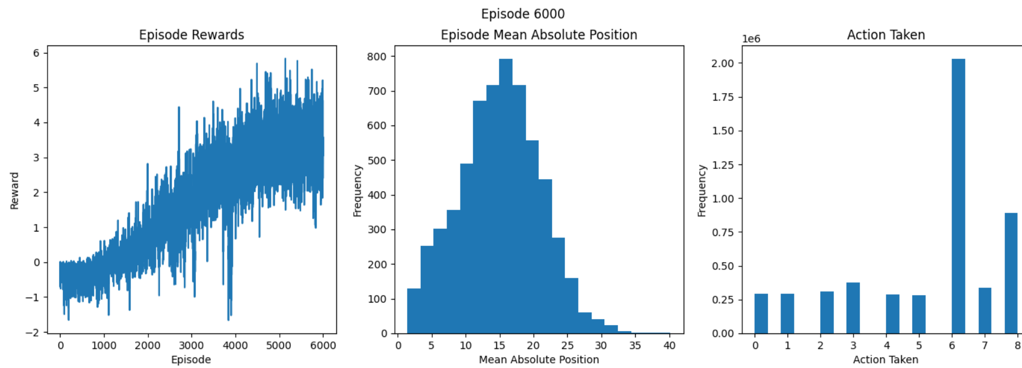

Figure 3.21: Rewards of Q-learning in the queue reactive setting after episode 6000.

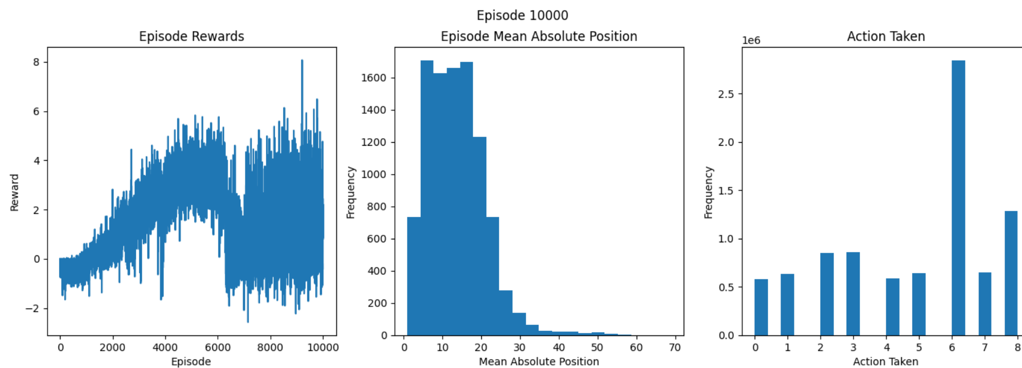

Figure 3.22: Rewards of Q-learning in the queue reactive setting after episode 10000.

As we can see in Figure 3.21 the Q-learning agent seems to quite quickly improve upon its policy, with a significant increase in episode rewards. But, as we see, the deviations in these values also start to increase. Confirming the suspicion that the agent seems to be taking larger risks, the histogram of the mean absolute position show that the agent takes a mean of around 17, which is the highest we have seen so far. Figure 3.22 shows that right after episode 6000, there is a large correction, and the agent now seems to have very large deviations, albeit with a lower mean absolute position.

Below, we present the Q values of the agent after 1 episode and 10000 episodes.

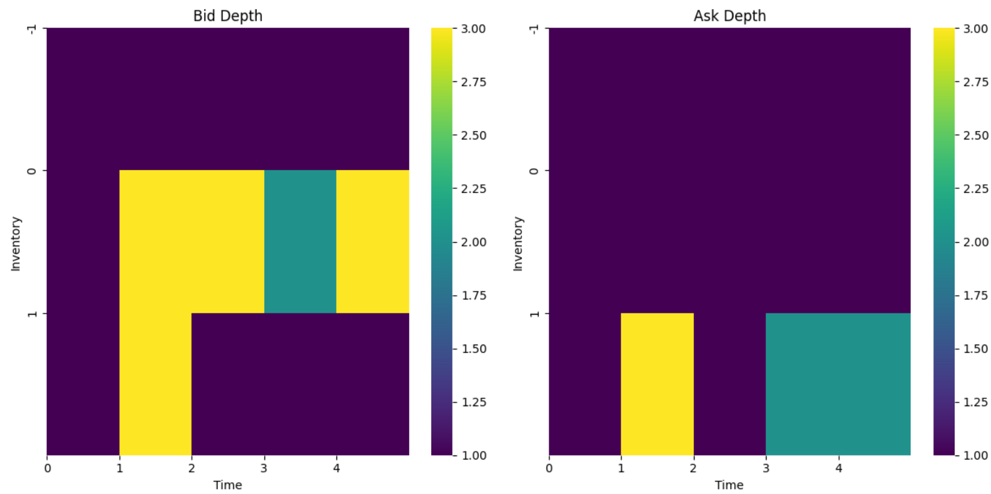

Figure 3.23: Policy Q-learning in the queue reactive setting after 1 episode.

