Linköping University | Department of Computer and Information Science
Master thesis, 30 ECTS | Datateknik
2018 | LIU-IDA/LITH-EX-A--18/025--SE

Optimized Trade Execution with Reinforcement Learning

Optimal orderexekvering med reinforcement learning

Axel Rantil, Olle Dahlén

Supervisor : Marcus Bendtsen
Examiner : Jose M. Peña

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

In this thesis, we study the problem of buying or selling a given volume of a financial asset within a given time horizon to the best possible price, a problem formally known as optimized trade execution. Our approach is an empirical one. We use historical data to simulate the process of placing artificial orders in a market. This simulation enables us to model the problem as a Markov decision process (MDP). Given this MDP, we train and evaluate a set of reinforcement learning (RL) algorithms all with the objective to minimize the transaction cost on unseen test data. We train and evaluate these for various instruments and problem settings, such as different trading horizons.

Our first model was developed with the goal to validate results achieved by Nevmyvaka, Feng and Kearns [9], and it is thus called NFK. We extended this model into what we call Dual NFK, in an attempt to regularize the model against external price movement. Furthermore, we implemented and evaluated a classical RL algorithm, namely Sarsa(λ) with a modified reward function. Lastly, we evaluated proximal policy optimization (PPO), an actor-critic RL algorithm incorporating neural networks in order to find the optimal policy. Along with these models, we implemented five simple baseline strategies with various characteristics. These baseline strategies have partly been found in the literature and partly been developed by us, and are used to the evaluate the performance of our models.

We achieve results on par with those found by Nevmyvaka, Feng and Kearns [9], but only for a few cases. Furthermore, dual NFK performed very similar to NFK, indicating that one can train one model (for both the buy and sell case) instead of two for the optimized trade execution problem. We also found that Sarsa(λ) with a modified reward function performed better than both these models, but is still outperformed by baseline strategies for many problem settings. Finally, we evaluated PPO for one problem setting and found that it outperformed even the best of the baseline strategies and models, showing promise for deep reinforcement learning methods for the problem of optimized trade execution.

Acknowledgments

We would first like to thank our supervisors at Lynx Asset Management, Per Hallberg and Jonas Johansson. Their sharp comments, questions and feedback has significantly improved the quality of this thesis project, both from a practical and research perspective.

We also want to thank our supervisor at Linköpings University, Marcus Bendtsen, for his valuable feedback and comments. We wish you all the luck with your future research and golf endeavours.

Contents

Financial terms

A financial asset/instrument
is a non-physical asset which value is based on a contractual claim.

A financial exchange/market
is a centralized marketplace where buyers and sellers meet to exchange money for financial assets/instruments.

Lots/quantity/volume
is a discrete number of units of a financial asset/instrument. For instance, 1 lot/quantity/volume of Apple Inc. share (AAPL) refers to one share of the company Apple Inc.

Liquidity
of a financial asset is used to describe how much the price would be impacted when the asset is bought or sold. One of many possible ways of estimating liquidity is by measuring the turnover volume executed per day for the asset. A lower volume generally makes it more difficult to sell or buy larger quantities without impacting the price significantly and vice versa.

Returns
are the total value gained/lost during a time period in comparison to the initial investment.

Volatility
is the dispersion of returns or prices of an asset/instrument.

Abbreviations

MDP	Markov Decision Process
RL	Reinforcement Learning
NFK	Nevmyvaka, Feng and Kearns
PPO	Proximal Policy Optimization
DL	Deep Learning
DRL	Deep Reinforcement Learning
LOB	Limit Order Book
TD	Temporal Difference
MC	Monte Carlo
NN	Neural Network
SGD	Stochastic Gradient Descent
TRPO	Trust Region Policy Optimization
IE	Instant Execution
SL	Submit & Leave
CP	Constant Policy
CPWV	Constant Policy With Volume
ED	Evenly Distributed
PR	Participation Rate
ADV	Average Daily Volume
MPD	Minutes Per Day
EDA	Exploratory Data Analysis

Chapter 1
Introduction

Large institutional investors, such as investment and pension funds, manage savings with the objective to generate positive returns. They provide an important societal function, by providing capital growth to their clients at the same time as they are providing capital to businesses and entrepreneurs. This is often done by engaging in the activity of selling and buying assets at various financial markets. This activity is often referred to as portfolio management.

The act of selling (buying, respectively) a financial asset consists of two sub problems. The first is to determine which asset to sell (buy), how many lots of that asset that should be transacted and within which time horizon the transaction needs to be done. Secondly, selecting an order placement strategy which sells (buys) the volume within the time horizon to the best possible price. For the second problem, different strategies will result in better or worse prices when orders are placed in a market. Empirical research shows that increasing immediacy and size of an order results in an unfavourable price movement [1, 2, 3, 4]. For example, Perold [5] noted that a theoretical portfolio which bought and sold assets at *observed* market prices outperformed the market by almost 20% per year, to be compared with the *implemented* portfolio that outperformed the market by only 2.5% per year. This so called implementation shortfall can thus have a big impact on a portfolio's overall performance. The incurred difference between the observed prices at the beginning of the period and the achieved prices during the period can be seen as a type of transaction cost. The problem of reducing this cost is known as optimized trade execution.

Since the optimized trade execution is a multistage decision process (what orders to place at different time steps) and has a clear objective (minimize transaction cost) it can be seen as a control problem and can be solved analytically using dynamic programming [6, 7]. When doing so, certain aspects need to be assumed, such as the evolution of prices in the market. A potentially more realistic approach is to use historical financial data and define a Markov decision process (MDP) with a trading simulator to model the market dynamics. In this context, an area of machine learning called reinforcement learning (RL) can be applied to solve the problem of optimized trade execution. Research which have used historical data has so far explored various RL algorithms [8, 9, 10]. The use of these techniques has reduced the transaction costs compared to some baseline strategies and shown promising results for RL techniques for this problem [9, 10].

Following a recent surge of deep learning (DL) and RL research, new techniques that incorporate DL into RL have been developed [11, 12, 13]. These techniques have often been evaluated on video games and simulated control tasks. Thus, it is interesting to investigate whether these new techniques can reduce transaction costs further. Addressing the optimized trade execution problem, we will in this thesis implement a set of algorithms, including one incorporating DL, and compare these to a few chosen baseline strategies.

1.1 Aim

The purpose of the thesis is to implement and evaluate a set of RL algorithms that aim to reduce the cost of selling or buying financial instruments compared to a set of chosen baseline strategies.

1.2 Research Questions

The following research questions have been specified to reach the aim.

1. How does our chosen RL algorithms perform compared to a set of baselines strategies with respect to transaction cost?
2. Can the transaction cost be reduced if information regarding historical prices or orders is included in the RL algorithms?

1.3 Delimitations

The overall study is limited to financial data provided by Lynx Asset Management AB. The performance of the models will only be evaluated using simulated markets based on historical data and not with live trading.

Chapter 2
Financial background

In this chapter we will formally define the problem of optimized trade execution. To solve the problem empirically we must simulate a market. We will therefore also introduce a method used to simulate markets, as well as central concepts regarding limit order markets.

2.1 Optimized Trade Execution

As described in the introduction, we are concerned with the optimized trade execution problem. More formally, we define the problem as follows:

Definition 1 *Optimized trade execution is concerned with finding the strategy that sells (respectively, buys) a fixed number of lots of an asset within a given time period (horizon) so that the total achieved price is maximized (respectively, minimized).*

We call a realization of a strategy an episode (e for short) with the maximum length in time units called horizon which we denote as H.

To study the problem of optimized trade execution one must be able to evaluate orders placed in a market. Naturally, doing this in a real market is the most realistic evaluation but can obviously be extremely costly. Additionally, it would take a significantly long time before sufficient amount of data is gathered. For these reasons, researchers and practitioners simulate markets and trades. To do so, three things must be modelled; the price movement without any order placements (evolution of prices over time), the result of an order placement (resulting cash and volume traded) and the price impact of an order (modifications of the price due to order placements). This is generally either done theoretically (as a stochastic process) or empirically (by using historical price data) [6, 7, 8, 10]. In this thesis we will use an empirical simulation of a limit order market. However, we will first provide a simple description of relevant concepts and terms beyond those introduced in the glossary.

2.2 Limit Order Markets

In this section we will present limit orders and limit order markets. All definitions are based on the notation provided by Gould et al. [14]. However, for the purposes of this report, modified and less rigorous definitions are often sufficient. We begin by defining a limit order.

Definition 2 *A limit order $x = (\tau_x, p_x, \omega_x)$ with order type $\tau_x = 1$ (respectively, $= 0$) submitted with price p_x and size $\omega_x > 0$ is a commitment to buy (respectively, sell) up to ω_x lots of the traded asset at a price no greater than (respectively, no less than) p_x.*

We call this a buy (respectively, sell) order for short. Furthermore, the price p_x of an order must be a multiple of the tick size κ of that particular market.

Definition 3 *The tick size κ of a market is the smallest permissible price interval between different orders within it. All orders must arrive with a price that is a multiple of κ.*

When e.g. a buy order is submitted to a limit order book (LOB), the matching algorithm of the market instantaneously checks whether there is any other existing sell order in the LOB to match with. Matching is done automatically if the matching criteria can be met; the submitted buy order is placed at a higher or equal price as the lowest priced sell order in the book. If the matching criteria is met, the lower order volume of either of the matching limit orders is traded for the price of the order already in the LOB. The matching order with the lower volume is removed and the traded volume is subtracted from the order with the most volume. The matching is done repeatedly until either the submitted order has executed its requested volume ω_x or when the matching criteria cannot be met. If an order cannot match all its volume, it is active with the remaining volume. An order is active until matched by a new order, its time limit expires or removed by the investor. The sell process is analogous to the buy process and an example of this process can be found in Section 2.2.2.

Definition 4 *A LOB $\mathcal{L}_t$ is the set of all active orders in a market at time t.*

It is also worth noting that if an order is not matched, i.e. placed directly in the order book, the order is placed last in the queue for that particular price level. A price level consists of a price and the total volume of all active orders with that price. The active orders are always matched according to a first in first out principle. Furthermore, the LOB can be divided into a buy book and a sell book, each consisting of all current buy and sell orders respectively. This can also be referred to as the side, i.e. buy side or sell side of the order book.

In contrast to limit orders, market orders are placed without a specification of the price, i.e. to whatever price that can be achieved. A market order can be placed in a limit order market and is synonymous placing an order in the opposing book at such an extreme price so that the order matches until all volume is executed or no volume is left in the opposing book. When we say order in this thesis we refer to a limit order unless otherwise stated.

2.2.1 Order Book Variables

Based on the properties of the LOB, certain variables can be defined.

Definition 5 *The ask price at time t is the lowest stated price in the sell side of the LOB,*

$$ask_t = \min_{x \in \mathcal{L}_t} \{p_x | \tau_x = 0\}$$

and the bid price at time t is the highest stated price in the buy side of the LOB.

$$bid_t = \max_{x \in \mathcal{L}_t} \{p_x | \tau_x = 1\}$$

Definition 6 *The mid-price at time t is $m_t = (ask_t + bid_t)/2$.*

Definition 7 *The bid-ask spread at time t is $spread_t = ask_t - bid_t$.*

For a complete and thorough formalization of a LOB, see [14].

2.2.2 Example Order in a Limit Order Market

Consider the example where a sell order of 500 lots at price 15.24, $x = (0, 15.24, 500)$, is placed in a limit order market with the buy book before and after the trade presented in Table 2.1. The price of the placed sell order is lower than the current bid_t of 15.30. The sell order is consequently executed instantly at the prices of the buy orders that are already in the books. The first 320 lots are sold at 15.30, the next 80 are traded at 15.29 and so on until, in this instance, all lots are sold. Note that transactions are executed at consecutively lower prices (i.e. *worse* prices) as the sell orders are matched with the buy orders. If the total volume in the buy book at price 15.24 or higher was less than 500, the remaining orders would be placed as the ask in the sell book. The order would remain in the book until removed by the investor or met by a buy order at the same price or higher.

Table 2.1: A buy order book of a specific instrument before and after a sell order of 500 lots at price 15.24 has been placed.

Buy book, 12:03:01.024	
Volume	Price
320	15.30
80	15.29
120	15.24
400	15.22
...	...

Buy book, 12:03:01.025	
Volume	Price
20	15.24
400	15.22
...	...
...	...
...	...

2.3 Trade Simulation

Modern electronic markets, such as NASDAQ, do not only register and publish historical market trades. Many of them also publish the current and historical LOB. The historical data can be used to simulate a limit order market. We will in this section describe a method of simulating a match between an (artificial) order with historical data. This method of simulation was developed with the help of Lynx Asset Management AB. This simulation will be used by the RL algorithms to address the problem of optimized trade execution. We begin with presenting the structure of the data.

2.3.1 Historical Data

The historical LOB data that is commonly available consist of the following information:

- The highest 10 populated price levels and their total order volume respectively
- The lowest 10 populated price levels and their total order volume respectively

We have this data at every minute shift (09:00, 09:01, 09:02, ...) of the hour during the time we choose to be active in the market. More specifically, the data consists of a timestamp, what side (buy or sell side), what price level and the total volume available for that particular price level. We call this data the order book depth or just depth and denote it as $\mathcal{L}_t$.

Along with the historical order book depth, we will also make use of aggregated market trades which consist of all trades that took place during the succeeding one-minute period of the available depth data. More specifically, this data consists of a timestamp, a price level and the total volume that was traded at that particular price level during the minute. We call this data the aggregated trades or just trades and denote the set of aggregated trades that took place between t and $t + \Delta_t$ as $\mathcal{T}_t$. Here Δ_t represents the number of minutes before the next order placement can be done, and depends on how the problem is specified.

2.3.2 Example Trades

To explain the matching process, we present two examples in Figures 2.1 and 2.2. In order example 1, a buy order for 25 lots with price 8 is placed. As per Definition 3, this makes an order $x = (\tau_x, p_x, \omega_x) = (1, 8, 25)$. Since the buy order's price is lower than the $ask_t = 10$, we do not match any volume with $\mathcal{L}_t$. There are thus no trades done *immediately* (i.e. no immediate matching). We call orders that do not match immediately as passive. After the immediate matching, we then assume that our order is placed in the order book and let it trade with the trades that took place after t. These trades are in a sense done *continuously*, we thus call it the continuous matching. In the continuous matching for order example 1, there are $10 + 15 = 25$ lots for a price of 8 or lower in the aggregated trades to be matched with. But the orders in the depth for the price level of 8 must first be filled (since those are in front of our order in the queue), in this case 5 lots. Therefore $25 - 5 = 20$ lots are left to match with. This results in $v_t = 20$ and $cash_t = 8 \times 20 = 160$. Remember, the price of the continuous match will be the price of the artificial order, since we simulate that it is placed in the order book. Here we take into consideration the order book queue.

Order book depth $\mathcal{L}_t$

price	volume	side
11	8	sell
10	10	sell
8	5	buy
7	15	buy

Aggregated Trades $\mathcal{T}_t$

price	volume
11	15
8	**15**
7	**10**

Figure 2.1: Order book example 1.

Order book depth $\mathcal{L}_t$

price	volume	side
11	**8**	sell
10	**10**	sell
8	5	buy
7	15	buy

Aggregated Trades $\mathcal{T}_t$

price	volume
12	15
11	**10**
10	**10**

Figure 2.2: Order book example 2.

In order example 2, we consider a buy order with price 11 for 25 lots, a more aggressive order than in order example 1. Now the order equals $x = (1, 11, 25)$. First, 10 and 8 lots are matched at prices 10 and 11 respectively during the immediate matching. We call an order that matches immediately an aggressive order. There are then $10 + 10 = 20$ lots in aggregated trades to a price of 11 or lower to be matched with. But, in reality we would have already matched with 18 of those when we matched in the immediate matching. They are therefore removed and only $20 - 18 = 2$ lots remain to be matched with in the continuous match. This result in $v_t = 10 + 8 + 2 = 20$ and $cash_t = 10 \times 10 + 11 \times 8 + 11 \times 2 = 210$. In this example, double counting is considered.

2.3.3 Matching Process

The evaluated execution strategies will be limited to only place artificial orders at time points where we have an order book depth $\mathcal{L}_t$. This is due to limitations of the data. Depending on the properties of the artificial order x, the depth $\mathcal{L}_t$ and the aggregated trades $\mathcal{T}_t$, the order may be matched with the order book depth $\mathcal{L}_t$ (called immediate matching), with the aggregated trades $\mathcal{T}_t$ (called continuous matching) or both, as shown in the two example orders.

In short, the immediate matching is done as long as the matching criteria can be met. In the continuous matching, it is assumed that (what is left of) the artificial order is placed in the order book and can be matched with the trades that took place after the immediate matching. Thus, all the volume traded at a lower (higher) price than our order price when buying (selling) is matched with the artificial order. This can be done since we simulate that the order was placed in the books *before the orders that historically was traded*. Note that all the volume executed during the continuous matching is done at the price of the artificial order and not the aggregated trades.

2.3.4 Modifications of the Process

To make the matching process more realistic, we do three modifications. Firstly, as shown in order example 1, if the artificial order is not matched during immediate matching, the trade simulator will take into account the queue for that price, and let the existing volume (if any) for that price level match during the continuous match before the artificial order is allowed to match.

Secondly, shown in order example 2, if the artificial order is matched during immediate matching, the trade simulator will adjust so that the artificial order cannot match with the same historical orders twice. This is done by first removing the volume traded during the immediate matching from the aggregated trades $\mathcal{T}_t$ before the artificial order can match in the continuous part.

Lastly, in the case of a very aggressive order for a large volume which *empties* the order book depth $\mathcal{L}_t$, i.e. match with all the volume for the (maximum 10) price levels in the order book, we assume that the volume for the following price levels will be an average volume of the price levels that were in the book.

The results of the matching process are thus $cash_t$ and v_t. $cash_t$ is the cash spent if buying or received if selling and v_t the total volume executed with the order. To facilitate reproducibility, we present a thorough description of the matching engine used for this thesis in Appendix A.

After a artificial order has been placed and a matching process took place, it may be repeated at the next time step, which may be the one or several minutes later, depending on how the problem is set up (more on this later). At the next chosen time step, the artificial order is removed and a new artificial order must be placed. This may be done several times during an episode until the time limit (horizon) is exceeded. Thus, one can simulate a complete episode of several matching processes.

2.3.5 Assumptions Regarding the Trade Simulation

This method of simulating a market has its shortcomings. Firstly, it only considers orders visible in the order book. For all the markets we simulate in, other participants can place so called hidden orders or partly hidden orders (called iceberg orders). Hidden orders are not showed in the books, but can still be matched with. Iceberg orders only show part of the volume in the LOB, with more volume being added last in the queue for that price upon the visible volume being traded (however, this may vary depending on the exchange). The ratio of hidden to visible orders can be quite significant for some markets. However, since the data used in this thesis miss this type of information, it is impossible to estimate this ratio.

Secondly, it is assumed that during (at shortest) one minute the market will have recovered from our impact. We thus assume that the following minute's order book without any modifications is valid in our simulation. In a real setting, our impact and modifications of the order book persists over time. Again, this is a consequence of the structure of the data.

Lastly, it is assumed that the execution strategies and their simulated orders will not affect the behavior of other market participants. In this method, market reaction to our activity is not simulated, such as adjusting orders following a particularly aggressive artificial order or triggering of new orders following our behavior. One example of such a behaviour could be that a large passive order close to the mid-price might motivate opposite participants to place more aggressive orders to exploit the volume that appeared. For a complete discussion on the assumptions made for the trade simulation, see Section 7.1.1.

Chapter 3
Theory

In this chapter we first present the framework used to model the problem, namely a Markov decision process. Thereafter, we shortly introduce a dynamic programming method. The rest of the theory is concerned with reinforcement learning. We first present classical tabular methods and then introduce concepts related to deep reinforcement learning.

3.1 Markov Decision Process

All definitions, notations and theorems for this section (up until 3.2) are primarily taken from David Silver's lectures in Reinforcement Learning [15].

Consider a finite state space $\mathcal{S}$ and a stochastic process $(S_t)_{t\geqslant 0}$ on $\mathcal{S}$ in discrete time and a finite set of actions $\mathcal{A}$, i.e. a control signal. If the Markov property holds, the probability of the next state S_{t+1} is independent of previous states $S_0, ..., S_{t-1}$ given the current state S_t and an action A_t. We denote a realization of the current state as s, the next state as s' and action taken as a.

Definition 8 *A state S_t is Markov if and only if*

$$\mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a) = \mathbb{P}(S_{t+1} = s'|S_t = s, S_{t-1} = s_{t-1}, ..., S_0 = s_0, A_t = a) = P^a_{ss'}$$

Where $P^a_{ss'}$, the probability of transitioning from state s to another state s' given an action a, is an entry of the transition probability matrix P and fulfills

$$P^a_{ss'} \geqslant 0 \qquad \sum_{s'\in\mathcal{S}} P^a_{ss'} = 1 \tag{3.1}$$

i.e. each row of the matrix represents a probability distribution. Furthermore, a reward is defined. A reward is a type of utility value received as a result of a transition from one state to another, normally a scalar. Given the stochastic nature of state transitions and rewards, a reward function is defined as the *expected* reward given some state and action.

Definition 9 *Reward function is the expected reward given some state s and action a*

$$\mathcal{R}^a_s = \mathbb{E}[R_t|S_t = s, A_t = a]$$

Finally, a discount factor $\gamma \in (0, 1]$ and a return G_t is introduced. A low γ favours short-sightedness whereas a high γ favours a far-sightedness when the return is calculated.

Definition 10 *The return is the sum of future discounted rewards*

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

All variables necessary to define a Markov Decision Process (MDP) are now introduced. In short, to model a MDP one must define a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a probability of state transitions given an action $P^a_{ss'}$, a reward function $\mathcal{R}^a_s$ and a discount factor γ.

Definition 11 *Markov Decision Process is a tuple <$\mathcal{A}$,$\mathcal{S}$,P,$\mathcal{R}$,γ >*

- *A finite set of actions $\mathcal{A}$*
- *A finite set of states $\mathcal{S}$ all with the Markov property*
- *A state transition probability $P^a_{ss'}$*
- *A reward function $\mathcal{R}^a_s$*
- *A discount faction $\gamma \in (0, 1]$*

3.1.1 Policy and Value Functions

In order to solve a MDP, additional definitions are needed. The policy π is defined as essentially a mapping from states to actions.

Definition 12 *A policy π is a distribution over actions given a state*

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

Given a policy, the value function v can be defined, i.e. a function describing the value of being in a particular state following a certain policy.

Definition 13 *Value function is the expected return from following policy π starting from state s.*

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

When the transition matrix and reward function are unknown or not modelled, the problem is considered model-free and an action value function q is considered.

Definition 14 *Action value function is the expected return starting from state s, taking action a and then following policy π*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

3.1.2 Optimality

Solving a MDP is the task of finding the optimal value functions.

Definition 15 *The optimal state-value function $v_*(s)$ is the point-wise maximum value function over all policies*

$$v_*(s) = \max_\pi v_\pi(s)$$

The optimal action-value function $v_*(s)$ is the point-wise maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

We can compare policies though the partial ordering the value function induces,

Definition 16 *A partial ordering of policies is defined as*

$$\pi \geqslant \pi' \quad \textit{if} \quad v_\pi(s) \geqslant v_{\pi'}(s), \forall s$$

This defines a notion of optimality for policies. Given all this, we have the following important theorem.

Theorem 1 *For any Markov Decision Process*

- *There exists an optimal policy* π_* *that is better than or equal to all other policies,* $\pi_* \geqslant \pi, \forall \pi$
- *All optimal policies achieve the optimal value function,* $v_{\pi_*}(s) = v_*(s)$
- *All optimal policies achieve the optimal action-value function* $q_{\pi_*}(s,a) = q_*(s,a)$

A proof for this theorem can be found in [16].

Bellman Expectation Equations

We can decompose the value function and action value function into immediate reward plus discounted value of successor state. These decomposed equations are called the Bellman's expectation equations and are important for iterative solutions.

Definition 17 *The state value function can be decomposed using Bellman's expectation equation for the value function*

$$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1})|S_t = s]$$

The action value function can be decomposed using Bellman's expectation equation for the action-value function

$$q_\pi(s,a) = \mathbb{E}_\pi[R_t + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

3.2 Dynamic Programming

In a MDP where the transition matrix and reward function are known or modelled, dynamic programming for optimal control can be used. This can be seen as an optimal path problem. To find the optimal path we use the Bellman's Principle of Optimality.

Definition 18 *Bellman's Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. [17]*

There are several variants of dynamic programming. We choose to present the backward induction method with an example. Consider a MDP that can be represented as an acyclic graph like Figure 3.1. In this particular example, we wish to sell I units of some goods during T time steps. We have the variables i as units left to sell and t as time steps elapsed. The action is $a \in \mathcal{A}_t$ is the number of units to sell at time step t and reward R_s^a is the cash we get from selling the unit(s). We wish to maximize the total reward during the process.

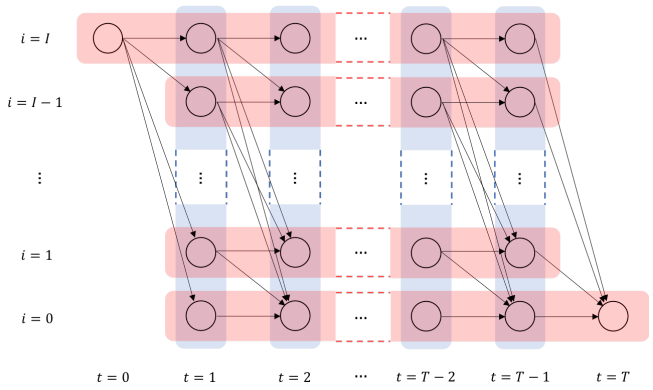

Figure 3.1: Optimal path problem visualized as a directed acyclic graph.

Since the process is a MDP, an optimal path can be found by using the Bellman expectation equation. This equation lets us break down the problem in to smaller subproblems. In Algorithm 1 we iterate backwards in time, solving first the state for $t = T$ and iterate the same procedure back to $t = 0$. Optimality of the solution is given by the principle of optimality.

$v_*(s) \leftarrow 0$ for $s = (T, I)$ since no future rewards will be achieved after that state.
for $t = T - 1$ *to* 0 **do**
  for $i = 0$ *to* I **do**
    $s \leftarrow (t, i)$
    $v_*(s) \leftarrow max_{a \in \mathcal{A}_t}[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_*(s')]$
    Record optimal action for s
  end
end
The optimal policy is given by taking the recorded optimal action for each state you encounter in the path.

Algorithm 1: Backward induction using dynamic programming.

3.3 Reinforcement Learning

To solve a control problem in the model-free setting (without knowing or modelling the transition probabilities and the reward function), the area of machine learning called RL can be applied. RL attempts to learn a policy through interaction with an environment which maximizes the long run reward [18]. The method is appealing since no specification about *how* the task should be performed needs to be designed. More specifically, the RL problem consists of episodes containing states, actions and rewards. The environment initiates the episode by returning a start state. The agent then selects an action A_t based on the start state S_t from the set of possible actions $\mathcal{A}$. The environment returns a reward $r_t \in \mathbb{R}$ and the next state S_{t+1}, see Figure 3.2. This is done repeatedly until the episode terminates (some end state is reached) or some stopping criteria is met. In this thesis we only consider the episodic case that terminate.

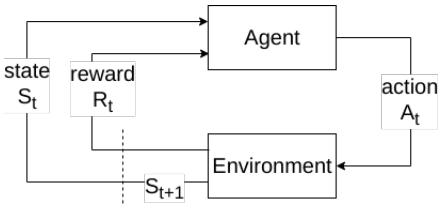

Figure 3.2: The reinforcement learning problem. The agent takes an action a_t and receives a reward r_t and the environment transitions from state s_t to s_{t+1}.

RL differs from supervised learning in that the former attempts to learn from data in a dynamic environment, where the data may come from different distributions over time. Another difference is that there is no supervisor instructing which action is preferable and that rewards can be delayed [19]. The agent in RL must therefore learn through trial and error which imposes the problem of exploration versus exploitation. If the agent maximizes its reward by exploiting actions known to give high reward, the agent might never explore unseen rewards in the environment.

In trading and finance, RL has been used for optimized trade execution, generating trade signals and managing portfolios [9, 20, 21]. These problems are control problems with a clear goal of increasing wealth (or decreasing cost) and with well-defined actions, namely to place buy or sell orders. The two latter applications differ from our application in that we only consider how to execute a order given a trade signal and don't decide what asset to execute. The goal when generating trade signals is to create positive returns while our goal is to reduce transaction cost.

3.3.1 Learning Methods

We will now present some classic methods for learning optimal control using an action value table $Q(s, a)$ to update the value of taking action A in state S. Note that all these methods use tables and are thus limited to discrete states and actions. These methods are similar in that they gather experiences from episodes consisting of a state, action, reward and the next state where the action is chosen according to some policy. This is done continuously until the episode terminates. As they do this, they update the table $Q(s, a)$ and these algorithms mainly differ in how they update the table values. For a complete review of this process, see David Silver's lectures or Sutton and Barto's book on the topic [15, 18].

Q-learning

A well-known algorithm for finding an optimal action value function $q^*(s, a)$ is Q-learning. In Q-learning the $Q(s, a)$ table is updated towards the observed reward R_t plus the estimation of the maximum expected return over actions for the next state as in Equation 3.2. We introduce the learning rate α that controls how large the update is. A large α means that we discard old experiences in favor of new experiences and vice versa.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{A'} Q(S_{t+1}, A') - Q(S_t, A_t)] \tag{3.2}$$

Temporal Difference and Monte Carlo Learning

Two important learning methods are Temporal Difference (TD) and Monte Carlo (MC) learning. TD-learning exploits the Markov property and updates the action-value table using the current estimate of the next action-value of the action chosen in the next state. Contrary to TD, MC-learning samples the complete return G_t (with observed rewards) as a true estimate when updating the action-value table. Using G_t as a target yields an unbiased estimator of

$Q(S_{t+1}, A_{t+1})$ (since it samples rewards from the environment) as seen in Equation 3.3. However, sampling many rewards results in a high variance of the update. TD learning, however, updates towards a sampled reward plus the estimated value of taking an action in the next state $R_t + \gamma Q(S_{t+1}, A_{t+1})$ (using the table value) as in Equation 3.4. This estimation is biased, but has considerably lower variance. Using an estimate in this case is called bootstrapping.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)] \tag{3.3}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.4}$$

Sarsa(λ)

With TD being a 1-step look-ahead, one could also consider a 2-step or 3-step look-ahead which samples 2 or 3 rewards and then bootstrap. This can be generalized to a n-step q-return as seen in Equation 3.5. We use lower case q for scalars such as these.

$$q_t^{(n)} = R_t + \gamma R_{t+1} + ... + \gamma^{n-1} R_{t+n} + \gamma^n q(S_{t+n}, A_{t+n}) \tag{3.5}$$

With MC and TD learning representing two extremes. The target for Sarsa(λ) is striking a compromise between the two by using a weighted sum of all n-step q-returns $q_t^{(n)}$, see Equation 3.6.

$$q_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)} \tag{3.6}$$

Where $\lambda \in [0, 1]$ is the parameter that regulates between sampling rewards and bootstrapping. The update using the n-step q-returns of the action-value table the weighted sum of all n-step q-returns, see Equation 3.7.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[q_t^{\lambda} - Q(S_t, A_t)] \tag{3.7}$$

In a sense, λ can thus regulate the bias and variance in the table update.

3.4 Deep Reinforcement Learning

Using tabular methods limits the state space and action space to be discrete and finite. A table is also limited by the fact that it cannot estimate the value of an unseen states [15]. These two limitation poses a problem when the state and action space is high dimensional or continuous in terms of learning speed and loss of information when discretizing the state or action space [18]. To overcome these issues a function approximator can be used instead of a table. A possibility is to use a neural network as a function approximator. The group of RL methods that incorporates deep neural networks into the reinforcement learning problem, are called deep reinforcement learning (DRL).

3.4.1 Feed Forward Neural Networks

A feed forward neural network (NN) can be seen as an acyclic directed graph where the input x is propagated through a network consisting of layers [22]. A visualization of a one layer feed forward NN is seen in Figure 3.3.

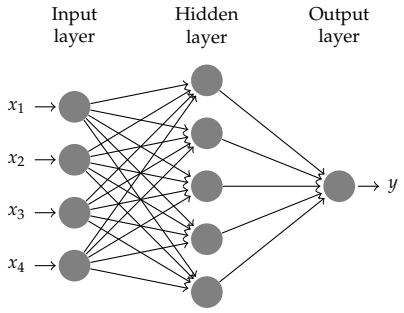

Figure 3.3: A one layer feed forward NN with four inputs, five neurons in the hidden layer and one output.

The input x is propagated through all layers $l \in L$ by multiplying the previous layer's output a^{l-1} with a weight matrix W^l plus a bias b^l and then applying an activation function $f(z^l)$ as in Equation 3.8. Common choices for activation functions for the input and hidden layers are the sigmoid, hyperbolic tangent (tanh) and Rectified Linear Unit (ReLU). A deep NN consists of multiple hidden layers.

$$\begin{aligned} a^0 &= x \\ z^l &= W^l a^{l-1} + b^l \\ a^l &= f(z^l) \\ a^L &= \hat{y} \end{aligned} \tag{3.8}$$

$$sigmoid(z) = 1/(1 + \exp(-z)) \tag{3.9}$$

$$tanh(z) = (\exp(z) - \exp(-z))/(\exp(z) + \exp(-z)) \tag{3.10}$$

$$relu(z) = \max(0, z) \tag{3.11}$$

When an input x has been forward propagated through the network, a prediction $\hat{y}$ is given. With a true value y given the input x, a loss function $L(\hat{y}, y)$ can be defined. To minimize this loss function with gradient descent, the derivatives for each weight matrix W^l and bias b^l can be computed through an algorithm called back propagation. The algorithm, as presented in Algorithm 2, first computes the gradient of the loss function with respect to the predictions and then propagates the gradient g backwards from the first to the last layer. For each layer l, the gradient g is element-wise multiplied with the derivative of the activation function $f'^{(z^l)}$. Then the derivatives of the weight matrix $\nabla_{W^l} L$ and bias $\nabla_{b^l} L$ of that layer is assigned and the gradient is propagated back to the previous layer.

$g \leftarrow \nabla_{\hat{y}} L(\hat{y}, y)$
for l = L, L-1,...,1 **do**
  $g \leftarrow g \odot f'(z^l)$ element-wise multiplication
  $\nabla_{b^l} L = g$
  $\nabla_{W^l} L = g a^{(l-1)T}$
  $g \leftarrow W^{lT} g$
end

Algorithm 2: Back Propagation [22].

After the forward and backward propagation each weight and bias in the network can be updated using stochastic gradient descent (SGD) with learning rate α as in Equation 3.12.

$$W^l = W^l - \alpha \nabla_{W^l} L$$
$$b^l = b^l - \alpha \nabla_{b^l} L \tag{3.12}$$

Adaptive Moment Estimation

An extension of gradient descent is Adaptive Moment Estimation (ADAM), which has been found to be robust and suited for many non-convex optimization problems in machine learning [23]. ADAM uses two exponential moving averages for the first moment m_t and the second moment v_t estimate of the gradient. ADAM does not require a stationary objective and performs a natural learning rate annealing [23]. The ADAM algorithm is presented in Algorithm 3, it calculates the gradient at iteration t and then calculates the first and second moment of the gradient. The parameters are then updated with the first moment divided by the square root of the second moment. The parameters θ can be the weights of a NN and $L(\theta)$ the loss based on some performance measure.

Require
α: learning rate
$\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for moment estimates
$L(\theta)$: Stochastic loss function with parameters θ
θ_0: Initial parameter vector
$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
$t \leftarrow 0$
while *not converged* **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta L(\theta)$
  $m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$
  $v_t \leftarrow \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (β_1 to the power of t)
  $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (β_2 to the power of t)
  $\theta_t \leftarrow \theta_{t-1} - \alpha * \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$
end
return θ_t

Algorithm 3: ADAM: g_t^2 is the element-wise square. Tested hyperparameters to perform well on machine learning task are: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ [23].

Initialization of Weights and Preprocessing of Inputs

The weight matrices of a NN are often initialized randomly from a uniform or normal distribution and the biases to zeros [22]. When the weight initialization comes from a normal distribution a mean of zero is often used [22]. The weights and the inputs x of the NN should be of the same scale to learn efficiently [24]. Scaling the inputs to a minimum of zero and a maximum of one can be used to make the input the same scale as the weights [22, 25].

3.4.2 Policy Gradient Methods

In value-based DRL one commonly parameterizes an action-value function, optimize it to reduce some loss and then implicitly derives a policy by acting greedily (choosing the action with the highest action-value) for each state it encounters. Instead, one can directly optimize the policy, i.e. parameterize a function mapping a state to an action, $\pi_\theta(a|s)$ and then optimize that policy with respect to the parameters in order to maximize the long term reward

[18, 26]. Methods that explicitly finds a policy by following a gradient are called policy gradient methods. One advantage with these methods compared to value-based methods are that they allow for stochastic policies, which may be the optimal policy for some problems [18]. Furthermore, they are also more suitable for problems with continuous action spaces. Consider a differentiable policy that estimates the mean $\mu_\theta(s)$ and standard deviation $\sigma_\theta(s)$ of a normal distribution which it can sample actions from.

$$\pi_\theta(a|s) = \frac{1}{\sigma_\theta(s)\sqrt{2\pi}} \exp\left(-\frac{(a-\mu_\theta(s))^2}{2\sigma_\theta(s)^2}\right) \tag{3.13}$$

In this thesis we will consider a deep NN for function approximation with θ as the weights of the network. In an episodic environment we choose to define the value of being in the initial state given a policy v_{π_θ}

$$J(\theta) = v_{\pi_\theta}(s_0) = \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) R_s^a \tag{3.14}$$

as a performance measure that we wish to maximize where $d(s)$ is the on-policy distribution under π_θ. If we wish to maximize the quantity $J(\theta)$ we can make use of the policy gradient theorem.

Theorem 2 *The policy gradient theorem establishes that*

$$\nabla J(\theta) \propto \sum_{s\in S} d(s) \sum_{a\in\mathcal{A}} q_\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

$\nabla J(\theta)$ need only be proportional to the right hand side, which we call sample gradients, since any constant could be absorbed by the learning rate in the gradient ascent update. If we manipulate the right hand side we can rewrite the gradient as

$$\nabla J(\theta) = \mathbb{E}_\pi[G_t \nabla_\theta \log \pi_\theta(A_t|S_t)] \tag{3.15}$$

We now have an expression that enables us to sample returns from the environment and get an unbiased estimator of the gradient of $J(\theta)$. This yields the parameter update in the gradient ascent.

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla_\theta \log \pi_\theta(A_t|S_t) \tag{3.16}$$

where α is the learning rate. A proof for the policy gradient theorem and the manipulation of the right hand side can be found in [18].

The simplest group of algorithms that uses this update are the REINFORCE algorithms, introduced by Williams [27]. Since G_t is the complete return from t to the end of the episode, this method can be seen as a MC algorithm. As shown, this is unbiased, but MC updates has high variance, which the training process is sensitive to. This is due to a training process which is non-stationary since G_t is generated from different policies $\pi_{\theta_t}(a|s)$ over time. Much of the research regarding policy gradient methods tries to lower this variance, sometimes with the consequence of introducing some bias [11, 13, 28].

Variance Reduction and Advantage Function

The high variance of the REINFORCE updates can be addressed with the use of a baseline function. Any baseline function $b(s)$ not depending of action a, can be subtracted from $q_\pi(s,a)$ in Theorem 2 as presented in Equation 3.17. The update rule is then as in Equation 3.18. [29]

$$\nabla J(\theta) \propto \sum_{s\in S} d(s) \sum_{a\in\mathcal{A}} (q_\pi(s,a) - b(s)) \nabla_\theta \pi_\theta(a|s) \tag{3.17}$$

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t))\nabla_\theta \log \pi_\theta(A_t|S_t) \tag{3.18}$$

A common baseline is a value function approximation $v_\theta(s)$ with parameters θ which enables the update to separate which actions that are better or worse than the estimated value of being in a state s. The value function approximation can also be used to reduce variance in G_t by replacing observed rewards with the value function approximation. Methods that uses an update where observed rewards r_t have been replaced with value function estimates are called actor-critic methods [29]. The actor is the policy $\pi(a|s)$ and the critic is the value function estimate $v_\theta(s)$. Consider a one step return $G_{t:t+1}$, from t to $t+1$, update in Equation 3.19. This return can be replaced with a reward and the value function approximation of the next step. This update induces bias from the value function approximation $v_\theta(S_{t+1})$.

$$\theta_{t+1} = \theta_t + \alpha(G_{t:t+1} - v_\theta(S_t))\nabla_\theta \log \pi(A_t|S_t, \theta) \tag{3.19}$$

$$= \theta_t + \alpha(R_t + \gamma v_\theta(S_{t+1}) - v_\theta(S_t))\nabla_\theta \log \pi(A_t|S_t, \theta) \tag{3.20}$$

A T-step advantage function $\hat{A}_t$ can now be defined as

$$\hat{A}_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} v_\theta(s_T) - v_\theta(s_t) \tag{3.21}$$

The advantage is higher if the observed returns are higher than the estimated value of being in that state. The T-step advantage function be extended to a Generalized Advantage Estimation (GAE) in a similar fashion to Sarsa(λ) learning where λ is used as a parameter to adjust the degree of bootstrapping to perform [30].