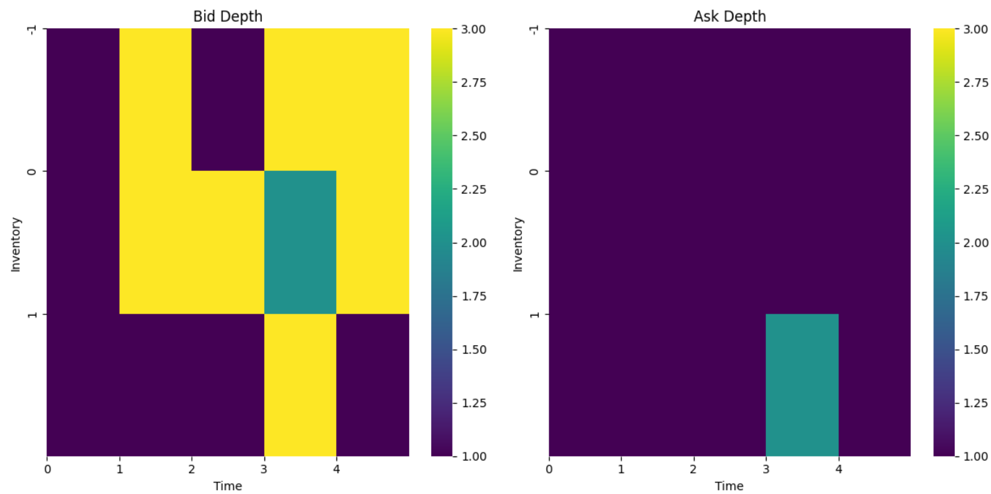

Figure 3.24: Policy Q-learning in the queue reactive setting after 10000 episodes.

The policy after 10000 episodes does not seem to have changed that much, outside of introduce more asymmetry in bid/ask depths. This could be a result of the model being more volatile, and thus the agent having difficult parameterising a policy that is both profitable and stable.

Double Deep Q-Network

We now move onto the results of the Double Deep Q Network agent in the Queue-Reactive model. We start by presenting the rewards.

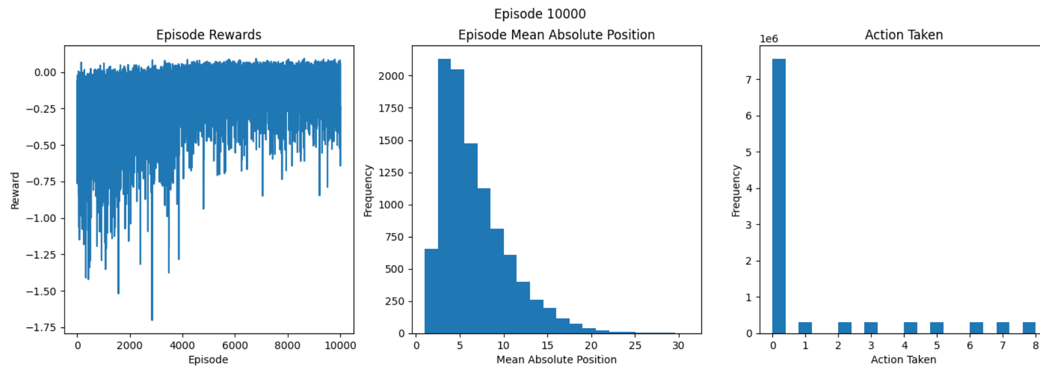

Figure 3.25: Rewards for DDQN in the queue reactive setting.

Figure 3.25 shows that the DDQN agent seems to have learned a more stable policy than the Q-learning agent. The rewards are more consistent, comparable almost to the Markov chain model scenarios, and the mean absolute position is also much lower. This is a good sign, as it shows that the agent is able to learn a policy that is stable.

Below, we present the Q values of the agent after 1 episode and 10000 episodes.

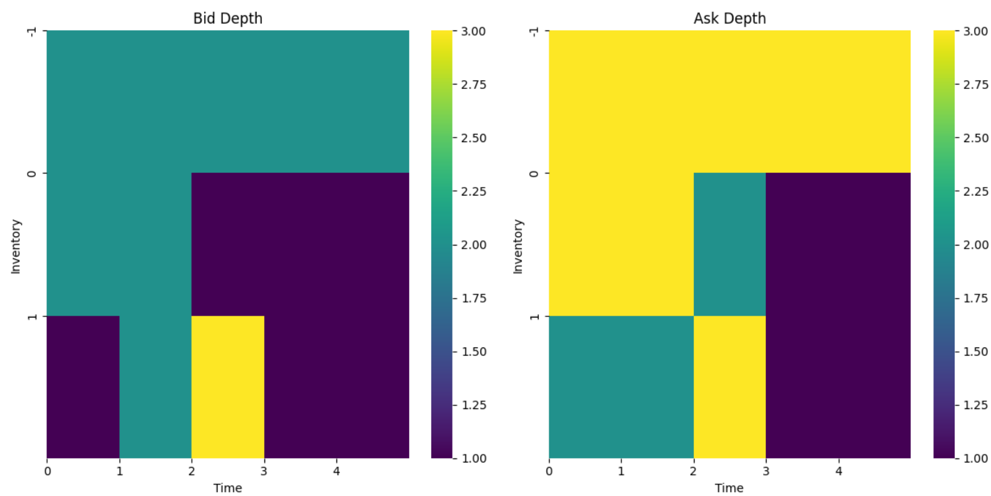

Figure 3.26: Policy DDQN in the queue reactive setting after 1 episode.