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{3.22}$$

$$\delta_t = r_t + \gamma v_\theta(s_{t+1}) - v_\theta(s_t) \tag{3.23}$$

Two special cases are when $\lambda = 0$ which corresponds to $\hat{A}_t = \delta_t$ and $\lambda = 1$ that corresponds to $\hat{A}_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - v_\theta(s_t)$. The first case is TD learning and the latter case is MC learning.

3.4.3 Current Research

There have been many recent developments in research regarding policy optimization [11, 12, 28, 31]. An important development for algorithms in environments with high uncertainty in the reward is the stability of the training. No precise definition of training stability is found in the literature. But the variance of the reward from multiple trainings is often observed [11]. The stability of the training is important especially for on-policy algorithms, which we will use in this thesis. This is because on-policy algorithms use the current policy for collecting experiences. The experiences are collected by running the policy in the environment and storing the states, actions and rewards observed. If the policy cannot improve using these experiences the learning is limited. Therefore, stability of the policy updates has been researched [12].

Loss Function

One topic that current research has focused on the loss function to improve the stability of the policy updates. A big focus of this research is to manipulate the loss function in an attempt to achieve more stable updates [12, 13]. One approach is to use the generalized advantage estimation $\hat{A}_t$ from Equation 3.22 instead of G_t. When the gradient in Equation 3.15 is estimated using a finite batch of samples of $\hat{A}_t$ we get

$$\hat{g} = \hat{\mathbb{E}}_\pi[\hat{A}_t \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{3.24}$$

The loss function is then

$$L(\theta) = \hat{\mathbb{E}}_\pi[\hat{A}_t \log \pi_\theta(a_t|s_t)] \tag{3.25}$$

In the paper called Trust Region Policy Optimization (TRPO) a method, also called TRPO, manipulated the loss function in Equation 3.25 to ensure monotonic improvement of the expected return [12]. The term $\log \pi(a|s,\theta)$ was replaced by a surrogate loss incorporating a probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{3.26}$$

which results in the loss function

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_\pi[\hat{A}_t r_t(\theta)] \tag{3.27}$$

that is maximized for each iteration with respect to a constraint, see [12]. TRPO managed to learn complex control tasks such as swimming, walking and hopping in a physics simulator [12].

3.4.4 Proximal Policy Optimization

Proximal policy optimization (PPO) builds on the ideas of TRPO but does not perform a constrained maximization. Instead, it uses a clipped loss function without a constraint.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_\pi[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\xi, 1+\xi)\hat{A}_t)] \tag{3.28}$$

Here ξ is a constant, typically in the range 0.1 to 0.3 [13].

The first term in the min operator is the same as in TRPO. But the second term clips the probability ratio which removes the possibility of making too large policy updates. The min operator makes the loss function to a pessimistic bound of the unclipped loss, since we want to maximize the loss in policy optimization. The clipping of $r(\theta)$ incentives the optimizer to optimize the policy such that $r(\theta)$ stays within $[1-\xi, 1+\xi]$. This scheme makes the clipping only active when the loss is getting worse [13]. The loss function is maximized if the actions generating large advantages becomes more probable when altering the parameter vector θ. When a value function approximator $v_\theta(s_t)$ is used, there is an additional loss with respect to the value function.

$$L^{VF} = \hat{\mathbb{E}}_\pi[(v_\theta(s_t) - G_t)^2] \tag{3.29}$$

An entropy bonus is also added to the loss function to enable sufficient exploration. For example, more exploration is performed if the policy's standard deviation is increased. See Equation 3.30 for the entropy H of a normal distribution [32]. Here π and e represents the numbers $\pi \approx 3.141$ and $e \approx 2.718$.

$$H_{\pi_\theta}(s_t) = \frac{1}{2}\log(2\pi e \sigma_\theta(s_t)^2) \tag{3.30}$$

The resulting loss for PPO is

$$L^{PPO}(\theta) = \hat{\mathbb{E}}_\pi[L^{CLIP} - k_1 L^{VF}(\theta) + k_2 H_{\pi_\theta}(s_t)] \tag{3.31}$$

Where k_1 and k_2 are constants regulating the loss function's prioritization of the terms.

Training Algorithm

Now when the generalized advantage function $\hat{A}$ and loss function L_t^{PP0} has been defined the training algorithm can be introduced. The training algorithm consists of iterations i where N number of actors runs through the environment for T steps and samples actions from the policy $\pi_\theta(s|a)$, storing the experiences (s, a, r, s') in a temporary memory $\mathcal{D}$. The advantage for each experience is calculated and then stochastic gradient ascent is performed with mini-batches of size M on these advantages for K times. The training algorithm is presented in Algorithm 4. In practice, an extension is that the data collection is performed until U experiences are collected, ensuring that a certain number of experiences are gathered.

for *iteration i=0,1,...,IT* **do**
  $u = 0$
  while $u <= U$ **do**
    for *actor=1,2,...,N* **do**
      Run policy $\pi_{\theta_{old}}$ for T time steps
      Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end
    $u+ = N \times T$
  end
  Optimize loss L_t^{PPO} with respect to θ and minibatch size $M <= U$
  $\theta_{old} \leftarrow \theta$
end

Algorithm 4: Training algorithm PPO.

Many new variants of policy optimization have been developed in the recent years [11, 12, 28]. But the authors of PPO describe it as easier to implement and tune compared to these new variants [13].

3.4.5 Reward Manipulation

To limit the policy updates further, clipping of the rewards have been used [33]. This also simplifies the use of the same learning rate for multiple games (Atari games in [33]) with different magnitudes of the rewards.

3.4.6 Hyperparameter Tuning

Most recent DRL research evaluates algorithms in simulated games or control tasks. When algorithms have been evaluated on games, the authors have tuned the hyperparameters on one or a few games through informed search or grid search and then used the same hyperparameters for all games [11, 13, 33, 34].

Chapter 4
Models

In this chapter we will present how we choose to model the RL environment, the baseline strategies and the algorithms used to learn and act in this environment. We have divided the thesis into two parts which both rely on the matching simulator described in Section 2.3. In Table 4.1 we present what the two parts constitute.

Table 4.1: Arrangement of the thesis project.

Part	Action space	Baseline strategies	Models	Dual
Part I	Agents may only choose price of the order (all volume)	Instant Execution (IE) Submit & Leave (SL) Constant Policy (CP)	NFK Dual NFK Sarsa(λ)	No Yes Yes
Part II	Agents may choose both price and volume of the order	Constant Policy With Volume (CPWV) Evenly Distributed (ED)	PPO	No

We also distinguish between the dual and non-dual case. In the non-dual case we train two models for each problem setting, one for buying and one for selling. In contrast we have the dual case where we train one model for each problem setting that can both buy and sell.

4.1 Part I

In this section we will present the most basic environment used for training agents. This basic environment will be the foundation for all agents, it will however be modified depending on the model. Each environment contains order book and trades data of the particular period of interest, e.g. the training period. The environment has two modes, either train or testing. In training mode, the environment will randomly pick a time point in the order book to start an episode. This can be done arbitrarily many times in order to train a model. In testing mode, however, the environment will start the episode at the first time point in the order book. Each time the environment is reset, it will begin the next episode at the next time point. This maximizes the use of data when evaluating and is also realistic in the sense that an investor might start an execution any time during a time period.
When the agent acts (either when training or evaluating) it will automatically take a time step in the episode, return the next state and a reward. This will be done until either the volume is executed or when the time horizon limit H is reached. At this point, the environment will execute the remaining inventory no matter what action the agent chooses to take, making it costly to end an episode with a lot of volume.

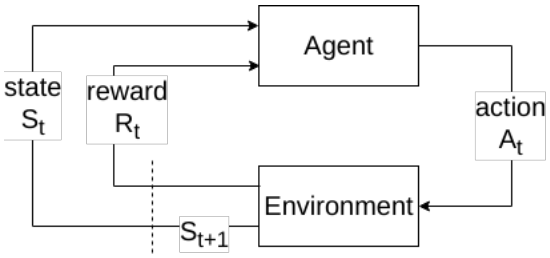

Figure 4.1: The interaction between the agent and the environment.

4.1.1 Action Space

We begin with defining a one dimensional discrete action space $a = p_\Delta \in \mathbb{Z}$. The value of the action is the number of tick size κ from the bid_t when buying and ask_t when selling. The price is thus

$$p_x = bid_t + p_\Delta \times \kappa \tag{4.1}$$

when buying and

$$p_x = ask_t - p_\Delta \times \kappa \tag{4.2}$$

when selling. For example, taking action $p_\Delta = 0$ when buying means placing an order for all remaining volume at $p_x = bid_t$. A larger positive p_Δ implies a more aggressive trade in the opposing book. Inversely, a larger negative p_Δ means placing a more passive order in the own side. An order will be replaced at each time step with a new order.

4.1.2 Rewards

The reward is defined as a continuous scalar and is calculated using the cash flows generated by the executed order relative to the mid price of the start of the episode. This results in a cost c_t at time step t as in Equation 4.3 in the case of buying. The cost is negative in the case of selling.

$$c_t = cash_t - m_0 \times (v_t) \tag{4.3}$$

As we generally want to maximize rewards in RL, we simply use the negative cost and define it as the reward $r_t = -c_t$.

4.1.3 Baseline Strategies for Part I

For Part I we will apply three simple baseline strategies. These are Immediate Execution (IE), Submit & Leave (SL) and Constant Policy (CP). Note that all of them are forced to execute the remaining volume (if any) at the end of the episode.

Baseline strategy 1: Instant execution (IE)

The simplest strategy is to submit a market order for the full volume at the beginning of the episode. This ensures that all volume is executed and reduces the risk of unfavourable price movement. However, by selling or buying everything at once, the investor will effectively get worse prices when orders are consumed in the opposing book, as found by [8]. IE acts as a worst case baseline to show what a market order would cost.

Baseline strategy 2: Submit and leave (SL)

Another strategy, presented by Nevmyvaka, Kearns, Papandreou, and Sycara [8] is the *Submit-and-Leave* policy. In the SL strategy one submits a limit order in the book at a price for the whole volume at the beginning of the period and then wait until the end of the episode to sell/buy all remaining volume. This strategy works as a trade-off between risk of non-execution and a low cost. Another advantage, the authors argue, is that costly monitoring of the price is reduced. However, with technological advancements, this monitoring is often performed cheaply by computers today. In their paper, they show empirically that this strategy performs better than instantly executing all volume (IE) [8]. SL act as a baseline to compare the performance of Model 1 and to verify results from [9].

Baseline strategy 3: Constant Policy (CP)

We introduce Constant Policy (CP), a baseline strategy that places an order for all the remaining volume at a constant number of ticks from the bid if buying or ask if selling. This order is replaced with a new order a given number of times (depending on the problem setting). Each time the order is replaced, it will be placed with the remaining volume left to execute and with a new price relative to the bid or ask for that particular time point. CP is added to evaluate if a model with a higher degree of freedom (being able to select different actions at different points in time) learned a successful policy.

4.1.4 Model 1 - NFK

In this section, a model fusing dynamic programming and Q-learning will be presented for finding an optimal policy to the optimized trade execution problem. The presented methodology is based on Nevmyvaka, Feng, and Kearns [9], hence the name NFK. As mentioned before, the goal is to find a policy which buys or sells a certain volume V in a limited time horizon H to the best possible price.

State

The environment will return the remaining volume $v_{remaining}$ and the elapsed time t. These are called private variables. We will also make use of market variables, i.e. variables that are derived from the order book (independent of the agent's actions). We will for this model make use of three market variables: immediate market order cost (imoc), bid-ask volume misbalance (bavmb) and the spread as defined in Equations 4.4 to 4.6. Here v_t^a and v_t^b represents the volume at the ask and bid level respectively.

$$spread_t = ask_t - bid_t \tag{4.4}$$

$$bavmb_t = v_t^b - v_t^a \tag{4.5}$$

$$imoc_t = m_t \times v_t - cash_t \tag{4.6}$$

Discretizing the state

NFK uses a table for the state-representation. To limit the number of entries in the tables, private and market variable will be discretized. The private variables are the time step t in the episode and the inventory left i. The remaining volume will be divided by a volume interval V/I and then rounded up, creating the private variable $i = \lceil v_{remaining}/(V/I) \rceil$. The elapsed time is divided into T time steps. I and T are the maximum values of i and t. All market variables will be limited to three integer values. All market variables are computed for each time step in the data set and then discretized. This is possible since the agent's actions are assumed not impact the market variables. Immediate market order cost is discretized relative

which inventory it based on. If the agent has a state inventory $i = 3$ the immediate market order cost is based on the volume which that volume corresponds to. All market variables except the spread are discretized in three quantiles. The spread is discretized as $\kappa = 0$, $2\kappa = 1$, $> 2\kappa = 2$. The resulting state s is then a vector containing private and market variables $s = (t, i, spread_t, bacmb_t, imoc_t)$. The market variables is for a timepoint t in the dataset.

Algorithm

The policy will be learned using an action value table combined with a dynamic programming approach as in [9]. At time step T a market order (execute all volume) will be evaluated for all different $i = 1, ..., I$. The algorithm will then solve $t = T - 1, ..., 0$ inductively with a backward induction approach, see Algorithm 5. The action value function in this case is the expected cost of taking action a in state s and is updated according to Equation 4.7. c_t is the immediate cost of taking action a in s and will depend on t. s' is the next state when taking action a in s and a' is the best action taken in s'. Note that this algorithm can be implemented without market variables, simply by setting the state $s \leftarrow (t, i)$, this will also be done. a_{min}, $a_{max} \in \mathbb{Z}$ constitutes the given minimum and maximum actions for the algorithm to evaluate for each step.

$\mathcal{A} \leftarrow \{a_{min}, a_{min} + 1, ..., a_{max} - 1, a_{max}\}$
Initialize $C(s, a) = 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
for $t = T$ *to* 0 **do**
  while *not end of data* **do**
    Get market variables $\rightarrow spread_t, bacmb_t, imoc_t$
    Get data $\rightarrow \mathcal{L}_t, \mathcal{T}_t$
    for *i=1 to I* **do**
      for $a \in \mathcal{A}$ **do**
        $s \leftarrow (t, i, spread_t, bacmb_t, imoc_t)$
        $v_{remaining} \leftarrow \frac{i \times V}{I}$
        if $t = T$ **then**
          Translate $v_{remaining}$ into an market order x
        else
          Translate a and $v_{remaining}$ into an order x
        end
        $v_t, cash_t \leftarrow match\_order(x, \mathcal{L}_t, \mathcal{T}_t)$
        $i' \leftarrow \lceil (v_{remaining} - v_t)/(V/I) \rceil$
        Get the market variables of the next time step
        $\rightarrow spread_{t+1}, bacmb_{t+1}, imoc_{t+1}$
        $s' \leftarrow (t + 1, i', spread_{t+1}, bacmb_{t+1}, imoc_{t+1})$
        Calculate $c_t(s, a)$
        Look up $\min_{a'} C(s', a')$
        Update $C(s, a)$
      end
    end
  end
end

Algorithm 5: NFK(V,H,T,I, a_{min}, a_{max}).

With the update function below where n is the number of time the particular action a has been updated for state s.

$$C(s, a) \leftarrow \frac{n}{n+1} C(s, a) + \frac{1}{n+1} [c_t(s, a) + \min_{a'} C(s', a')] \tag{4.7}$$

4.1.5 Model 2 - Dual NFK

Since the problem is symmetric in the sell and buy case, one can instead of doing two separate cost tables for the buy and sell case, train one agent on both the buy and sell case simultaneously. This lets the agent experience both cases when training, thus potentially make the agent robust against price movement. This is the motivation of extending NFK into what we call Dual NFK. The state space, action space, cost table update and experimental settings are the same. However, to isolate the dual factor and limit training time we choose to not evaluate this model with market variables.

$\mathcal{A} \leftarrow \{a_{min}, a_{min}+1, ..., a_{max}-1, a_{max}\}$
Initialize $C(s,a) = 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
for $t = T$ *to* 0 **do**
  while *not end of data* **do**
    Get data $\rightarrow \mathcal{L}_t, \mathcal{T}_t$
    for *i=1 to I* **do**
      for $a \in \mathcal{A}$ **do**
        for *both the sell and the buy case* **do**
          $s \leftarrow (t, i)$
          $v_{remaining} \leftarrow \frac{i \times V}{I}$
          if $t = T$ **then**
            Translate $v_{remaining}$ into a buy or sell market order x
          else
            Translate a and $v_{remaining}$ into a buy or sell order x
          end
          $v_t, cash_t \leftarrow match\_order(x, \mathcal{L}_t, \mathcal{T}_t)$
          $i' \leftarrow \lceil (v_{remaining} - v_t)/(V/I) \rceil$
          $s' \leftarrow (t+1, i')$
          Calculate $c_t(s,a)$
          Look up $\min_{a'} C(s', a')$
          Update $C(s,a)$
        end
      end
    end
  end
end

Algorithm 6: Dual NFK(V,H,T,I, a_{min}, a_{max}).

4.1.6 Model 3 - Sarsa(λ)

NFK and Dual NFK maximizes data-efficiency by evaluating all actions for all (private variable) states for all training data. Sarsa(λ), the third model to be evaluated, will instead start at a random time point in the training data and follow an ϵ-greedy policy to explore the environment. This introduces the problem of exploration vs. exploitation. For this reason, we choose not to evaluate any market variables for this model since it would increase the state space significantly and lead to very slow convergence. However, with a non-exhaustive search we will reduce the risk of evaluating extreme actions at extreme time points, i.e. encounter outliers in the rewards which may affect the learning. Additionally, when starting randomly in the environment, we also randomly assign the agent to either buy or sell. We thus let the agent experience both cases, for the same reason Dual NFK was developed. The other main difference between NFK and Sarsa, and also the main motivation for applying Sarsa(λ), is the fact that Sarsa(λ) does not update its action-value table with a one step look-ahead. NFK

does this by exploiting the Markov assumption. We wish to relax this Markov assumption by taking several succeeding rewards into account when updating the action-value table.

Rewards

The problem with using a cost relative to the mid price in the beginning of the episode (as NFK does) is that the absolute value of the reward might be considerably large if the price has moved away from the start price of the episode. The magnitude of the reward will thus only partly be due to the action taken. This is problematic since the agent will have trouble distinguishing if an action caused a high reward, or if it was due to a price movement. NFK can handle this since it does an exhaustive training of all data, and is thus not so sensitive to price movement. For Sarsa(λ), we choose another reward function during the training. This reward is relative to the mid price of that time step t instead. The cost expressed in Equation 4.8 is for the buy case, the sell case is simply negated.

$$c_t = cash_t - m_t \times v_t \tag{4.8}$$

Where the reward is the negative cost $r_t = -c_t$. The intuition why this reward function might work is that if the agent learns to take the best action relative to the mid price at each time step, it will in total learn to take the optimal trajectory (since we assume that it is a Markov process). Of course, it will not receive feedback if the price has moved. However, with the limited state space, Sarsa(λ) is unfortunately exposed to this risk, and cannot predict it with any market variables.

Algorithm

The degree of bootstrapping can be regulated by the constant $\lambda \in [0, 1]$, where $\lambda = 0$ represents a one step look-ahead (TD-backup) and $\lambda = 1$ leads to a full-episode update (MC). We use eligibility traces $E(s, a)$ (a counter table) for doing so in a backward view manner when implementing this algorithm. In Algorithm 7 we present the pseudo-code of the implementation used. ep represents the number of episodes. $N(s, a)$ is a counter table used for creating a learning rate which results in an average of each state and action. $N(s)$ is a counter table dependent on only state to keep track of exploration level together with N_0, a constant regulating the level of exploration.

Initialize $Q(s,a) = 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
Initialize $N(s,a) = 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
Initialize $N(s) = 0, \forall s \in \mathcal{S}$
for $e = 0$ *to* ep **do**
  $E(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S from environment
  $N(S) \leftarrow N(S) + 1$
  $A \leftarrow \epsilon$-greedy$(S, N(s), N_0, Q(S,A))$
  for *each step of the episode* **do**
    Take action A, observe R, S'
    $N(S') \leftarrow N(S') + 1$
    $A' \leftarrow \epsilon$-greedy$(S', N(S), N_0, Q(S,A))$
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S,A)$
    $E(S,A) \leftarrow E(S,A) + 1$
    $N(S,A) \leftarrow N(S,A) + 1$
    for $s \in \mathcal{S}$, $a \in \mathcal{A}$ **do**
      $\alpha \leftarrow \frac{1}{N(s,a)}$
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
      $E(s,a) \leftarrow \gamma\lambda E(s,a)$
    end
    $S \leftarrow S'; A \leftarrow A'$
  end
end

Algorithm 7: Sarsa(λ,ep,N_0, V,H,T,I, a_{min}, a_{max}).

With the following ϵ-greedy scheme.

Function $\epsilon - greedy(s, N(S), N_0, Q(S,A))$ **:**
  $\epsilon \leftarrow \max(0.1, \frac{N_0}{N_0 + N(s)})$
  $\iota \sim \mathcal{U}(0,1)$
  if $\iota < \epsilon$ **then**
    $a \sim \mathcal{U}\{\mathcal{A}\}$
  else
    $a \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s,a)$
  end
  return a

Where N_0 regulates the level of exploration. Note that ϵ will never go below 0.1 to ensure that the agent will not stop exploring, as it cannot learn better policies if it doesn't explore.

4.2 Part II

We will in this section present the models and baseline strategies of Part II. As previously mentioned, most of the environment of Part II is built upon the environment of Part I with an extended action space being the most important difference between the two.

4.2.1 Extending the Action Space

To reduce transaction cost further we introduced the possibility to control the volume ω_x of the order. All the models for Part II can control both the price and volume of the order. This is an extension of the action space $a = (p_\Delta, v_\%), p_\Delta \in \mathbb{Z}, v_\% \in \mathbb{R}$. The price of the order p_x is the same as in Part I. The volume of the order ω_x is set as a percentage of the remaining

volume.

$$\omega_x = v_{remaining} \times v_{\%} \tag{4.9}$$

4.2.2 Baseline Strategies for Part II

With this extended action space we can apply two new, potentially better, baseline strategies. These are Constant Policy With Volume and Equally Distributed. These two are also forced to execute the remaining volume (if any) at the end of the episode.

Baseline strategy 4: Constant Policy With Volume (CPWV)

We introduce Constant Policy With Volume (CPWV). This baseline strategy, as the name suggests, is exactly like CP with the addition that it also places a fixed percentage of the remaining volume instead of the all the remaining volume at each order placement.

Baseline strategy 5: Evenly distributed (ED)

In practice, a common strategy is to break up a large order into smaller trades of equal size. This removes risk of non-execution, avoids opportunity cost and lowers the instant market impact of the market orders. This has also shown to be a good strategy in theory. Under the assumptions that price impact is linear in the trade size and the price follows an arithmetic random walk, Bertsimas and Lo [7] showed with dynamic programming that the strategy to evenly split up the order into equally sized smaller orders is optimal.

4.2.3 Model 4 - PPO

We decided to evaluate PPO for three main reasons. First, after evaluating ED we realized that one can reduce transaction cost by controlling the volume of the order. Second, NFK performed better with market variables, so a model that can handle a larger state space is preferred. Third, since our reward is stochastic a model that use a large amount of data to update its policy will be less sensitive to outliers in the reward function. PPO is suitable for these three reasons. It can use a multidimensional and continuous action space, we use a NN as function approximation for its policy to handle a larger state space and selecting the amount of data used for policy updates is a parameter.

The neural network is trained with the training algorithm and loss function as in PPO, see Section 3.4.4. A graphical representation of the neural network is given in Figure 4.2. The model uses a continuous action space of two dimensions. It estimates the vector of means and the vector of logarithmic standard deviations of a multivariate normal distribution. No estimation of covariances are made, the covariance matrix is $\sigma(s)^2 I$. We tested with both estimating the logarithmic standard deviations and without. If the model does not estimate the logarithmic standard deviations they are set to a constant for all estimated means, in our case to one. Our network estimates the price delta p_Δ and the percentage of remaining volume $v_{\%}$ to be ordered. The estimated action is rounded to closest integer. Consider a buy case, where $bid_t = 10$, the tick-size $\kappa = 1$, and $v_{remaining} = 10$. If the network outputs $\mu_{price}(s) = -1.3$ and $\mu_{volume-pct} = 0.73$ the order is then $x = (p_x = 10 + 1 \times round(-1.3) = 9, \omega_x = round(10 \times 0.73) = 7, \tau_x = 1)$. The network also outputs an estimate of the value function for a given state. The neural network consists of two layers. Two layers and 64 hidden neurons in each layer with tanh activation function was used in [13].

PPO was trained in the same way as Sarsa(λ), randomly starting an episode at a random timestamp in the training data. Though it was not trained on both the buy and sell case, one model was trained for each. This since the market variables can make the problem asymmetric where the policy acts differently on a market variable's value.

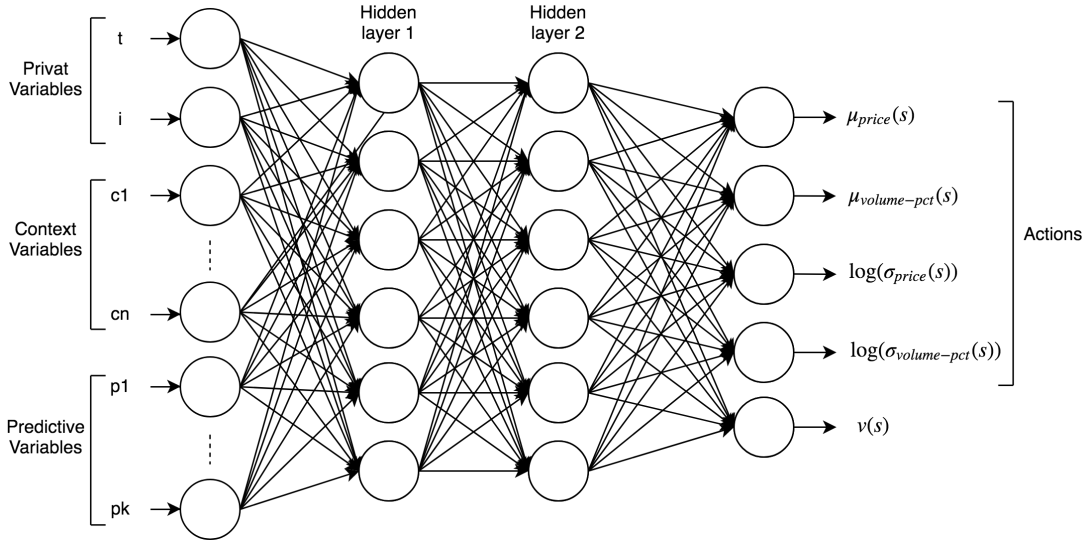

Figure 4.2: The Neural Network used to approximate the policy in PPO. The input is a state s consisting of private, context and predictive variables. Two hidden layers is used with a certain number of neurons with an activation function. The outputs are the mean and logarithmic standard deviation of a normal distribution for both the price and percentage of remaining volume. We tested with estimating the standard deviations and setting them fixed to one. There is also an output for the approximation of the value function given the state s.

Input Variables

PPO will use the same private variables t and $v_{remaining}$ as previous models. Both t and $v_{remaining}$ was divided with T and V respectively. Since the network can interpolate between states it trains more efficiently and adding more input variables is feasible. We have divided the market variables in context variables and predictive variables. The context variables describe the current state of the market and the predictive variables gives the neural network information about price movement.

Context Variables

Because we saw that the activity of the market differs during the day (See Appendix B.1). We added the current minute of the day, the past aggregated trades $\mathcal{T}_{t-1}$ price standard deviation and sum of volume as context variables. We define an aggregated trade as z_t^i with price level i, that consists of a price and volume $z_t^i = (p_{z_t^i}, \omega_{z_t^i})$. In Equation 4.12 Std refers to the standard deviation.

$$min_t = (hours_t \times 60 + minutes_t)/(24 \times 60) \tag{4.10}$$

$$volume_t = \sum_{z^i \in \mathcal{T}_{t-1}} \omega_{z^i} \tag{4.11}$$

$$std_t = Std(\{p_{z^i} | z^i \in \mathcal{T}_{t-1}\}) \tag{4.12}$$

Predictive Variables

A previous master thesis at Lynx Asset Management studied predictive variables to forecast the mid price of the order book had been performed (see Pärlstrand [35]). The predictive variables with the most predictive power (see [35]) are presented in this section. First, the variables will be presented and then the settings for each variable will be summarized in a

table. The first variable, order flow imbalance (OFI), has been proposed to drive price changes [36]. Let e_t be the signed volume changes at time t. Consider the bid_t and ask_t with respective bid and ask volume v_t^b, v_t^a at time step t. If the bid remains the same, then $e_t = v_t^b - v_{t-1}^b$. If the bid increases, then $e_t = -v_t^b$. If the bid decreases, then $e_t = v_{t-1}^b$. Analogous definitions can be made for the ask side but with reversed signs. The order flow imbalance attempts to capture the demand and supply of the ask and bid price.

$$\begin{aligned} e_t = \quad & \mathbb{1}(bid_t >= bid_{t-1})v_t^b - \mathbb{1}(bid_t <= bid_{t-1})v_{t-1}^b \\ & - \mathbb{1}(ask_t <= ask_{t-1})v_t^a + \mathbb{1}(ask_t >= ask_{t-1})v_{t-1}^a \end{aligned} \tag{4.13}$$

The OFI over a time period $[t_{k-L}, t_k]$ is then defined as

$$OFI_k(L) = \sum_{t=k-L}^{k} e_t \tag{4.14}$$

For the following variables the simple and exponential moving average must be defined. Consider a price series Z_t. A simple moving average (SMA) of length L is the average over the last L data points.

$$SMA(Z_t, L) = \frac{1}{L} \sum_{i=0}^{L-1} Z_{t-i} \tag{4.15}$$

The exponential moving average (EMA) of the price series with decay rate λ is defined as

$$EMA(Z_t, \lambda) = \lambda \sum_{i=0}^{\infty} (1-\lambda)^i Z_{t-i} \tag{4.16}$$

Instead of specifying a decay rate λ, one can specify a length L as in SMA where the relationship between λ and L is $\lambda = \frac{2}{L+1}$. The next variable, the moving average convergence divergence (MACD) is based on the convergence and divergence of two exponential moving averages. One with shorter time length L_1 and one with longer time length L_2. MACD is a trend indicator that tries to capture price movements. For example, when MACD is high the EMA with short time may have risen, which could mean that the price has risen fast but will return to previous levels.

$$MACD(Z_t, L_1, L_2) = EMA(Z_t, L_1) - EMA(Z_t, L_2) \tag{4.17}$$

The next two variables uses the open p_t^o, high p_t^h, low p_t^l and close p_t^c price during time period $t-1$ and t. The Chaikin (CHV) volatility measures the rate of change between two exponential moving averages. A fast increase(decrease) in CHV is a sign of a low(top) in the market [37].

$$CHV(p_t^h, p_t^l, L_1, L_2) = \frac{EMA(p_t^h - p_t^l, L_1)}{EMA(p_{t-L_2}^h - p_{t-L_2}^l, L_1)} - 1 \tag{4.18}$$

The average true range (ATR) is based on the true range (TR) and measures absolute price changes which reflects the volatility in absolute terms [35]. Volatility affects the probability of execution of an order [38]. Therefore, measuring the volatility acts as a proxy for estimating the probability of execution.

$$\begin{aligned} TR(p_t^h, p_t^l, p_t^c) &= \max(p_t^h, p_{t-1}^c) - \min(p_t^l, p_{t-1}^c) \\ ATR(p_t^h, p_t^l, p_t^c, L) &= SMA(TR_t(p_t^h, p_t^l, p_t^c), L) \end{aligned} \tag{4.19}$$

A summary of all the variables and the settings used for them will be presented in the Section 5.7.6.

Reward

PPO used the same reward as in NFK, where the cost is measured against the mid price of the initial episode timepoint m_0 and the reward is the negative cost. The reward is then clipped to a maximum of 50 and a minimum of -50. This was performed because large price movements can occur, making the cost relative the initial mid price very large or very small. This can be due to external events which the agent does not have information about. After the clipping, the reward was also divided by 10 to reduce the scale of the reward. Remember that the reward is measured in ticks. A reward of 10 means that each lot was sold/bought at a price of 10 ticks better than the compared price. The clipping and scaling was only performed during training.

Chapter 5
Method

In this chapter we first present the characteristics of the data used in this study. We then present of method of evaluation and how our models were trained and how hyperparameters were chosen. Additionally, we present the problem settings for both Part I and II.

5.1 Data

The simulated market where the execution will take place is based on historical data. The characteristics of each data set is presented in Table 5.1. Common for all financial assets are that they are futures contracts.

Table 5.1: Characteristics of the different data sets used for this thesis project. Average daily volume (ADV) are presented in thousands over the data period (2016).

Underlying asset	Asset class	Exchange	Tick-size	ADV
Crude Oil (WTI)	Commodity	Chicago Mercantile Exchange	0.01	271
FTSE 250	Stock Index	Intercontinental Exchange	0.5	84
Gilts	Government bonds	Intercontinental Exchange	0.01	153
NASDAQ 100	Stock Index	Chicago Mercantile Exchange	0.25	199

The data will be divided into train and test sets divided by time, with training being performed on the first year of the data set (2016) and tested on the succeeding half a year (first half of 2017).

Figure 5.1: The training and test set.

5.2 Evaluation

All models and strategies was evaluated in terms of a cost C. This cost is defined as the difference between the cash flow achieved by the (unrealistic) scenario of selling/buying the quantity at the mid-price in the beginning of the period and the actual cash flow generated by the model or strategy. The cost for each episode is defined in Equation 5.1. V is the total

volume to be bought/sold during the episode and κ is the instrument's tick size. Dividing by V and the tick size enables comparison between instruments and volumes. The evaluation will be the average cost performed on all episodes $e \in E$ in the data, which can either be training or test data. If we evaluate an algorithm where the episode horizon $H = 8$, with opening hours from 8:30-16:30 (instrument FTSE) it will result in $8 \times 60 - 7 = 473$ episodes per day, due to that we test with overlapping episodes. The test data contains 103 opening days from 2017-01-01 to 2017-06-1 which results in $473 \times 103 = 48719$ number of episodes.

$$C_e = \frac{1}{V \times \kappa} \sum_{t=0}^{T} c_t \tag{5.1}$$

$$\bar{C} = \frac{1}{|E|} \sum_{e \in E} C_e \tag{5.2}$$

All models will be evaluated both in the case of buying and selling. This is performed since price movements affects the results. The evaluation metric for all models is $\bar{C}_{avg}$ which is an average over the buy and sell case. This is the final metric we will use for all models to evaluate its performance.

$$\bar{C}_{avg} = \frac{\bar{C}_{sell} + \bar{C}_{buy}}{2} \tag{5.3}$$

The standard deviation of the episode costs C^{std} will also be evaluated to measure how much spread there is in the mean episode cost $\bar{C}$. This will be averaged over the case of selling and buying resulting in $\bar{C}^{std}_{avg}$.

$$C^{std} = \sqrt{\frac{\sum_{e \in E}(C_e - \bar{C})^2}{|E| - 1}} \tag{5.4}$$

$$\bar{C}^{std}_{avg} = \frac{C^{std}_{buy} + C^{std}_{sell}}{2} \tag{5.5}$$

5.2.1 Participation Rate

Instead of using one fixed volume to execute for all instrument as in Nevmyvaka et al. [9], a few participation rates were chosen which can be converted to a certain volume depending on the instrument and horizon. Participation rate can be seen as the average share of participation our volume will constitute compared to the market as a whole. We translate this to an actual volume as in Equation 5.6. In the equation, Average Daily Volume (ADV) of the instrument over the training period and Minutes Per Day (MPD) is the number of minutes that we are active in the market. H is the length of the trading horizon in minutes. The volumes V and ADV are rounded to the closest integer.

$$V = \frac{PR \times ADV \times H}{MPD} \tag{5.6}$$

By doing this, we partly exclude the liquidity factor of trading an instrument, i.e. if an instrument is more liquid, we demand more volume to be bought/sold to challenge the agent accordingly.

5.3 Optimizing Baselines

Both SL and CP has a price distance from the bid if buying and ask if selling that can be optimized. This distance was chosen by testing all possible actions $a \in [-8, -7, ..., 7, 8]$ in the training data. CPWV has both a price distance and a percentage of remaining volume

to be set. Therefore, a grid search on the price ($[-8, -7, ..., 7, 8]$) and percentage of remaining volume ($[0.1, 0.2, ..., 0.9, 1.0]$) was performed and evaluated with $\bar{C}$. The action that yielded the lowest average cost $\bar{C}$ was then selected as the a to be evaluated. The optimal action will depend on various settings of the environment, so this optimization must be repeated for each problem setting, such as instrument, horizon of the episode, participation rate and number of time steps.

5.4 NFK

How the data in Algorithm 5 is iterated over is not explicitly explained in [9]. We selected to start from the last time point (chronologically) and then iterate backwards. Meaning that, line 2 in Algorithm 5 is performed such that the algorithm starts at the last time point and then the previous one until the first time point. This means that we have overlapping episodes during training. NFK will be trained both with and without the market variables from Equation 4.4 to 4.6 to measure the impact of using the market variables. This was performed on one setting to test if the market variables could reduce the transaction cost.

5.5 Hyperparameter Tuning of Sarsa(λ)

The implementation of Sarsa(λ) has three hyperparameters; λ, N_0 and the number of episodes ep. These were partly chosen by heuristic and experimentation. After some knowledge was gained while testing various parameters on a couple of problem settings, the number of episodes ep was chosen so that the model would only run within a reasonable running time. Given ep, various combinations of λ and N_0 was tested. All evaluation of parameters was done on training data in terms of the cost $\bar{C}$. The test data was saved for evaluation. The tests were also performed on a couple of problem settings and instruments due to the time scope of the thesis.

5.6 Experimental settings for Part I

The experimental settings for Part I is based on Nevmyvaka et al. to facilitate comparison between our results and theirs [9]. However, some differences needed to be made to translate the problem to our setting. The main difference was that our matching simulation is limited to only trade during minute shifts, i.e. when one minute shifts to another. To be able to have the same time resolution as Nevmyvaka et al., we had to scale 2 and 8 minutes up to 8 and 32 minutes respectively. The participation rates were chosen according to the approximate participation rate that the authors had ($\sim$ 1-13%). Since the volume we chose depends on the horizon, a longer horizon results in a larger volume to trade. For all four instruments in the data set all combinations of PR, H, I, and T in Table 5.2 was tested. Testing all these values enables the analysis of how the agent will act in different situations.

Table 5.2: Settings to be tested with NFK.

PR	1%, 5%, 10%
H	8 min, 32 min
I	4, 8
T	4, 8
a_{min}	-8
a_{max}	8

For both the buy and sell case (NFK and Baseline Strategies) we have $4 \times 3 \times 2 \times 2 \times 2 = 96$ combinations of problem settings. For the dual case (Dual NFK, Sarsa(λ) and baseline trate-

gies) we have additionally 96 problem settings. In total this results in 288 different problem settings to train and evaluate.

5.7 PPO

In this section we will present the choice of parameters for the optimizer, how hyperparameters where determined, the tests performed regarding PPO and the input variables.

5.7.1 Gradient Ascent Optimizer

The optimizer used for gradient ascent was ADAM with recommended parameters as in [23], $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

5.7.2 Hyperparameter Tuning of PPO

Authors of previous literature regarding DRL algorithms have tuned the hyperparameters on one task and then used the same hyperparameters on several others. Therefore, we decided to tune the hyperparameters in one setting, the case of selling for one instrument. The same hyperparameters is then used for all instruments and in the case of buying and selling. Since there are many hyperparameters and we had limited amount of time and hardware resources, an initial testing was first performed. The initial tuning consisted of measuring the training performance, $\bar{C}$, on 8 days of training data on one instrument, this to test stability and performance of certain parameters. Table 5.3 gives a summary of the model's parameters and how they were tested. The variables tested during the initial testing is marked as initial testing in the table. After the initial testing we fixed certain parameters, and a grid search was performed on the remaining parameters.

Table 5.3: Hyperparameters for PPO. The table gives a description over the hyperparameters and how they were selected. The parameters have been divided in to three categories; neural network-related, optimizer-related and loss function-related parameters. Hidden layers of [64, 64] means two hidden layers with 64 hidden neurons each.