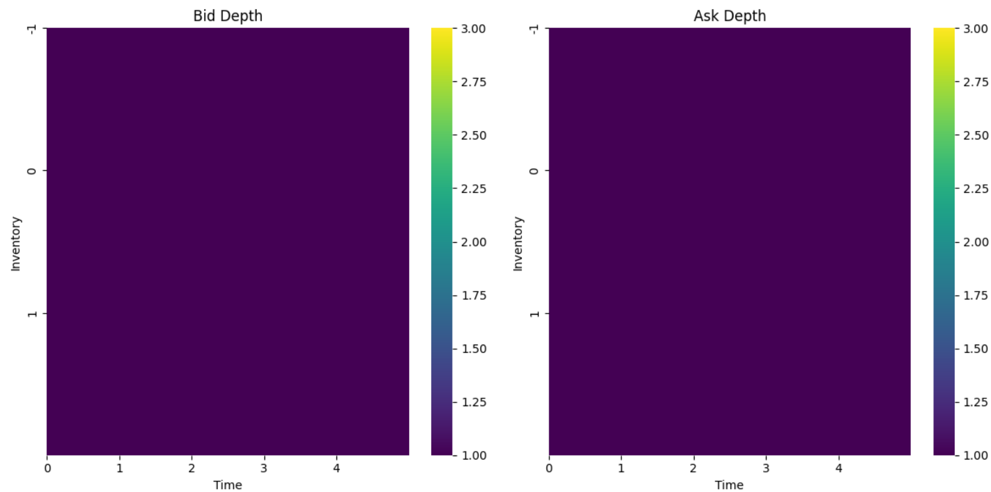

Figure 3.27: Policy DDQN in the queue reactive setting after 10000 episodes.

The policy after 10000 episodes differ a lot from the initial policy it came up with after 1 episode. The agent seems to have learned a policy that is more stable, namely that it only does one action in all states, placing an order only 1 depth away from the best-bid and ask. This is similar to some of the policies that were learnt in the Markov chain setting.

Double Deep Q-Network Full

Finally, we present the results of the Double Deep Q-network.

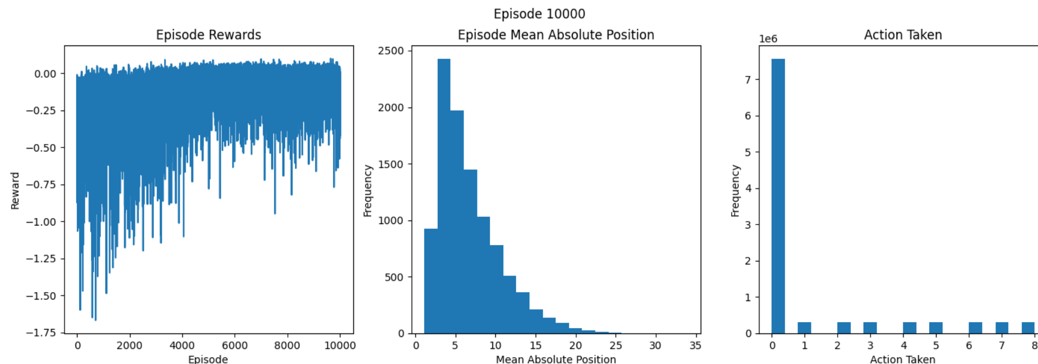

Figure 3.28: Rewards for DDQN Full in the queue reactive setting.

Similar to before, the DDQN Full agent seems to slowly decrease its variance over episodes. It is able to maintain a mean absolute position that is comparable to the DDQN agent, and the rewards are also quite stable. This is a good sign, as it shows that the agent is able to learn a policy that is stable.

Below, we present the Q values of the agent after 1 episode and 10000 episodes.

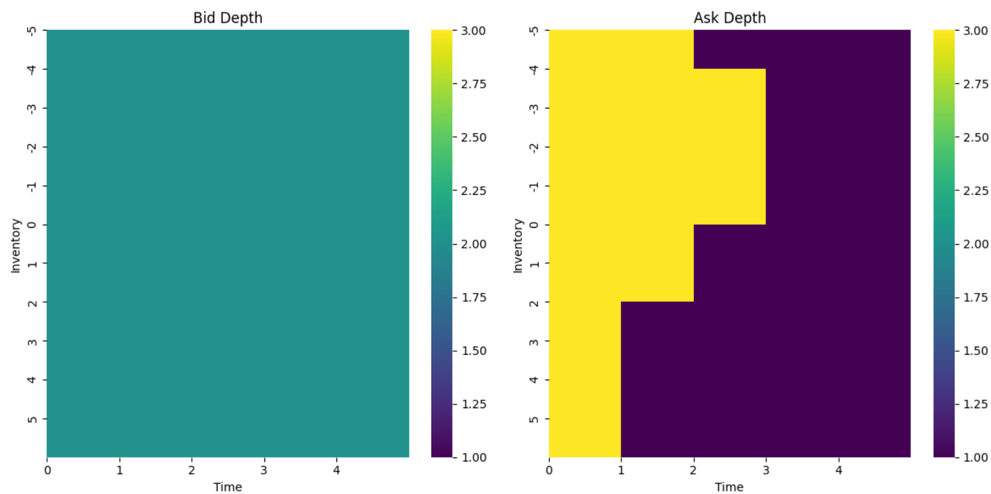

Figure 3.29: Policy DDQN Full in the queue reactive setting after 1 episode.