Parameter	Description	Selection
Neural network parameters		
Estimate std.	If the model should also estimate std or not	Initial testing, either True or False
Hidden layers	Number of layers and number of neurons in each layer	grid search [[64, 64], [128, 128]]
f(z)	Activation function for all hidden layers	Initial testing of either ReLu or tanh
Optimizer parameters		
K	Number of times per iteration to perform gradient ascent	grid search [4, 8]
α	Learning rate of SGD	grid search [0.0005, 0.00005]
M	Minibatch size for SGD	grid search [10240, 20480]
U	Number of experience to collect before performing gradient ascent	Initial testing
IT	Number of training iterations to perform	Initial testing, until convergence
Loss function parameters		
k1	Value function loss coefficient	Initial testing
k2	Entropy loss coefficient	Initial testing, only if estimating standard deviation
ξ	Clipping parameter for loss function	grid search [0.1,0.2,0.3]
λ	GAE λ. 0 for TD-error update and 1 for MC-update	Initial testing of 0.7 and 1

5.7.3 Evaluation Metric Hyperparameters

Using $\bar{C}$ on the training data as an evaluation metric for the hyperparameter tuning is expensive because it requires thousands of episodes to be evaluated. Therefore, we created a different metric. Recall the training iteration i and temporary data set $\mathcal{D}$ of PPO's training algorithm from the Section 3.4.4. Here the data set $\mathcal{D}$ is available during training making it less expensive to use the episodes in the data set as evaluation. Also, because of hardware and time constraints, each hyperparameter setting training will run for 24 hours. This will result in a number of iterations IT for each hyperparameter setting. Due to that the training performance is fluctuating due to exploration of actions during training, the average of the last 25 iterations of the training will be used. The metric that is averaged is the average episode reward $\bar{r}_i$ from the temporary training data set $\mathcal{D}$. This result in the evaluation metric $\bar{h}$ defined as

$$\bar{r}_i = \frac{1}{|E_{\mathcal{D}}|} \sum_{e \in E_{\mathcal{D}}} r_e \qquad\qquad \bar{h} = \frac{1}{25} \sum_{i=IT-25}^{I} \bar{r}_i \tag{5.7}$$

Where $E_{\mathcal{D}}$ is all the episodes in $\mathcal{D}$ and r_e the reward for each episode e.

After the hyperparameters had been determined. The model was trained for all instruments and in the case of buying and selling. This will be performed for $IT = 500$ iterations, selected by observing convergence of $\bar{r}_i$. The weights at iteration i that generates the highest $\bar{r}_i$ will be saved and used on the test data.

5.7.4 Stability of Training

The training algorithm of PPO has three parts which includes random sampling that can result in different performance in terms of transaction cost in different runs. Firstly, the neural networks weights are sampled when initialized. Secondly, the actions are sampled from the policy in the training algorithm. Thirdly, the data is sampled from the temporary memory $\mathcal{D}$ in the training algorithm. This makes the training non deterministic. Because of this, we tested the stability of the training algorithm by running the same hyperparameters four times with different seeds on one instrument. We consider the model to have a stable training if it gives consistent results for multiple trainings, i.e the variance of the rewards for multiple trainings is observed to be low.

5.7.5 Preprocessing and Initialization

Because our agents' actions do not mechanically impact the order book and trades, the variables derived from the order book or aggregated trades could be precomputed. The variables where also scaled to have a minimum of zero and maximum of one. The market variables for the test data were scaled with the min and max of the training data. Initialization of the weight parameters θ was done by initializing the weight matrices W from a normal distribution with mean zero and standard deviation one. The bias terms b in the weight parameter were initialized as zeros.

5.7.6 Input Variables

All the input variables used for PPO and their setting are presented in Table 5.4. The data used for the open, high, low and close price series of the instruments trades have a time resolution in seconds. Meaning that the length L is used in seconds.

Table 5.4: Input variables for PPO, divided by category. The setting for each variable is also presented.

Category	**Private**		**Context**			**Predictive**						
Name	t	i	min_t	vol_t	std_t	$spread_t$	$bavmb_t$	imc_t	ofi_t	$macd_t$	atr_t	chv_t
Setting	-	-	-	-	-	-	-	-	$L = 1$	$L_1 = 120, L_2 = 240$	$L_1 = L_2 = 100$	$L = 140$

Market Variables

The hyper parameter tuning was performed with all input variables. To test the impact with context and predictive variables, we performed a training with only private variables, then with private and context variables and finally with all variables. This test was performed on one instrument.

5.8 Experimental Setting for Part II

All results for Part II will be for one problem setting due to time limitations. This problem setting is $H = 32$, $T = 8$ and $PR = 5\%$. This setting was chosen since it allows for comparison with the results from Part I.

5.9 Software and Hardware

The implementations of this thesis were written in Python. Important libraries used was Numba for faster processing of the trade simulation, Pandas for time series handling and Numpy for its data structures and numerical computing. NFK, Dual NFK and Sarsa(λ) was trained and evaluated on desktop computers. For the implementation of PPO, we used Ray RLlib which is built on top of Ray [39, 40]. Google Cloud Compute Engine (GCE) was used to for the hyperparameter tuning and training of PPO.

Chapter 6
Results

In this chapter we will present the results from implementing the models and baseline strategies. We first present how the models from Part I perform compared to the baselines strategies of Part I in terms of cost. We will also present various interesting results such as parameters found and policies learned (by both the models and the baseline strategies). We then present the results from Part II. In this section, we also show policies learned and hyperparameters found. We conclude the chapter with a table summarizing the results of all models and baseline strategies evaluated in this thesis. All results will be expressed in terms of the cost $\bar{C}_{avg}$ on test data if not otherwise specified.

6.1 Part I

6.1.1 NFK compared to IE, SL and CP

We will now show how NFK performs compared to the baseline strategies of Part I. To save space, we choose to only show resolutions of T=I=4 and T=I=8. The transaction cost $\bar{C}_{avg}$ for each setting, instrument and model can be seen in Table 6.1. To conclude these results, NFK generally performs worse than CP despite having the possibility of creating a more complex policy. However, NFK generally performs better than SL for longer horizons and larger participation rates.

Table 6.1: Cost table of $\bar{C}_{avg}$ on test data for multiple settings, instruments and settings. Bold font indicates the lowest cost for that setting.

	Horizon	H=32						H=8					
	Time	T=4			T=8			T=4			T=8		
	Inventory	I=4			I=8			I=4			I=8		
	PR	PR=0.01	PR=0.05	PR=0.1	PR=0.01	PR=0.05	PR=0.1	PR=0.01	PR=0.05	PR=0.1	PR=0.01	PR=0.05	PR=0.1
Instrument	Agent												
Crude	CP	**0.78**	**1.91**	**3.15**	**0.82**	**1.82**	**2.86**	**0.55**	**0.94**	**1.45**	0.63	**0.90**	**1.35**
	IE	1.96	8.81	17.09	1.96	8.81	17.09	0.86	2.38	4.59	0.86	2.38	4.59
	NFK	1.09	2.49	3.93	1.15	2.59	3.78	**0.55**	1.15	1.75	**0.53**	1.21	1.65
	SL	1.15	2.68	4.44	1.15	2.68	4.44	0.59	1.11	1.94	0.59	1.11	1.94
FTSE	CP	0.93	**2.04**	**2.83**	0.93	**1.95**	**2.44**	**0.55**	**1.03**	**1.37**	0.56	**0.99**	**1.20**
	IE	1.44	4.07	7.38	1.44	4.07	7.38	0.83	1.61	2.46	0.83	1.61	2.46
	NFK	1.35	2.56	3.42	1.39	2.33	2.98	0.97	1.34	1.81	0.97	1.37	1.55
	SL	**0.92**	2.17	3.53	**0.92**	2.17	3.53	**0.55**	1.05	1.52	**0.55**	1.05	1.52
Gilt	CP	**0.46**	**1.00**	**1.47**	**0.47**	**0.97**	**1.31**	**0.31**	**0.53**	**0.73**	**0.31**	**0.50**	**0.66**
	IE	0.99	2.83	5.13	0.99	2.83	5.13	0.63	1.11	1.69	0.63	1.11	1.69
	NFK	0.65	1.25	1.74	0.65	1.18	1.52	0.48	0.68	0.87	0.40	0.65	0.75
	SL	0.51	1.19	1.88	0.51	1.19	1.88	0.35	0.61	0.89	0.35	0.61	0.89
NASDAQ	CP	**1.36**	**3.14**	**4.46**	**1.38**	**3.04**	**3.97**	**0.77**	**1.56**	**2.16**	0.78	**1.52**	**1.95**
	IE	2.73	8.39	15.65	2.73	8.39	15.65	1.25	3.11	4.90	1.25	3.11	4.90
	NFK	1.74	3.81	5.15	1.79	3.45	4.62	1.18	2.18	2.79	1.20	2.03	2.47
	SL	1.49	3.61	5.77	1.49	3.61	5.77	**0.77**	1.70	2.52	**0.77**	1.70	2.52

6.1.2 Optimized Baselines of Part I

For all 360 (120 for buy, 120 for sell, 120 for dual) problem settings a grid search optimization that tested all $a \in \{-8, -7, ..., 7, 8\}$ to find the optimal price for Submit and Leave p_{Δ}^{SL} and Constant Policy, p_{Δ}^{CP}. This was done on the training data and the best price shift was then chosen for evaluation. Note that since SL and CP does not depend on inventory I, we only have 180 settings (60 for buy, 60 for sell, 60 for dual). Furthermore, SL action does not depend on the time resolution T, we thus only have 90 (30 for buy, 30 for sell and 30 for dual) unique counts of SL actions and 180 unique counts of CP actions. We count the number of times each price is optimal on all these settings:

Table 6.2: Counts of optimal actions for all problem settings.

Optimal price shift (p_{Δ})	**Counts for SL**	**Counts for CP**
-8	1	2
-5	2	10
-4	0	4
0	23	102
1	52	64
2	9	0
3	3	0

Passive actions were found to be best for long horizons with low participation rate, where the probability of matching all volume with a passive action is higher.

6.1.3 Resolutions of T and I

In Figure 6.1, the mean cost per episode achieved during the test data with different time and inventory resolutions of NFK is shown. We choose to only show the results for instrument Gilts, however the relative performance is similar, no matter what instrument.

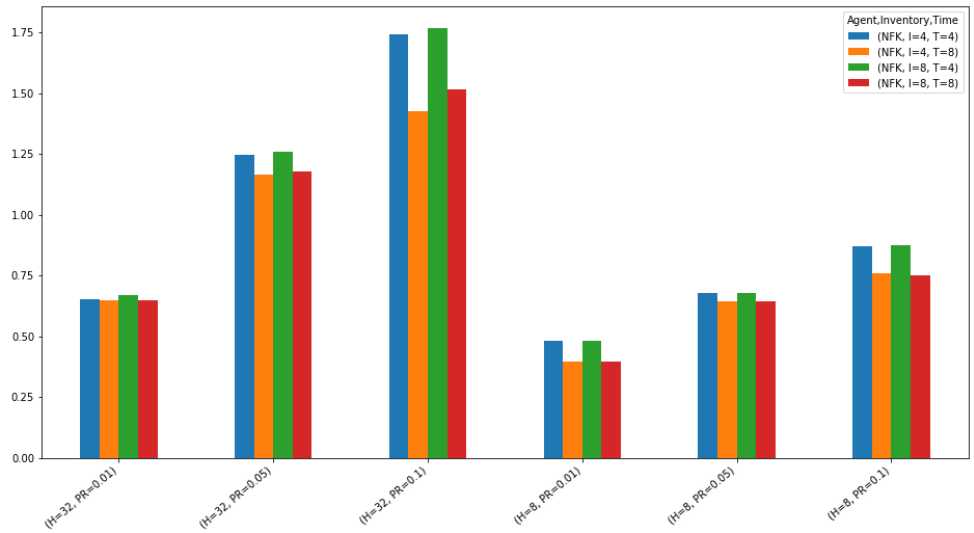

Figure 6.1: Cost of trading instrument Gilts.

Across instruments, horizons and participation rates, it can be concluded that higher resolution in T generally leads to better performance. However, a higher resolution of inventory I

does not necessarily lead to better performance. In fact, in some cases, a larger I lead to worse performance in the test phase, see e.g. H=32, PR=0.1.

6.1.4 Value Functions and Policies of NFK

We now present learned value functions and policies for NFK without market variables. Without the market variables we have a state $s = (t, i)$. Note that the cost table C learned has three dimensions (t, i, a) and is thus hard to visualize. Since we are only interested in the best action we now present the value function $V(s) = \min_{a \in \mathcal{A}} C(s, a)$ and policy $\pi(s) = \text{argmin}_{a \in \mathcal{A}} C(s, a)$. For practical reasons we will not present the results for all 240 problems settings (for NFK), but rather only some interesting and representative excerpts. In Figures 6.2 and 6.3 we show the policy and value function for a table trained with the sell case for instrument FTSE. Here, the participation rate was 5% and the horizon 8 minutes. The resolutions T and I was both 4 for this problem setting. In Figures 6.4 and 6.5 we show the policy and value function for the buy case for Crude. For this problem, the participation rate was 10% and the horizon was 32 minutes. Furthermore, in this problem the resolution of T and I was 8.

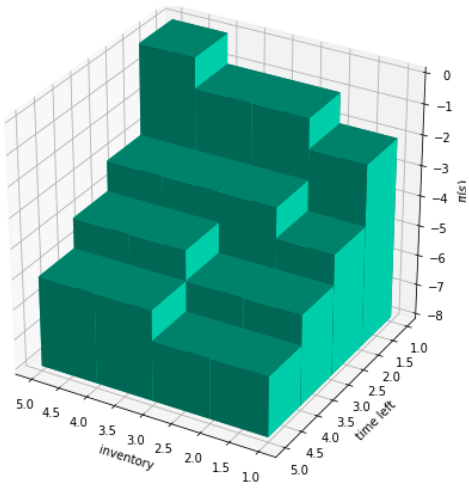

Figure 6.2: $\pi(s)$ trained with the sell case for FTSE, PR=5%, H=8.

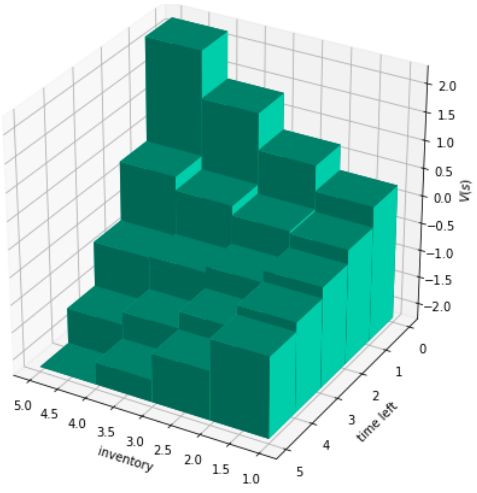

Figure 6.3: $V(s)$ trained with the sell case for FTSE, PR=5%, H=8.

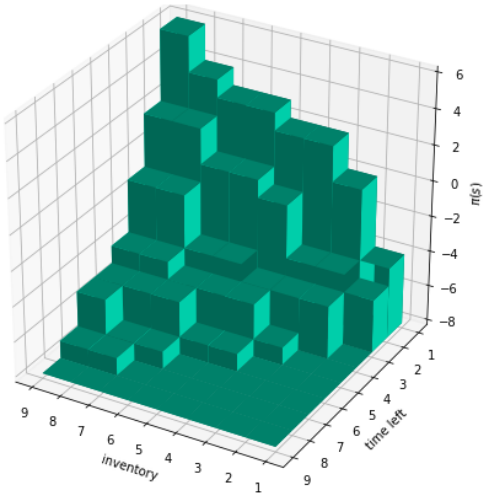

Figure 6.4: $\pi(s)$ trained with the buy case for Crude, PR=10%, H=32.

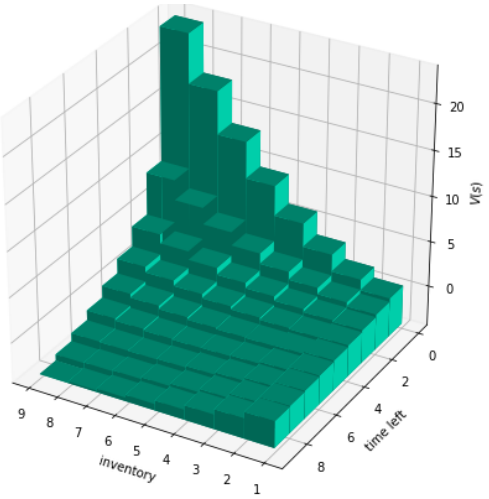

Figure 6.5: $V(s)$ trained with the buy case for Crude, PR=10%, H=32.

When the time runs out and the agent have a large inventory left to execute, the estimated expected cost from that state onwards $V(s)$ increases and the agent takes more aggressive actions $\pi(s)$ in order to execute its remaining quantity before the time runs out and the agent is forced to execute remaining inventory. This shape is more or less consistent across all problem settings.

6.1.5 Market Variables

The cost reduction in percentage when comparing NFK with and without the market variables are seen in Table 6.6.

Table 6.3: Cost reductions in terms of $\bar{C}_{avg}$ with market variables for NFK. The settings were $T = 8$, $I = 8$, $H = 8$ and $PR = 5\%$.

Instrument	Cost Reduction
Crude	1,9%
FTSE	6,5%
Gilts	3,77%
NASDAQ	2,5%

6.1.6 Dual NFK, Sarsa(λ) and Baselines

In this section we present the result of Dual NFK and Sarsa(λ) compared to the baseline strategies of Part I. In Table 6.1 the transaction cost for all the selected settings is presented. We observe that Sarsa(λ) has a lower cost than Dual NFK for all problem settings and even performs better than CP for some cases. However, for low participation rates both Submit and Leave and Constant Policy performs better than Sarsa(λ).

Table 6.4: Cost table $\bar{C}_{avg}$ on test data for multiple settings, instruments and agents. Bold font indicates the lowest cost for a setting.

	Horizon	H=32						H=8					
	Time	T=4			T=8			T=4			T=8		
	Inventory	I=4			I=8			I=4			I=8		
	PR	PR=0.01	PR=0.05	PR=0.1	PR=0.01	PR=0.05	PR=0.1	PR=0.01	PR=0.05	PR=0.1	PR=0.01	PR=0.05	PR=0.1
Instrument	Agent												
Crude	CP	**0.78**	**1.91**	**3.15**	**0.80**	1.82	2.86	0.55	**0.94**	**1.45**	0.59	0.90	1.35
	IE	1.96	8.81	17.09	1.96	8.81	17.09	0.86	2.38	4.59	0.86	2.38	4.59
	DUAL NFK	1.09	2.49	3.93	1.12	2.52	3.70	0.55	1.15	1.71	0.52	1.21	1.62
	SL	0.87	2.68	4.44	0.87	2.68	4.44	**0.47**	1.11	1.91	0.47	1.11	1.91
	Sarsa	0.95	2.22	3.24	0.68	**1.78**	**2.61**	0.49	1.04	1.47	**0.46**	**0.89**	**1.18**
FTSE	CP	0.93	**2.04**	**2.83**	0.93	**1.95**	**2.44**	**0.55**	**1.03**	**1.37**	0.56	**0.99**	1.20
	IE	1.44	4.07	7.38	1.44	4.07	7.38	0.83	1.61	2.46	0.83	1.61	2.46
	DUAL NFK	1.35	2.45	3.43	1.39	2.35	2.95	0.97	1.34	1.83	0.97	1.37	1.55
	SL	**0.92**	2.17	3.53	**0.92**	2.17	3.53	**0.55**	1.05	1.52	**0.55**	1.05	1.52
	Sarsa	1.36	2.38	3.07	1.43	2.12	2.51	1.04	1.37	1.49	0.91	**0.99**	**1.19**
Gilt	CP	**0.46**	**1.00**	**1.47**	**0.47**	**0.97**	1.31	**0.31**	**0.53**	**0.73**	**0.31**	**0.50**	0.66
	IE	0.99	2.83	5.13	0.99	2.83	5.13	0.63	1.11	1.69	0.63	1.11	1.69
	DUAL NFK	0.66	1.24	1.77	0.65	1.20	1.52	0.48	0.68	0.87	0.40	0.65	0.75
	SL	0.51	1.24	1.88	0.51	1.24	1.88	0.35	0.61	0.92	0.35	0.61	0.92
	Sarsa	0.64	1.14	1.50	0.55	1.04	**1.28**	0.45	**0.53**	**0.73**	0.34	**0.50**	**0.64**
NASDAQ	CP	**1.36**	**3.14**	**4.46**	**1.38**	3.04	3.97	**0.77**	**1.56**	**2.16**	0.78	**1.52**	**1.95**
	IE	2.73	8.39	15.65	2.73	8.39	15.65	1.25	3.11	4.90	1.25	3.11	4.90
	DUAL NFK	1.74	3.70	5.14	1.79	3.45	4.55	1.18	2.24	2.81	1.20	2.01	2.47
	SL	1.46	3.61	5.77	1.46	3.61	5.77	**0.77**	1.70	2.52	**0.77**	1.70	2.52
	Sarsa	1.68	3.43	4.70	1.54	**3.01**	**3.94**	1.08	1.81	2.36	1.01	1.59	2.00

6.1.7 Dual NFK Compared to NFK

Recall that Dual NFK is an extension of NFK that is trained on both the buy and sell case, thus training one agent instead of two. Both are evaluated for the buy and sell case (with NFK having one agent for each case). To compare the performance of Dual NFK to NFK, we calculated the metric $\frac{\bar{C}_{avg}^{DualNFK}}{\bar{C}_{avg}^{NFK}}$ for each problem setting. All these models were trained without market variables. The statistics of this metric is shown in Table 6.5

Table 6.5: Average reward for Dual NFK divided by average reward for NFK.