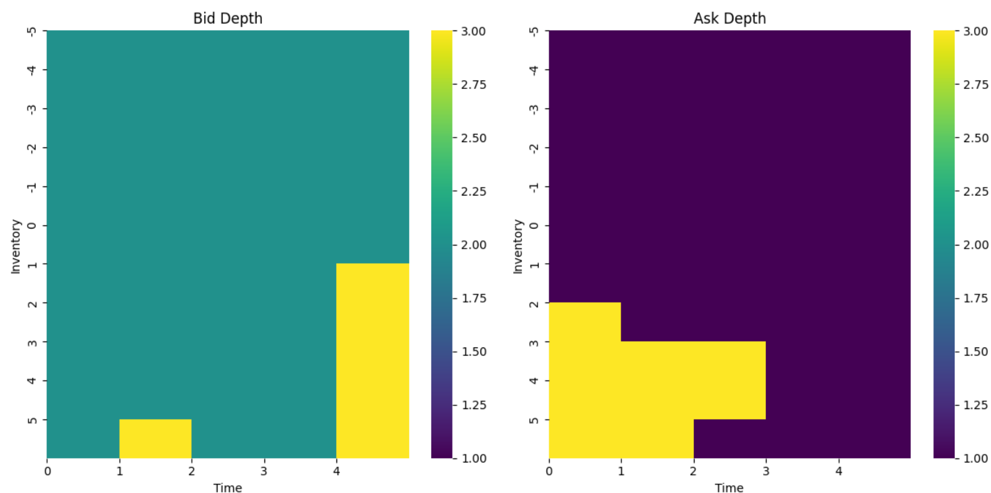

Figure 3.30: Policy DDQN Full in the queue reactive setting after 10000 episodes.

The policy learnt after 10000 episodes is quite similar to the policy learnt after 1 episode. The agent seems to not have been able to learn that much more about the environment than it initially did, outside of taking more 1 depth actions in the ask depth. To note, due to the relatively large state space, some of these states are very rarely visited, and as such, the agent might not have been able to learn a good policy for these states.

Again, visualising DDQN Full is difficult, and making the choice of using a symmetric order book around the mid price might result in taking conclusions from the wrong visualisations.

3.4.2. Results

We now present the results of the agents in the queue-reactive model. We will present the rewards and the mean absolute position of the agents, and compare them to the Follow BBO agent.

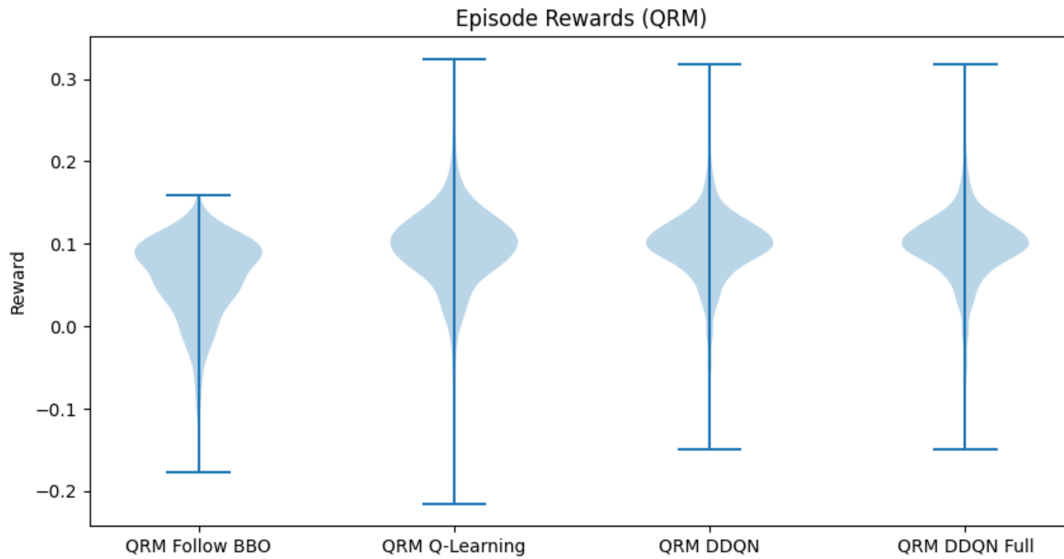

Figure 3.31: Violin plot of rewards of different methods in the queue reactive setting.

Agent	Mean Reward	Standard Deviation
Follow BBO	0.0595	0.0489
Q-Learning	0.0949	0.0522
DDQN	0.0968	0.0424
DDQN Full	0.0968	0.0424

Table 3.9: Mean rewards and standard deviation of different agents in the queue reactive setting, rounded to 4 decimal places.

Table 3.9 shows that all the agents have a higher mean reward than the Q-learning agent. This can also readily be seen in the violin plot in Figure 3.31. The DDQN and DDQN Full agents even have a lower standard deviation than that of the Follow BBO agent and Q-learning.

We now present the mean absolute position of the agents.

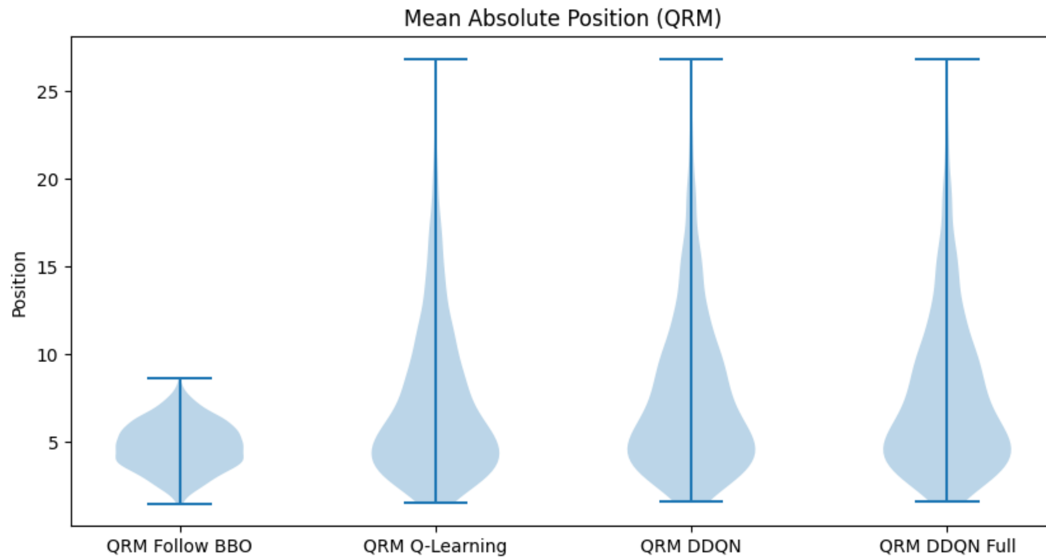

Figure 3.32: Violin plot of Mean Absolute Position of different methods in the queue reactive setting.

Agent	Mean Position	Standard Deviation
Follow BBO	4.78	1.36
Q-Learning	7.23	4.34
DDQN	7.64	4.39
DDQN Full	7.64	4.39

Table 3.10: Mean positions and standard deviation of different agents in the queue reactive setting, rounded to 2 decimal places.

Table 3.10 show us that our agents all have a much higher mean absolute position and standard deviation than Follow BBO. We see this very clearly in the violin plot in Figure 3.32. The DDQN and DDQN Full agents also have a slightly higher mean absolute position than the Q-learning agent, with a slightly higher standard deviation.

To dive deeper, we present the histograms of the rewards and the mean absolute position of the agents.

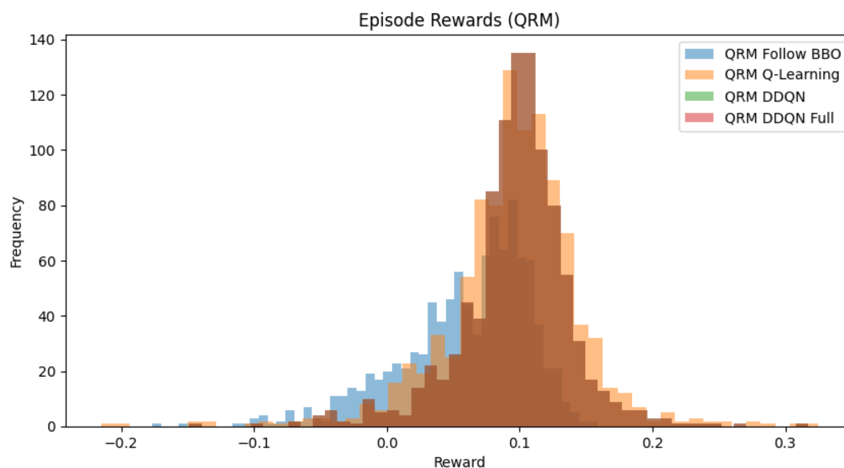

Figure 3.33: Histogram of rewards of different methods in the queue reactive setting.