Number of settings	96
Mean	1.00
Standard deviation	0.01
Minimum value	0.96
Median value	1.00
Maximum value	1.03

On average for all settings, Dual NFK performs on par to NFK.

6.1.8 Parameters of Sarsa(λ)

To find appropriate hyperparameters for Sarsa(λ), experimenting was done for several problem settings and instruments. However, the results presented here is for one instrument and setting (Crude, PR=5%, I=8, T=8, H=8). The first parameter, number of episodes, was fixed to limit the duration of the running time. It was found that 4×10^6 episodes had a reasonable running time with the possibility of convergence. To choose λ, we trained a model with $\lambda \in \{0.0, 0.1, ..., 0.9, 1.0\}$ on the first 10 months of the training data and evaluated the last two months of the training data. It was found that lower λ led to better convergence as seen in Figure 6.6.

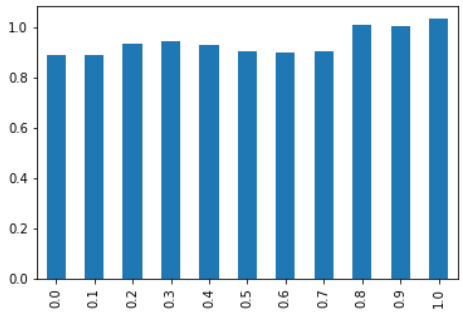

Figure 6.6: $\bar{C}_{avg}$ of Sarsa(λ) on part of the training data for various choices of λ.

The differences of performance are not large but if we compare the policies in Figures 6.7 and 6.8 we can see that Sarsa($\lambda = 0.1$) has converged to a less jagged and more reasonable policy than Sarsa($\lambda = 0.9$).

All though the shape of the surface plots for $V(s)$ and $\pi(s)$ shown before are interpretable, we argue the actual values are hard to distinguish. We use heatmaps for both the value function and the policy. Note that axes have been reversed for the heatmaps.

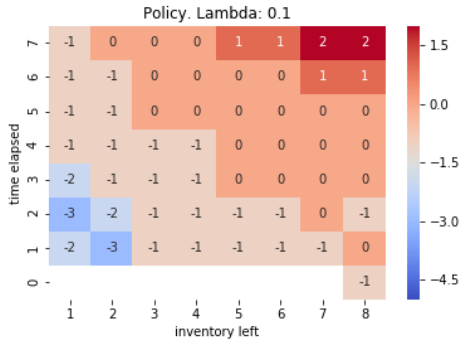

Figure 6.7: $\pi(s)$ for $\lambda = 0.1$.

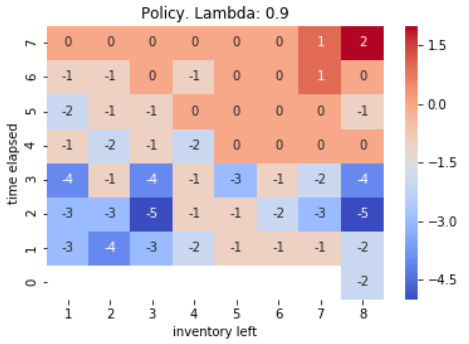

Figure 6.8: $\pi(s)$ for $\lambda = 0.9$.

We choose $\lambda = 0.1$ for the final model. Furthermore, a similar test was done for various parameters of N_0, a parameter regulating the level of exploration. It was found that a rather long exploration led to better performance. The final choice for $N0$ was 10^5.

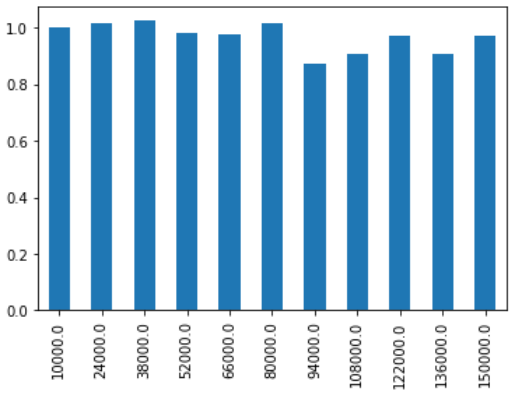

Figure 6.9: $\bar{C}_{avg}$ of Sarsa(λ) for various choices of N_0.

6.1.9 Sarsa(λ) compared to Dual NFK

Sarsa(λ) generally performs better than Dual NFK for all instruments. However, the performance differs depending on the instrument. We measure this difference with the metric $\frac{\bar{C}_{avg}^{Sarsa(\lambda)}}{\bar{C}_{avg}^{DualNFK}}$. The statistics of this measure for all 24 (2 H, 2 I, 2 T and 3 PR) problem settings per instrument are shown in Table 6.6. Both these are evaluated without market variables.

Table 6.6: Average reward for Dual NFK divided by average reward for NFK for each problem setting.

Measure	Crude	FTSE	Gilt	NASDAQ
Mean	0.80	0.93	0.86	0.87
Standard deviation	0.09	0.11	0.06	0.05
Minimum value	0.61	0.72	0.77	0.79
Median value	0.82	0.92	0.85	0.87
Maximum value	0.99	1.07	1.03	0.96

To explore why Sarsa(λ) performs better, we compare the policy of the two models where the performance differs the most (instrument Crude Oil), PR=0.01, I=8, H=32, T=8). For this setting, Sarsa(λ) performs on average 39% better on the test data.

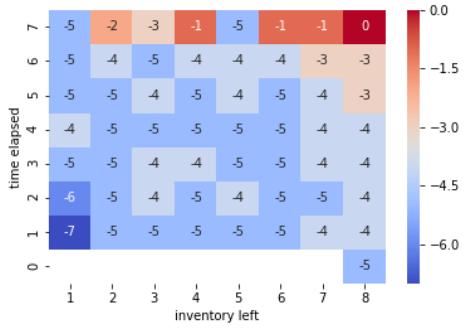

Figure 6.10: $\pi(s)$ for Sarsa(λ).

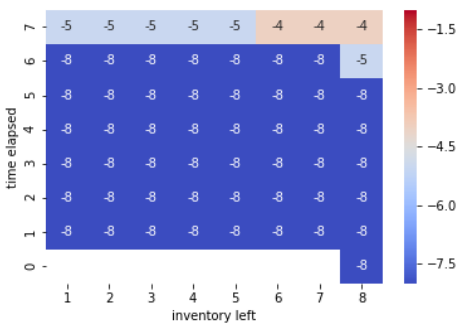

Figure 6.11: $\pi(s)$ for NFK.

As seen in Figures 6.10 and 6.11, Dual NFK learns a much more passive policy than Sarsa(λ). Even though Sarsa(λ) has not fully converged to a monotonic behaviour, it performs a lot better.

6.2 Part II

Recall from Section 5.8 that for Part II only one setting was evaluated. The result for this experiment can be seen in Table 6.7, where PPO outperforms the baseline strategies for all instruments.

Table 6.7: Cost table $\bar{C}_{avg}$ Part II. Setting: $H = 32$, $T = 8$, $pr = 0.05$, $I = 4$. Bold font represents the lowest transaction.

Strategy\Instrument	**Crude**	**FTSE**	**Gilt**	**NASDAQ**
CPWV	0.97	0.90	0.54	1.41
ED	1.32	1.09	0.78	1.92
PPO	**0.73**	**0.77**	**0.32**	**1.31**

6.2.1 Grid search CPWV

To find the optimal action of the extended action space $a = (p_\Delta, v_\%)$ (action space of Part II) a grid search of all actions $p_\Delta \in [-8, -7, ..., 7, 8]$ and $v_\% \in [0.1, 0.2, ..., 0.9, 1.0]$ was performed on the training data for each instrument. The action that yielded the lowest cost $\bar{C}_{avg}$ (average of buy and sell) are shown in Table 6.8. These actions were chosen for evaluation on the test data.

Table 6.8: Optimal constant policy found for each instrument.

Instrument	**Price shift (p_Δ)**	**Percentage of remaining volume ($v_\%$)**
Crude	0	30%
FTSE	1	30%
Gilt	0	30%
NASDAQ	1	30%

6.2.2 Hyperparameters and Training Stability

The full results of the hyper parameter tuning can be seen in Appendix D. The selected hyperparameters are presented in Table D.2. The results regarding the training stability of PPO

is given in Figure 6.12, which shows the training performance $\bar{r}_i$ for each training iteration i. There are differences between the trainings but the observed variance is not high.

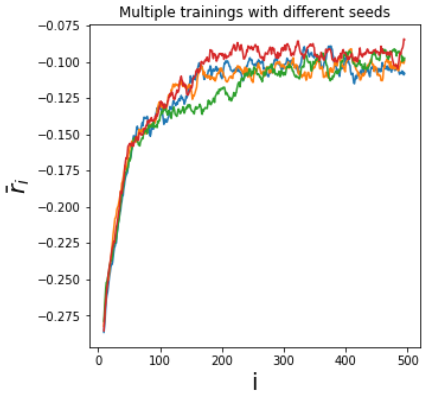

Figure 6.12: Training stability. Since the weights are randomly initialized and the agent samples actions during training the training performance can vary over different runs. The plot shows the train performance $\bar{r}_i$ for four runs. The instrument was Crude in the case of buying. The four runs vary in performance but reaches a stable convergence. Meaning not a large observed variance between the trainings.

6.2.3 Market Variables

The result of training with only private variables, with private and context variables and with all variables are presented in Table 6.9. The same hyperparameter setting as in Table D.2 was used. We can observe that adding the context variables did not reduce the transaction cost but using all the variables did.

Table 6.9: Test performance with different input variables. For NASDAQ and in the case of selling.

Variables	$\bar{C}_{avg}$
private	0.93
private and context	1.01
all	0.74

6.2.4 Policy Analysis PPO

When the policy learned by PPO was tested on the test data we saved all the states it observed and actions that it performed. Then the mean action for each time step in the episode was calculated. The result of this can be seen in Table 6.10. The observed pattern is that the policy orders a more aggressive price and more of the remaining volume as time passes in the episode. The behaviour is sound since it is expensive to have a large remaining volume at the end of the episode.

Table 6.10: Actions performed on the test data. When the trained model was evaluated on the test data, the states and actions were saved. The mean price $\bar{p}_\Delta$ and mean volume percentage $\bar{v}_\%$ for each time step ordered by the agent is presented. This result was for NASDAQ in the case of buying.

t	$\bar{p}_\Delta$	$\bar{v}_\%$
0	-0.12	12%
1	-0.11	17%
2	-0.09	22%
3	-0.07	27%
4	-0.05	32%
5	-0.03	38%
6	-0.02	43%
7	-0.01	48%
8	-0.01	52%

6.3 Cost and Standard Deviation for all Models

In Table 6.11 we present the test results for all models and baseline strategies on test data. The standard deviations of the costs are presented in Table 6.12. The setting shown for both these tables are $H = 32$, $T = 8$, $pr = 0.05$, $I = 4$ and we boldfaced numbers to highlight what strategy that performed best for each particular part and instrument. We can observe that being able to control the volume of the order reduces the transaction cost since Part II has lower costs than Part I. We can also observe the trade-off between low costs and high variance for ED, and high costs and low variance for IE. Another result is that PPO performs best for all instruments compared to the other models.

Table 6.11: Cost table $\bar{C}_{avg}$ for Part I and Part II.

Strategy \Instrument	**Crude**	**FTSE**	**Gilt**	**NASDAQ**
	Part I			
IE	8.81	4.07	2.83	8.39
SL	2.68	2.16	1.24	3.61
CP	1.82	**1.95**	**0.98**	3.04
NFK	2.48	2.32	1.16	3.45
Dual NFK	2.45	2.33	1.19	3.44
Sarsa(λ)	**1.76**	2.11	**0.98**	**3.03**
	Part II			
CPWV	0.97	0.90	0.54	1.41
ED	1.32	1.09	0.78	1.92
PPO	**0.73**	**0.77**	**0.32**	**1.31**

Table 6.12: Standard deviation $\bar{C}^{std}_{avg}$ for Part I and Part II.

Strategy\Instrument	**Crude**	**FTSE**	**Gilt**	**NASDAQ**
	Part I			
IE	**4.82**	**2.64**	**1.24**	**2.87**
SL	5.37	5.92	3.01	10.98
CP	5.04	3.52	2.99	7.63
NFK	12.86	12.84	6.92	17.08
Dual NFK	12.79	12.85	6.97	17.10
Sarsa(λ)	8.83	8.38	3.06	14.16
	Part II			
CPWV	**8.21**	**6.40**	**4.71**	**11.80**
ED	10.30	9.15	5.62	16.73
PPO	9.01	9.38	4.85	17.14

Chapter 7
Discussion

In this chapter we first discuss our method of market simulation and offer some critique and justifications on this matter. We then discuss other ways of performing the hyperparameter search for Sarsa(λ) and PPO. Finally, the results are discussed with regards to the research questions.

7.1 Method

This section will discuss our method with a strong focus on the market simulation. This is since it is a building block for all our results achieved and thus vital to the thesis.

7.1.1 Market Simulation

Our matching simulator builds upon three main assumptions:

- We assume no hidden liquidity
- We assume that the order book recovers from our impact
- We assume no behaviour impact of our orders

We will in this section discuss these assumptions.

Hidden Liquidity

As described in Section 2.3, all the markets we simulate contain hidden liquidity in reality. This could partly be modelled by assuming a ratio of hidden volume to the visible volume, and let an artificial order match with the hidden volume as well, as is done in [8]. However, we are not able to assess this ratio based on the data we have. Thus, due to the scope of the thesis, we assume that there is no hidden liquidity.

This assumption affects how transferable our results are to real markets. In reality, the cost of placing aggressive orders would in general result in a lower cost than in our simulated environment, since we would match with more volume at better prices as we dig in the opposing order book. Similarly, the queue might be longer than shown when passive orders are placed, which mean that passive orders might have a lower probability of being filled in reality than in our simulated environment. Our simulated environment will thus be biased and reward passive orders and punish aggressive orders compared to how the real markets would reward such orders.

In reality, the cost of the baseline strategies IE and ED would probably be lower (since they place aggressive orders). Furthermore, SL, CP and CPWV might have learned more aggressive actions during the grid search if hidden liquidity was taken into account. Lastly, our models would probably all learn more aggressive actions in general. However, since all models and strategies are exposed to this same weakness, our hypothesis is that the relative difference would be similar.

Recovery of Order Book

During the matching process we do some modifications in the order book to gain a realistic process. However, at the next order placement time point we assume that a limit order book without any modifications is valid (i.e. we use the historical limit order book for that time point). This can partly be motivated if the market is resilient. There have been several studies of the resilience of a limit order market. Resilience refers to a market's ability to absorb and recover from very large trade shocks. Griffiths, Smith, Turnbull, and White [41] found that only the most aggressive orders lead to a significant unfavourable price impact when studying the Toronto Stock Exchange . They also found that passive orders have a very small market impact, sometimes even negative (given that trades were executed).

A study of the Paris Bourse, a pure limit order stock market, found that after about 50 trades the order book replenished after even the most aggressive orders [42]. A small analysis of the markets we are trading in confirms that there are indeed more than 50 trades every minute, which partly justifies that our impact can be assumed to have dissipated the next minute. With the same argument, Hendricks and Wilcox [10] argue that permanent market impact can be assumed to be zero when addressing the trade execution problem empirically for the South African stock market.

Behavioural Impact

When placing orders in the market, it is likely that we affect the order placement behaviour of other market participants. However, estimating and modelling this behaviour is problematic. In reality it is not possible to measure the price impact with and without one's activity. In other words, one can never know how prices would evolve with or without an order, and the two scenarios can thus not be compared. Therefore, it is impossible to measure in real life and difficult to model in simulations. Of course, one could incorporate some strategic behaviour by other market participants into the order book, but this adds a layer of complexity that is hard to justify. For this reason, we choose to ignore the behavioural price impact, as do other papers which have chosen to model the price movement empirically [9, 10].

7.1.2 Hyperparameters for Sarsa(λ)

Due to the time constraints of the thesis project, the hyperparameter tuning of Sarsa(λ) was limited in scope. For example, we chose to only vary one parameter at the time (assume that the parameters would independently affect the result) and that the tests could be done for one instrument and problem setting. A more exhaustive search (perhaps using an optimization method) would probably have resulted in better performance of Sarsa(λ).

7.1.3 Hyperparameters for PPO

The hyperparameter search for PPO could have been performed in a different way. We discuss the hyperparameter search in Appendix D but one important aspect is that better results could have been achieved with more time and hardware resources. We evaluated 48 different hyperparameter settings, where each setting was limited to 24 hours of training. In a more ideal setting, more settings would have been tested and an average of several runs for each setting would have been used. This would likely to have improved the result.

7.1.4 Data and Experimental Settings

We have used a few instruments and one and a half year as data for our experiments. For a more robust measure, several years and more instruments could have been used. One could also have used time series cross validation to achieve more robust results for the costs. This could reduce the effect that financial data is non stationary. However, the time of convergence

for our methods limited us from doing this within the time scope of the thesis. It is also not certain that more historical data would have benefited our models. Another choice that affected our results is that we tested many experimental settings for Part I but only one for Part II. More general conclusions would have been possible if we had used several settings for Part II. But again, the time and hardware limitations we had were limiting factors.

7.2 Results

In this section we will discuss our results achieved to be able to answer our research questions.

7.2.1 Results and Shortcomings of Baselines and NFK

Baselines

IE has overall the highest cost, which is not surprising since it at the first time step places a market order for all the volume. With a large volume, this will result in worse prices since the order book depth has a limited liquidity. Furthermore, SL is the second worst baseline strategy. The transaction cost is lower than IE but higher than CP, especially for higher participation rates. Nevmyvaka, Feng, and Kearns [9] argue that SL reduces costly monitoring, but that can more easily be done by computers today, so this argument is no longer justified. With a low horizon resolution $H = 8$ and a low participation rate $pr = 1\%$ there is not a big difference between SL and CP. But CP has the lowest cost of all the baselines in Part I. CP is better than SL over all instruments making it a robust baseline when one only controls the price of the order.

The best baseline for Part II is CPWV, and when compared to CP, it reduces the transaction cost by roughly 50% for each instrument. The other baseline strategy of Part II, ED, is still better than the models for Part I. This is probably since it can control the volume of the order.

NFK

Nevmyvaka, Feng, and Kearns [9] results show that NFK reduced transaction costs compared to SL. It is not stated whether their results hold across the buy and sell case. The results presented in Table 6.1 shows that NFK has a lower cost but not for all settings or instruments. For FTSE, NFK has a higher cost in 10 out of 12 settings compared to SL. But for instrument Crude Oil, NFK has a better or similar cost to SL for all settings. This could be because these markets have a different liquidity.

The policy and value function learned by NFK shows a similar pattern to the patterns learned in Nevmyvaka, Feng, and Kearns [9]. The policy for one setting is presented in Figure 6.4. The pattern observed is that the policy takes more aggressive orders with more inventory and less time left. The value function $v(s)$ in Figure 6.5 shows that the estimated cost of being in a state with more volume and less time remaining is costly. These patterns are sound since at the end of the episode, the policy must sell/buy all remaining volume which generally is costly.

NFK has a higher cost than CP meaning that NFK does not find a better policy than CP. As seen in Figure 6.2 and 6.4 the policy is passive with negative actions for most states. This can be compared to constant policy which uses a fixed action, usually 0 or 1.

Another result that Nevmyvaka, Feng, and Kearns [9] presents is that the increase in T and I improve performance can be compared with Figure 6.1. Increasing T results in a lower cost but not increasing I. Another finding is that the market variables reduce the cost for NFK as presented in Table 6.6. This result leads us to investigate more market variables and models that can efficiently use them such as PPO.

7.2.2 Training with the Dual Case

Since our models were exposed to price movement during the training phase we formulated a hypothesis that the models might overfit to this price movement (and thus perform worse if the trend was different in the testing phase). Say for example that the price of a specific instrument is moving upwards and we are training a model which tries to buy this instrument. Since we want to buy as cheap as possible, the model might learn to buy as fast as possible (before it becomes too expensive), something that probably would not be optimal when the price would go down, perhaps in a test phase. To test this hypothesis, we investigated whether a model could be regularized by training a dual model, that was exposed to both the sell and the buy case when training, which led to Dual NFK. To isolate the dual factor, we choose to not evaluate the model with market variables.

The result was that Dual NFK performance was on par with NFK. We can conclude that the trick of exposing the model to both the sell and buy case did not help, but it did not hurt neither. Of course, training one model rather than two and achieve the same result out of sample is preferable. This does also motivate the training process of Sarsa(λ), which was exposed to both the buy and sell case during training.

One hypothesis of why the dual trick did not improve the performance significantly was that this price movement did not affect the model in the first place. A small analysis of the data does indeed confirm that an average price movement exists during the training period. However, this movement might be insignificant compared to the volatility the model is exposed to during the period.

7.2.3 Sarsa(λ)

As the Appendix C show, the simple state representation $s = (t, i)$ is not Markovian in our environment. Recall that NFK exploits the Markov property fully by doing a one-step look-ahead when updating its cost table. We implemented Sarsa(λ) with the hypothesis that a large λ (less bootstrapping) could lead to better performance. Another issue with NFK was that it tried all actions for all states in all data points, thus encountered many outliers in the reward function which affects the cost table significantly.

The issue of bootstrapping turned out to not be a large issue, in fact, a large λ only led to slow convergence since it has higher variance, most likely since the variance of the update is larger with a large lambda, as theory suggests [18]. However, we believe that the fact that Sarsa(λ) was not exposed to outliers in the data led to a less passive behavior than NFK (see Section 6.1.9), which also performed better on test data.

When comparing the models, we are not isolating the difference between the algorithms since they have different reward functions. It is therefore hard to draw conclusions on what contributes most to the performance of Sarsa(λ).

7.2.4 PPO

PPO is the best performing model we evaluated. We did not expect a DRL technique which could have stability issues during training to perform that well in a trading problem where there is a large uncertainty in both the reward and the state transition. However, PPO has achieved good results on Atari games and simulated control tasks before [13]. These good results transfer to our problem, optimized trade execution, which has different characteristics than the games and control tasks. We believe this is because the loss function can limit the policy update, that PPO can use a continuous and multidimensional action space and that one can easily adjust the amount of data to be used for the policy update.

Performance

PPO has the lowest transaction cost in all four instruments of all models in the tested problem setting. This result shows that DRL can be applied on the optimized trade execution problem and indeed perform very well. It is worth noting that the standard deviation is higher than the second best baseline, CPWV. Reducing the standard deviations has not been the focus of this report, we rather focused solely on the cost. One could for example incorporate the standard deviation in the reward function if one would want to try to reduce it.

For the best result, on instrument Gilt, PPO reduces transaction cost with 41% compared to the best baseline CPWV. The main contribution to this result is that PPO can select a different action for each time step, something CPWV cannot.

Hyper Parameters and Training Stability

PPO gave good performance for several different hyperparameters but not for all. A tuning of the parameters is therefore a necessity. How much better the performance would have been with further tuning calls for more research (with more time and hardware resources).

The training algorithm proved stable for multiple runs as seen in Figure 6.12. If the training algorithm would have been unstable, the possibility to get stuck in a policy which does not generate data that can be learned from could have happened, but it did not. The most important parameters for stable updates are ξ, M, K and U. The loss function limits the probability ratio to be large or small with ξ, even though we found the best setting with a larger ξ, it still limits the policy update. The amount of data used for an update is controlled with M, K and U. Where we found that collecting a large amount of data U using a large batch size for gradient ascent M gives good results.

The idea that the same hyperparameters would work for all instruments and both in the case of buying and selling worked since PPO gave good results on all settings with the same parameters. This confirms previous literature, that one can tuned the parameters on one task and use them on similar ones. The fact that PPO is robust to different parameters makes it simpler to tune as described in the paper by Shulman et. al [13]. This also contributed to the good results since DRL algorithms are generally difficult to train.

State Variables

Since PPO uses a neural network as function approximator for its policy, the possibility to analyse exactly which state variables that improves the performance is difficult. This is because the neural network can find complex patterns in the input variables that are not easily investigated. Therefore, we performed the test with three categories of state variables, only private variables, private and context variables, and then with all variables (including predictive variables). The context variables did not improve the performance when compared to only using the private variables as seen in Table 6.9. But when using all variables, the transaction cost decreased. It is difficult to determine if the context variables are needed because they might increase performance in combination with the predictive variables. More research regarding which variables are of importance is necessary.

7.3 Wider Context

The applications of our work would be to use a RL algorithm for trade execution at a large investment institution. Then the ethical question arises, presented by Russel and Norvig [43], whether this intelligent system will replace human jobs. But instead of replacing a human job, using a RL algorithm would most likely mean replacing an existing algorithm with another. So, from an ethical viewpoint, this model would not automate jobs performed by humans.

Another important question is how intelligent systems for trading might affect the accountability of the performance. If a DRL method is used for executing orders rather than

lesser complex models, the predictability of the behaviour might be reduced. A problematic situation would arise if the model performs poorly in a new situation or learns to manipulate the market. Who would be accountable? Would it be the researcher who suggested the model or the supervisor who approved the model? We mention this question since this situation may arise, but since ethical questions are not within the scope of this thesis, we leave the question unanswered. Naturally, this risk could be reduced by taking certain measures, such as forbidding the models to take extreme actions when employed. Yet again, taking such security measures are out of the scope for this thesis.

Lastly, a common apprehension in media, is that as these methods become more complex and advanced, they run the risk of becoming autonomous and start performing tasks outside their original domain. We will not speculate on whether this is likely or not. However, with a limited state and action space as well as a specific goal to minimize transaction cost, we find it hard to believe that these models would do anything else than what they are trained to do. In the worst case, they will only do it poorly.

Chapter 8
Conclusion

We have evaluated NFK and found that it performed on par to the results achieved by Nevmyvaka, Feng and Kearns relative to SL in terms of transaction cost for a few problem settings [9]. However, for most evaluated problem settings, SL performs better. Of course, it is important to note that our results are generated using other data (different asset from a different time period) and a different matching simulator.

Additionally, we introduce CP, a quite simple baseline strategy, that consistently performs better than SL and NFK in terms of cost. CP is using the same action space and order update frequency as NFK (T), performs better than SL while still being relatively simple to train and implement. We thus argue that it qualifies as a better baseline strategy for comparison than SL.

Our findings suggest that the overfit to the price movement was not significant, since dual NFK performed on par with NFK in terms of cost in the test phase. However, there are other advantages of training a dual model, the most obvious being that one only needs to train one model instead of two. However, a dual model can only be trained when the problem is completely symmetric in the buy and sell case, which it often is not when market variables are introduced.

The fact that Dual NFK performed on par with NFK did however motivate us to develop Sarsa(λ) as a dual model. We found that Sarsa(λ) with a modified reward function and ϵ-greedy policy performed better than NFK across almost all instruments and problem settings. It also performed better than CP for some instruments and problem settings, something NFK did not.

It is worth noting that ED and CPWV performed better than all the above mentioned strategies and models. However, the comparison is not fair since they can also regulate the volume when placing orders. A better comparison is done with PPO.

We can conclude that PPO reduces transaction cost when compared to our baseline strategies. PPO gives robust results on both buying and selling for all instruments with the same hyperparameters. The training stability was sufficient to give consistent results on multiple runs. More research needs to be done to see if these results extend to longer horizons.

Our findings show that by introducing information regarding the market, one can further reduce transaction cost. This also confirms the result of previous research [9]. Both in the case of using a tabular RL method and when using a function approximation for the policy, the transaction cost is reduced when using market variables. However, when using a neural network to approximate the policy, more research needs to be performed regarding which variables help to reduce the transaction cost.

We can conclude that we have reached the aim of this thesis, namely to implement and evaluate a set of RL algorithms that aims to reduce the cost of selling or buying financial instruments compared to a set of chosen baseline strategies. Furthermore, we argue that we have achieved compelling results that show promise for DRL to solve the optimized trade execution.

8.1 Future Research

All though not the main focus of our research, it is interesting to research how to model a limit order market in a more realistic manner. Both with the data we had available as well as more granular data. A more realistic model, incorporating for example behaviour impact and hidden liquidity, would obviously generate results that are more transferable to real markets.

We have shown that market variables can indeed improve the performance for this problem, and it would thus be interesting to further research which market variables that could be used to achieve even better results when applying a DRL method for optimized order execution. Fixing all else equal, doing an extensive exploratory study which market variables improve performance, would be of great interest for both researchers and practitioners.

Furthermore, it would be interesting to apply a value based DRL method instead of a policy gradient method. All though limited in some cases (doesn't allow for continuous action spaces), value based DRL could incorporate other features, such as action-specific states when estimating the value of a particular action in a particular state. This could be done with a network estimating the action-value of $q_\theta(s, a)$ with the state, action and action-specific state as inputs. If all actions (and their corresponding action-specific states) was fed through the network, one would simply choose the action with the highest action-value $q_\theta(s, a)$. Examples of action-specific states could be an estimated fill probability for that particular action or number of lots in the book for that particular price level that the action corresponds to.

When considering a policy gradient method for this problem, it is naturally interesting to further evaluate more sets of hyperparameters and architectures. Obviously, a more rigorous and extensive hyperparameter search could generate new findings for this problem. Additionally, by using an input of several historical states into the network or by incorporating a recurrent neural network (such as a LSTM-network) instead of feed forward neural network one would allow the network to find relations over time.

Appendix A
Matching Simulation

In this appendix we explain in detail how we choose to simulate matching artificial orders generated by our models with historical depths and trades.

A.1 Immediate matching

The immediate matching algorithm refers to the matching that can be done directly with active limit orders in the order book depth. Recall our notation for an order $x = (\tau_x, p_x, \omega_x)$, where τ_x is the order type (0 for sell, 1 for buy), p_x is the price of the order and ω_x is the volume of the order. Let x be an artificial order inserted at time t into the LOB $\mathcal{L}_t \in \{y_t^1, y_t^2, ..., y_t^n\}$, where $y_t^i = (\tau_{y_t^i}, p_{y_t^i}, \omega_{y_t^i})$, $i = 1, ..., n$, are the aggregated historical limit orders in the limit order book. By aggregated we mean that the volume are summed over each particular price level.

Function `immediate_matching`$(\tau_x, p_x, \omega_x, \mathcal{L}_t)$**:**
- $v_{im}, cash_{im}, \bar{p}_{im}, v_{existing} \leftarrow 0$
- **if** $\tau_x = 0$ **then**
  - $y_t^{bid} \leftarrow \{y_t^{i*} | i* = \text{argmax}_{i \in 1:n} \{p_{y_t^i} | y_t^i \in \mathcal{L}_t, \tau_{y_t^i} = 1\}\}$
  - $bid_t \leftarrow p_{y_t^{bid}}$
  - **while** $bid_t \geqslant p_x$ ***and*** $\omega_x > 0$ ***and*** $\mathcal{L}_t \neq \varnothing$ **do**
    - $bidvol_t \leftarrow \omega_{y_t^{bid}}$
    - `// Match`
    - v_{im} += $\min\{bidvol_t, \omega_x\}$
    - $cash_{im}$ += $\min\{bidvol_t, \omega_x\} \cdot bid_t$
    - ω_x -= $\min\{bidvol_t, \omega_x\}$
    - `// Match complete`
    - $\mathcal{L}_t \leftarrow \mathcal{L}_t \backslash \{y_t^{bid}\}$
    - **if** $\{y_t^i \in \mathcal{L}_t | \tau_{y_t^i} = 1\} = \varnothing$ **then**
      - **if** $\omega_x > 0$ **then**
        - Match with linearly extrapolated volume
        - Exit loop
      - **else**
        - Exit loop
      - **end**
    - **else**
      - $y_t^{bid} \leftarrow \{y_t^{i*} | i* = \text{argmax}_{i \in 1:n} \{p_{y_t^i} | y_t^i \in \mathcal{L}_t, \tau_{y_t^i} = 1\}\}$
      - $bid_t \leftarrow p_{y_t^{bid}}$
    - **end**
  - **end**
- **end**

(*continue from previous page*)

if $\tau_x = 1$ **then**
  $y_t^{ask} \leftarrow \{y_t^{i*} | i* = \text{argmin}_{i \in 1:n}\{p_{y_t^i} | y_t^i \in \mathcal{L}_t, \tau_{y_t^i} = 0\}\}$
  $ask_t \leftarrow p_{y_t^{ask}}$
  while $ask_t \leqslant p_x$ ***and*** $\omega_x > 0$ ***and*** $\mathcal{L}_t \neq \varnothing$ **do**
    $askvol_t \leftarrow \omega_{y_t^{ask}}$
    `// Match`
    v_{im} += $\min\{askvol_t, \omega_x\}$
    $cash_{im}$ += $\min\{askvol_t, \omega_x\} \cdot ask_t$
    ω_x -= $\min\{askvol_t, \omega_x\}$
    `// Match complete`
    $\mathcal{L}_t \leftarrow \mathcal{L}_t \backslash \{y_t^{ask}\}$
    if $\{y_t^i \in \mathcal{L}_t | \tau_{y_t^i} = 0\} = \varnothing$ **then**
      if $\omega_x > 0$ **then**
        Match with linearly extrapolated volume
        Exit loop
      else
        Exit loop
      end
    else
      $y_t^{ask} \leftarrow \{y_t^{i*} | i* = \text{argmin}_{i \in 1:n}\{p_{y_t^i} | y_t^i \in \mathcal{L}_t, \tau_{y_t^i} = 0\}\}$
      $ask_t \leftarrow p_{y_t^{ask}}$
    end
  end
end
if *no orders were matched* **then**
  $v_{existing} \leftarrow \{\omega_{y_t^i} | \{y_t^i \in \mathcal{L}_t | \tau_{y_t^i} = \tau_x, p_{y_t^i} = p_x\}\}$
  if $v_{existing} = \varnothing$ **then**
    $v_{existing} \leftarrow 0$
  end
else
  $\bar{p}_{im} \leftarrow cash_{im} / v_{im}$
end
return $\bar{p}_{im}, v_{im}, v_{existing}$

If an order empties the opposing order book, still has volume left to execute, it is assumed that the remaining volume can be linearly extrapolated based on the depth of the book before the matching took place. In other words, we assume that the volume for the price levels missing will be an average of the ones we have.
This linear extrapolation can be justified by two reasons; (1) a small analysis of the data confirms that each price step in the order book on average has the same volume, hence justifying a linear extrapolation, and (2) it is generally expensive to dig this far into the order book, so no agent will learn to do so. See Appendix B.1 for more details regarding the volume in the order book depth.

If no orders were matched, we register $v_{existing}$, i.e. the volume of the order in the book that is of the same type τ_x and price p_x as the artificial order x. This volume is used in the continuous matching algorithm.

Figure A.1: An example of linear extrapolation of the volume of the order book.

A.2 Continuous matching

During the continuous matching, we let the remaining quantity $\omega_x^{remaining}$ of the artificial order match with historical trades that took place during the succeeding minutes (depending on the problem setting). We define the historical trades as the set $\mathcal{T}_t$ at time t. However, in the continuous matching, the priority of the orders must be considered. If limit orders of the same price and side existed in the books, the simulation will first execute their total volume $v_{existing}$ before the artificial order can match its volume during the continuous matching. Likewise, the quantity v_{im} is also removed from the matches of the continuous trade, since those trades already took place during the immediate matching in the simulated scenario, this is to avoid the scenario of matching twice with the same historical order. Let $x = (\tau_x, p_x, \omega_x^{remaining})$ be the artificial order after the immediate matching (with remaining $\omega_x^{remaining}$ corresponding to the remaining volume to be executed) inserted at time t into $\mathcal{T}_t = \{z_t^1, z_t^2, ..., z_t^n\}$ where $z_t^i = (p_{z_t^i}, \omega_{z_t^i})$.

Function `continuous_matching` $(\tau_x, p_x, \omega_x^{remaining}, \mathcal{T}_t, v_{existing}, v_{im})$**:**
- $v_{cont} \leftarrow 0$
- `// Demand` $v_{existing}$ `more lots from continuous trade`
- $\omega_x^{remaining}$ += $v_{existing}$
- **if** $\tau_x = 0$ **then**
  - $z_t^{max} \leftarrow \{z_t^{i*} | i* = \text{argmax}_{i \in 1:n}\{p_{z_t^i} | z_t^i \in \mathcal{T}_t\}\}$
  - $p_{max} \leftarrow p_{z_t^{max}}$
  - **while** $p_{max} \geqslant p_x$ ***and*** $\omega_x^{remaining} > 0$ **do**
    - $v_{p^{max}} \leftarrow \omega_{z_t^{max}}$
    - `// Match below`
    - v_{cont} += $\min\{v_{p^{max}}, \omega_x^{remaining}\}$
    - $\omega_x^{remaining}$ -= $\min\{v_{p^{max}}, \omega_x^{remaining}\}$
    - `// Match complete`
    - $\mathcal{T}_t \leftarrow \mathcal{T}_t \backslash \{z_t^{max}\}$
    - $z_t^{max} \leftarrow \{z_t^{i*} | i* = \text{argmax}_{i \in 1:n}\{p_{z_t^i} | z_t^i \in \mathcal{T}_t\}\}$
    - $p_{max} \leftarrow p_{z_t^{max}}$
  - **end**
- **end**

(*continue from previous page*)

if $\tau_x = 1$ **then**
  $z_t^{min} \leftarrow \{z_t^{i*} | i* = \text{argmin}_{i \in 1:n}\{p_{z_t^i} | z_t^i \in \mathcal{T}_t\}\}$
  $p_{min} \leftarrow p_{z_t^{min}}$
  while $p_{min} \leqslant p_x$ ***and*** $\omega_x^{remaining} > 0$ **do**
    $v_{p^{min}} \leftarrow \omega_{z_t^{min}}$
    `// Match below`
    v_{cont} += $\min\{v_{p^{min}}, \omega_x^{remaining}\}$
    $\omega_x^{remaining}$ -= $\min\{v_{p^{min}}, \omega_x^{remaining}\}$
    `// Match complete`
    $\mathcal{T}_t \leftarrow \mathcal{T}_t \backslash \{z_t^{min}\}$
    $z_t^{min} \leftarrow \{z_t^{i*} | i* = \text{argmin}_{i \in 1:n}\{p_{z_t^i} | z_t^i \in \mathcal{T}_t\}\}$
    $p_{min} \leftarrow p_{z_t^{min}}$
  end
end
`// Remove the lots that was before us in the queue`
if $v_{existing}$ *was registered during the immediate match* **then**
  $v_{cont} \leftarrow \max\{0, v_{cont} - v_{existing}\}$
end
`// Remove the lots that we already matched with during immediate matching`
if *any volume* v_{im} *was matched during the immediate match* **then**
  $v_{cont} \leftarrow \max\{0, v_{cont} - v_{im}\}$
end
return v_{cont}

Note that before returning v_{cont} we modify it to gain a more realistic result. If any $v_{existing}$ (volume already in the books for the same type of order as x) was registered, it must be removed from v_{cont} since it in reality it would be traded before us (in front of us in the queue). The other modification is that we remove v_{im} since we already matched with those trades during the immediate match (and can thus not match with them again). The max-operator assures that we never trade a negative volume.

To conclude, the whole trade simulator at time step t can be expressed in this short algorithm:

$$\bar{p}_{im}, v_{existing}, v_{im} \leftarrow \text{immediate\_matching}(\tau_x, p_x, \omega_x, \mathcal{L}_t)$$

$$\omega_x^{remaining} \leftarrow \omega_x - v_{im}$$

$$v_{cont} \leftarrow \text{continuous\_matching}(\tau_x, p_x, \omega_x^{remaining}, \mathcal{T}_t, v_{existing}, v_{im})$$

Algorithm 8: Trade simulator($\tau_x, p_x, \omega_x, \mathcal{L}_t, \mathcal{T}_t$).

The immediate and continuous matching result in a cash flow at time t given a order with price p_x as seen in equation A.1. The immediate price $\bar{p}_{im}$ represents the average price of all orders that were matched during immediate matching.

$$cash_t = \bar{p}_{imim} + p_{xcont} \tag{A.1}$$

The total volume achieved from the order is

$$v_t = v_{im} + v_{cont} \tag{A.2}$$

Appendix B
Exploratory Data Analysis

The data was investigated through an exploratory data analysis (EDA). This was done to understand results and visualize certain aspects of the data. Investigations regarding for instance stationarity and outliers in the data was made.

B.1 Order Book Depths

We evaluated the depth profile of the order books and the average volume per depth was calculated. This was performed to evaluate possible extrapolations of the order book when placing a market order larger than the current depth using level 2 data. The average volume for each instrument at price level $p = 1, ..., 10$ is displayed in Figure B.1.

B.2 Participation Rates

Since a average participation rate is calculated using the average daily volume it is possible that certain episodes have a higher participation rate than others. This is due to the fact that the instruments are traded more and less frequently over time. In Figure B.2 the distribution of participation rates for each episode and instrument. In Figure B.3 the average participation rate for each timestamp intra day is displayed for each instrument.