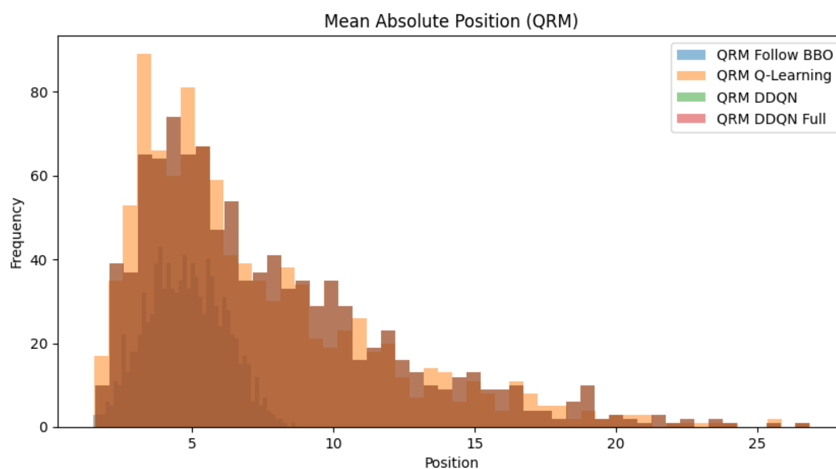

Figure 3.34: Histogram of inventory of different methods in the queue reactive setting.

Figure 3.33 confirms that in general all the agents have a much more profitable distribution compared to Follow BBO. The agent's distribution is more symmetric, and has slightly less heavier tails than the Follow BBO agent.

Interestingly, Figure 3.34 shows that the agents have a much heavier right tailed distribution than the Follow BBO agent. The agents also have a much higher variance. We cannot say if this is due to the model being more volatile, the action space not being diverse enough, or the agents not being able to learn a good policy for the states that are rarely visited.

One thing we can conclude from these experiments are that the underlying environment matters a lot in how well the agents learn a policy. The Queue-Reactive model is a lot more volatile than the Markov chain model, and as such, the agents have a harder time learning a policy that is both profitable and stable. This is evident in the higher mean absolute position and standard deviation of the agents in the queue-reactive model compared to the Markov chain model.

It is odd that DDQN and DDQN Full have exactly the same mean rewards and standard deviations. This is likely due to the fact that we do not use enough samples in evaluating the final policy.

4 Discussion

The training phase of the experiments were interesting to say the least. Initially, I was quite surprised by the results that value and policy iteration came up with, as it mostly took one action. However, as I continued with the other models, we see that many of the results were of this nature, save for a couple of different actions here and there. DQN took a depth of two for both bid and asks, with DDQN and most of DDQN Full taking a depth of one.

If we put ourselves in the agent's shoe, we can start to understand some of the decision making process it has to go through. Firstly, the agent does not have the choice to do nothing, due to the action space. As a result it places two limit orders somewhere in one of these depths of choice. Suppose now that the agent's orders gets hit or lifted, i.e. the market sells or buys from the agent. The agent is now left with a position, and as the next time step unlikely creates a large difference in price (at least in the Markov chain model setting), it immediately gets penalized for holding this inventory, due to the nature of the reward function. As such the agent is incentivized to get rid of this inventory as soon as possible, and to do so it constantly tries to place an order at the top of the book. The agent now has to figure out the sweet spot where it can get get a lot of two way trading, ideally a simultaneous buyer and seller of the agent's orders. This, to capture profit from the spread, while minimizing inventory risk. A natural choice is indeed to always either improve the bid/ask or join the top of the book, i.e. have a depth of one.

We do see that this is not as clear cut as it seems, as in the Markov chain model, DQN ends up performing the best out of the agent's where its policy took a depth of two. Furthermore, in the queue-reactive model, we see that such a policy is not able to always control its inventory well, despite being able to get more rewards than the benchmark policy.

Q-learning in both scenarios ended up having some strange behaviour during the training phase. In the Markov chain model, the agent takes the largest positions out of all the other agents. In the queue-reactive model we saw the same happening initially, where it corrected itself after episode 6000. The agent somehow learns something slightly different than all the other agents, with its policies also being the most varied, and it is not clear why. We theorize that this is perhaps due to not having enough episodes in the training phase, and thus the agent has not been able to explore the state space enough.

While the results are generally positive with respect to the research questions we aimed to answer, there are some limitations to the experiments. Firstly, hardware only allowed us to

train the agents on 10000 episodes, a relatively small training phase. Furthermore, it also seems that not enough episodes were used to evaluate the final policies. Secondly, the simplification of the state space and also the action space, while necessary, might have caused the agents to learn suboptimal policies. Fundamentally, the limit order book dynamics allow for some complex tactics to be used, such as having a resting order at a certain price level, that gets more valuable over time if the price approaches it and more orders join the queue behind it, an indication that there are many buyers/sellers at that price level. The way we have set up the state and action space, the agent is not able to take advantage of this. Lastly, the reward function might not be the best one to use. Inherently, each time step will have a very small reward, as the midprice does not move much. It then gets a relatively large penalty term for holding inventory, and as such most of the rewards are negative. An interesting thing that could have been looked at is how much active trading these agents do, and how often they get hit/lifted.

5 Conclusion

In conclusion, this thesis provides a review and implementation of reinforcement learning to solve for market making in two simulations of limit order books. We motivate the problem from the history of market making at the market's inception to the now mostly electronic markets. Traditionally, attempts to optimize market making have been done analytically, usually involving stochastic differential equations. However, in this thesis, we argue why the use of Markov decision processes are a more suitable way to solve for market making in limit order books and show how reinforcement learning is a natural candidate to solve for the optimal policy.

We recall the main research questions that we initially aimed to answer, namely

1. Can reinforcement learning methods approximate/find optimal solutions to the market making problem?
2. How do policies compare under different market dynamics?
3. How do different RL algorithms compare in terms of performance?

To address these research questions, this thesis establishes an experimental setup that compares trained policies with a well-known analytical benchmark, specifically the at-the-touch market-making agent proposed by Cartea et al. [27]. The experiment simulates limit order books using two models: the Markov chain model from Hult and Kiessling [5] and the queue-reactive model from Huang et al. [6]. It evaluates both traditional methods, specifically value/policy iteration and Q-learning, as well as more recent approaches involving deep reinforcement learning, namely DQN and DDQN.

In summary, the results show that reinforcement learning methods can approximate optimal policies to the market making problem, i.e. that reinforcement learning is a suitable way to solve for market making. The trained agents that we consider are able to outperform the benchmark both in terms of profitability as well as managing inventory risk. Furthermore, the policies are also able to adapt to different market dynamics, albeit with a lot more difficulty in getting suitable convergence, with our results showing that more samples are needed if a stable policy is sought in the more complicated queue-reactive model. Lastly, the results also show that deep reinforcement learning methods can indeed outperform traditional methods in terms of performance. Specifically, we see DQN outperforming all other methods in the Markov chain model. We caveat this by our argument that we believe more samples are needed

to fully evaluate the true performance of the models. Also, it is important to note that deep networks require significantly more hyperparameter tuning, a very time consuming process.

5.1. Future Work

We propose several directions for future work that was uncovered during the course of writing and implementing this thesis.

First, a more comprehensive exploration of different parameters and hyperparameters is needed to better understand the behavior of the models. This includes different neural network architectures within the Deep RL algorithms, but also different RL algorithms as well, such as more policy based approaches like PPO, developed by Schulman et al. [30]. Finally, the use of an alternative reward function that could be more suitable by incorporating a reward signal for 'winning' a trade could be beneficial.

Another promising direction involves extending the state and action spaces to include multiple types of orders and the entire limit order book, which would allow for more complex and realistic trading strategies. One could then also consider the use of more advanced techniques such as multi-agent reinforcement learning, where multiple agents are trained to interact with each other in a competitive or cooperative manner. You could have RL style execution algorithms interacting with multiple RL market making agents, for example. This could provide a more realistic representation of the market, where agents are not only competing against the market, but, also against each other.

Third, the use of a more diverse set of market simulators, and generalising the models to be able to use the same parameter set in different market simulators, could yield deeper insights into the models' behavior and provide a more robust policy.

Finally, calibrating model parameters to real-world data is ultimately essential if one wants to actually use the agents for financial gain. This could be achieved by training models on parameterized simulators and subsequently testing their performance on real market data to assess robustness and generaliseability.

We see that the field is still in its early stages, and there is a lot of potential for future research in this area. We hope that this thesis has provided a good foundation for future work in this area and that it has sparked interest into further exploration into the use of reinforcement learning in solving financial problems.

References

[1] T. Ho and H. R. Stoll, "Optimal dealer pricing under transactions and return uncertainty," *Journal of Financial Economics*, vol. 9, no. 1, pp. 47–73, 1981, ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(81)90020-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304405X81900209.

[2] M. Avellaneda and S. Stoikov, "High-frequency trading in a limit order book," *Quantitative Finance*, vol. 8, no. 3, 2008.

[3] A. Menkveld, "High frequency trading and the new market makers," *Journal of Financial Markets*, vol. 16, no. 4, pp. 712–740, 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. DOI: https://doi.org/10.1038/nature14236.

[5] H. Hult and J. Kiessling, "Algorithmic trading with markov chains," 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:17535093.

[6] W. Huang, C.-A. Lehalle, and M. Rosenbaum, "Simulating and analyzing order book data: The queue-reactive model," *Journal of the American Statistical Association*, vol. 110, no. 509, pp. 107–122, 2015. DOI: 10.1080/01621459.2014.982278. eprint: https://doi.org/10.1080/01621459.2014.982278. [Online]. Available: https://doi.org/10.1080/01621459.2014.982278.

[7] J.-P. Bouchaud, J. Bonart, J. Donier, and M. Gould, *Trades, Quotes and Prices: Financial Markets Under the Microscope.* Cambridge University Press, 2018.

[8] O. Guéant, C.-A. Lehalle, and J. Fernandez-Tapia, "Dealing with the inventory risk a solution to the market making problem," *arXiv*, 2012.

[9] O. Guéant, "Optimal market making," *arXiv*, 2017.

[10] J. D. Farmer, P. Patelli, and I. I. Zovko, "The predictive power of zero intelligence in financial markets," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 6, no. 102, pp. 2254–2259, 2005.

[11] R. Cont, S. Stoikov, and R. Talreja, "A stochastic model for order book dynamics," *Operations Research*, vol. 58, pp. 549–563, 2010.

[12] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 2018, ISBN: 9780262039246.

[13] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[14] H. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23, Curran Associates, Inc., 2010. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989, ISSN: 0932-4194. DOI: https://doi.org/10.1007/BF02551274. [Online]. Available: https://link.springer.com/article/10.1007/BF02551274.

[16] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer series in statistics). Springer, 2009, ISBN: 9780387848846. [Online]. Available: https://books.google.nl/books?id=eBSgoAEACAAJ.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv*, 2014.

[20] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992. DOI: https://doi.org/10.1007/BF00992699.

[21] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016. DOI: 10.1609/aaai.v30i1.10295. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10295.

[22] B. Gašperov, S. Begušić, P. Posedel Šimović, and Z. Kostanjčar, "Reinforcement learning approaches to optimal market making," *Mathematics*, vol. 9, no. 21, 2021, ISSN: 2227-7390. DOI: 10.3390/math9212689. [Online]. Available: https://www.mdpi.com/2227-7390/9/21/2689.

[23] S. Carlsson and A. Regnell, "Reinforcement learning for market making," M.S. thesis, KTH Royal Institute of Technology, 2022. [Online]. Available: https://kth.diva-portal.org/smash/get/diva2:1695877/FULLTEXT01.pdf.

[24] N. T. Chan and C. Shelton, "An electronic market-maker," *MIT*, 2001.

[25] Y.-S. Lim and D. Gorse, "Reinforcement learning for high-frequency market making," in *The European Symposium on Artificial Neural Networks*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53244064.

[26] T. Spooner, J. Fearnley, R. Savani, and A. Koukorinis, "Market making via reinforcement learning," *arXiv*, 2018.

[27] Á. Cartea, S. Jaimungal, and J. Penalva, *Algorithmic and High-Frequency Trading*. Cambridge University Press, 2015, ISBN: 9781107091146.

[28] M. Towers, A. Kwiatkowski, J. K. Terry, *et al.*, *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. [Online]. Available: https://github.com/Farama-Foundation/Gymnasium.

[29] J. Ansel, E. Yang, H. He, *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, ACM, Apr. 2024. DOI: `10.1145/3620665.3640366`. [Online]. Available: `https://pytorch.org/assets/pytorch2-2.pdf`.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: `1707.06347`. [Online]. Available: `http://arxiv.org/abs/1707.06347`.

A

Appendix

Bellman Equation Derivation

The Bellman equation for the state-value function $v_\pi(s)$ is derived as follows:

$$
\begin{aligned}
v_\pi(s) &:= \mathbb{E}_\pi\left[G_t|S_t=s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\gamma G_{t+1}|S_t=s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s\right]+\gamma\mathbb{E}_\pi\left[G_{t+1}|S_t=s\right] \\
&= \sum_{r\in\mathcal{R}} rp(r|s)+\gamma\sum_{g\in\mathcal{G}} gp(g|s) && (*) \\
&= \sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} r\pi(a|s)p(s',r|s,a)+\gamma\sum_{g\in\mathcal{G}}\sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} gp(g|s')\pi(a|s)p(s',r|s,a) && (**) \\
&= \sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} r\pi(a|s)p(s',r|s,a)+\gamma\sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mathbb{E}_\pi\left[G_{t+1}|S_{t+1}=s'\right]\pi(a|s)p(s',r|s,a) \\
&= \sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r|s,a)\left[r+\gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\right] \\
&= \sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r|s,a)\left[r+\gamma v_\pi(s')\right],
\end{aligned}
$$

where (*) follows by using the fact that $p(r|s)$ is a marginal distribution also containing the variables a, s'. We assume that G_{t+1} is a random variable that takes on a finite number of values, and thus we can do the same trick. More difficult, the right term in (**) is derived as follows:

$$
\begin{aligned}
p(g|s) &= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} p(s',r,a,g|s) \\
&= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} p(g|s',r,a,s)p(s',r,a|s) \\
&= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} p(g|s',r,a,s)p(s',r|s,a)\pi(a|s) \\
&= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} p(g|s')p(s',r|s,a)\pi(a|s),
\end{aligned}
$$

where the last line follows from Markov's property.

Hult and Kiessling Lemma 4.1

Lemma 1 *Given a policy alpha* $= (\alpha_0, \alpha_1, \dots)$ *and a state* $s \in \mathcal{S}$*, let* $\theta_s\alpha = (\alpha'_0, \alpha'_1, \dots)$ *be the shifted policy where* $\alpha'_0(s) : \mathcal{S}^n \mapsto \mathcal{A}$ *with* $\alpha'_n(s_0, \dots, s_{n-1}) = \alpha_n(s, s_0, \dots, s_{n-1})$.

The expected value of a policy α *satisfies*

$$V(s,\alpha) = I\{\alpha_0(s) \in \mathcal{C}(s)\}\left(v_C(s,\alpha_0(s)) + \sum_{s' \in \mathcal{S}} P_{ss'}(\alpha_0(s))V(s', \theta_s\alpha_0(s))\right) + I\{\alpha_0(s) \in \mathcal{T}(s)\}v_T(s,\alpha_0(s)).$$

Proof:

Proof can be found in page 11 of Hult and Kiessling [5]. □