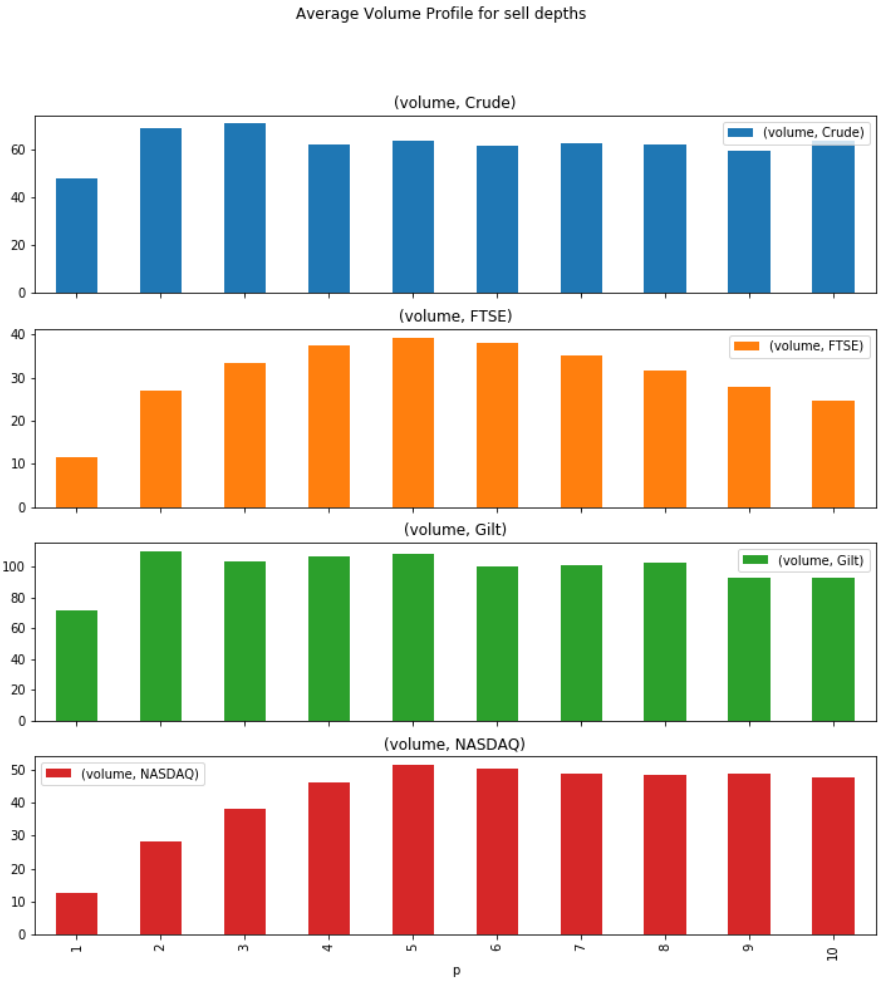

Figure B.1: Average volumes at different price levels p in the sell order book.

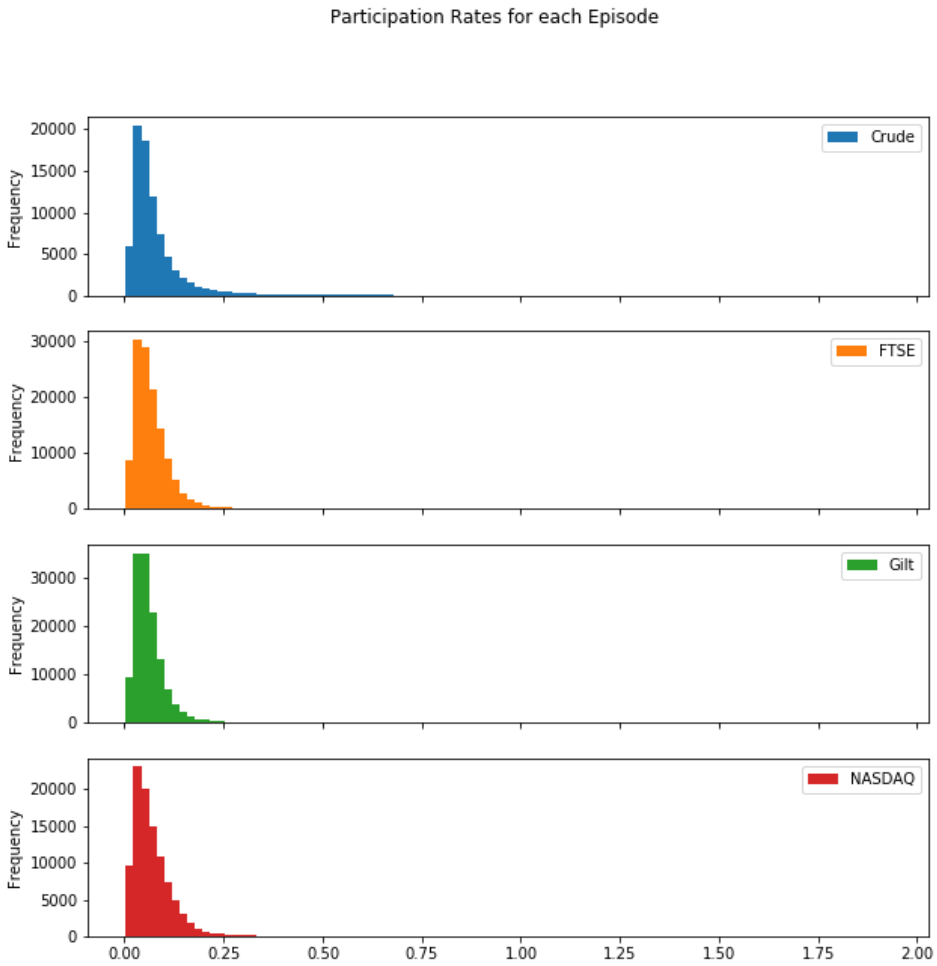

Figure B.2: Distribution of participation rates when using a volume (based on 5% participation rate) calculated on an average daily volume.

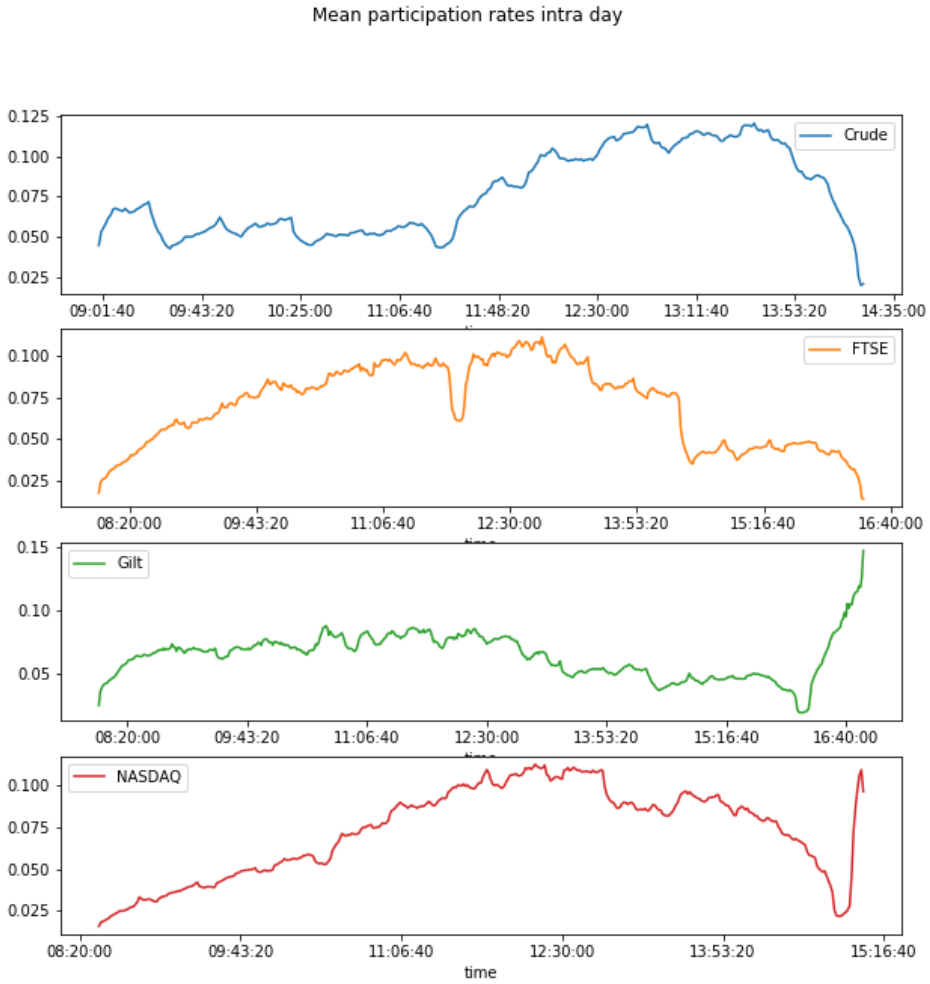

Figure B.3: The average participation rate for each timestamp and instrument given a volume calculated on the average daily volume.

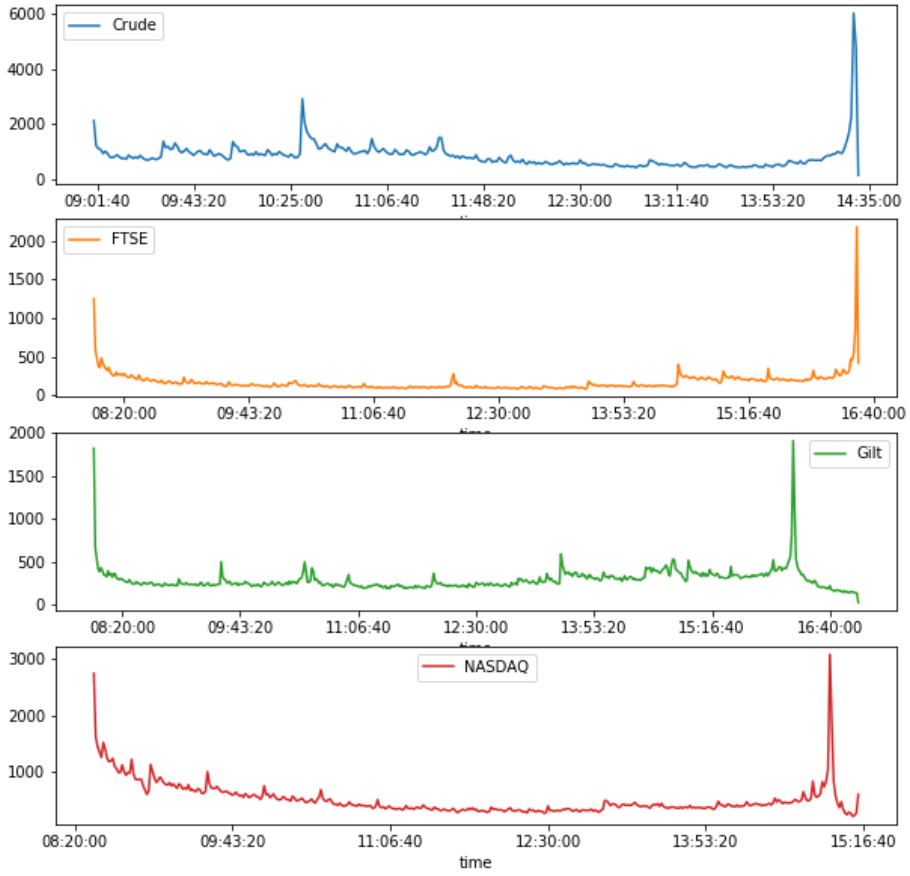

Figure B.4: Average intraday volume per timestamp and instrument.

Appendix C
Test of Markov Property

The cost update of NFK and Dual NFK

$$c(s,a) := \frac{n}{n+1}c(s,a) + \frac{1}{n+1}[c_t(s,a) + \min_{a'} c(s',a')]$$

exploits the Markov independence from one time step to the next by only doing a one step look-ahead. In the simplest state representation $s = (t,i)$, which we will denote i_t in this appendix, we can test this property. We do this by first training a NFK model with the setup

Table C.1: Configurations to test Markov property.

PR	0.1
H	8 min
I	9
T	8
a_{min}	-10
a_{max}	10

for Crude and then evaluate the model on the same data. During the evaluation we sample state transitions $i_t \rightarrow i_{t+1}$ given an action from a NFK model. According to the Markov property, the probability of the next inventory should be independent of the previous inventory.

$$\mathbb{P}(i_{t+1}|a_t, i_t) = \mathbb{P}(i_{t+1}|a_t, i_t, i_{t-1})$$

By sampling the frequencies of transitions one can test if the sample distributions differ depending on the previous inventory. This can done by doing a two-sample Kolmogorov-Smirnov test between the sample distribution for all states given all previous inventories and its corresponding sample probability independent of previous states. This resulted in a total of 324 test for both the buy and sell case. Since we are doing multiple tests, we correct them using False Discovery Rate (Benjamini-Hochberg procedure) [44]. The result is that we can reject the null-hypotheses that the transitions come from the same distributions with 95 % probability for 95 transitions in the buy case and 43 for the sell case. In Figure C.1 we show a state where the probability of the next inventory depends on 5 out of 6 previous different inventories when transitioning.

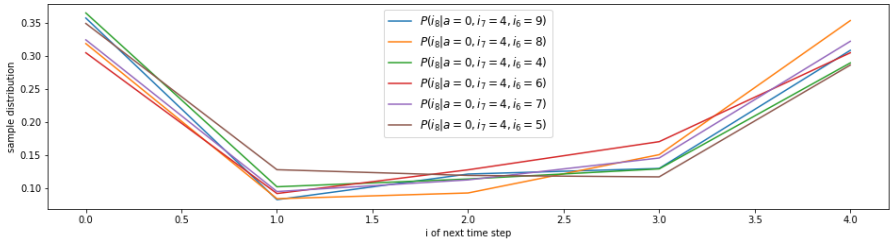

Figure C.1: Sample distributions of next inventory given previous inventory.

This failed Markov assumption motivates a higher-dimensional state-space and a non-bootstrapping model.

Appendix D
Hyperparameter Tuning

D.1 Initial Tuning

Initial tests where performed to evaluate some parameters and the range of others. The test was performed on 8 days for one instrument by evaluating $\bar{r}_i$ and $\bar{C}_{average}$ in sample. This was done since some variables did not provide a stable training or good performance in sample. The initial tuning resulted in the selection and range of variables as in Table D.1 .

Table D.1: Initial found hyperparameters and range for others.

Parameter	**Selected Value/Range**
λ	1
K	[4, 8]
α	0.0005, 0.00005
M	[10240, 20480]
k1	1
k2	0
Estimate std.	FALSE
Hidden Layers	[[64, 64], [128, 128]]
ξ	[0.1, 0.2, 0.3]
U	81920
IT	500
$f(z)$	tanh

D.2 Analysis Grid Search

The performance for different hyper parameters vary as seen in Table D.3. The best result is -0.130 and the worst result is -0.290. The top 4 results get similar performance. The common parameters for these results are that the clipping parameter ξ is 0.3. The size of the network, *hiddens*, the amount of times to perform gradient ascent per iteration, K, and the batch size, M, do not to change the performance much.

D.3 Hyperparameter Search Critique

If the time and hardware resources would not have been a constraint, another metric and time for each hyper parameter would have been chosen. Since the networks weights are randomly initialized, the policy samples random actions during the training, and gradient ascent uniformly samples data from the temporary memory $\mathcal{D}$, the result for a training is not deterministic. An approach is to run a hyper parameter setting multiple times and then average the result of these runs [13]. The time limitation we set was 24 hours per hyperparameter

which resulted on average in 200 episodes. If enough time would have been given, the hyper parameters would have been average over several runs and for at least 500 iterations.

D.4 Result Hyperparameters

The final selection of hyperparameters are presented in Table D.2.

Table D.2: Chosen hyperparameters.

Param	λ	K	α	M	k1	k2	Estimate std.	Hidden Layers	ξ	U	IT	$f(z)$
Value	1	8	0.0005	20480	1	0	FALSE	[128, 128]	0.3	81920	500	tanh

The results for the hyperparameter grid search will be presented in Table D.3.

Table D.3: Grid search hyperparameters.

hiddens	K	M	α	ξ	$\bar{h}$
128_128	8	20480	0.0005	0.3	-0.130
128_128	4	10240	0.00005	0.3	-0.131
64_64	8	10240	0.0005	0.3	-0.131
64_64	4	10240	0.0005	0.3	-0.132
64_64	4	20480	0.0005	0.3	-0.134
128_128	4	20480	0.0005	0.3	-0.139
64_64	8	10240	0.00005	0.3	-0.139
64_64	4	10240	0.00005	0.3	-0.144
128_128	8	10240	0.0005	0.3	-0.145
128_128	4	10240	0.0005	0.3	-0.146
64_64	8	20480	0.0005	0.2	-0.146
128_128	4	10240	0.00005	0.2	-0.148
64_64	8	20480	0.0005	0.3	-0.151
64_64	8	10240	0.00005	0.2	-0.151
64_64	4	10240	0.00005	0.2	-0.154
128_128	4	10240	0.0005	0.2	-0.163
64_64	4	10240	0.0005	0.2	-0.166
128_128	8	10240	0.0005	0.2	-0.169
128_128	8	20480	0.0005	0.2	-0.169
64_64	4	20480	0.0005	0.2	-0.173
64_64	8	10240	0.0005	0.2	-0.175
128_128	4	20480	0.0005	0.2	-0.181
128_128	8	20480	0.0005	0.1	-0.205
64_64	8	10240	0.0005	0.1	-0.217
64_64	4	10240	0.00005	0.1	-0.218
128_128	4	10240	0.00005	0.1	-0.219
64_64	8	20480	0.0005	0.1	-0.219
64_64	4	20480	0.0005	0.1	-0.222
64_64	4	10240	0.0005	0.1	-0.228
128_128	8	20480	0.00005	0.3	-0.235
128_128	4	10240	0.0005	0.1	-0.237
128_128	8	10240	0.00005	0.2	-0.241
128_128	4	20480	0.00005	0.3	-0.245
64_64	8	20480	0.00005	0.2	-0.245
128_128	8	10240	0.00005	0.3	-0.245
128_128	8	20480	0.00005	0.1	-0.25
128_128	4	20480	0.0005	0.1	-0.252
128_128	8	10240	0.0005	0.1	-0.254
64_64	8	20480	0.00005	0.3	-0.255
128_128	4	20480	0.00005	0.1	-0.257
128_128	8	20480	0.00005	0.2	-0.257
64_64	8	20480	0.00005	0.1	-0.263
64_64	8	10240	0.00005	0.1	-0.264
128_128	4	20480	0.00005	0.2	-0.273
128_128	8	10240	0.00005	0.1	-0.273
64_64	4	20480	0.00005	0.1	-0.282
64_64	4	20480	0.00005	0.3	-0.284
64_64	4	20480	0.00005	0.2	-0.290

Bibliography

[1] Thomas F Loeb. "Trading cost: the critical link between investment information and results". In: *Financial Analysts Journal* 39.3 (1983), pp. 39–44.

[2] Louis KC Chan and Josef Lakonishok. "The behavior of stock prices around institutional trades". In: *The Journal of Finance* 50.4 (1995), pp. 1147–1174.

[3] Louis KC Chan and Josef Lakonishok. "Institutional equity trading costs: NYSE versus Nasdaq". In: *The Journal of Finance* 52.2 (1997), pp. 713–735.

[4] Donald B Keim and Ananth Madhavan. "Transactions costs and investment style: an inter-exchange analysis of institutional equity trades". In: *Journal of Financial Economics* 46.3 (1997), pp. 265–292.

[5] Andre F Perold. "The implementation shortfall: Paper versus reality". In: *The Journal of Portfolio Management* 14.3 (1988), pp. 4–9.

[6] Robert Almgren and Neil Chriss. "Optimal execution of portfolio transactions". In: *Journal of Risk* 3 (2001), pp. 5–40.

[7] Dimitris Bertsimas and Andrew W Lo. "Optimal control of execution costs". In: *Journal of Financial Markets* 1.1 (1998), pp. 1–50.

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

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

[10] Dieter Hendricks and Diane Wilcox. "A reinforcement learning extension to the Almgren-Chriss framework for optimal trade execution". In: *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2104 IEEE Conference on*. IEEE. 2014, pp. 457–464.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[12] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[14] Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. "Limit order books". In: *Quantitative Finance* 13.11 (2013), pp. 1709–1742.

[15] David Silver. *Reinforcement Learning*. [Lecture notes, online; `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html` (accessed 6-January-2018)]. 2015.

[16] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[17] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

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

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

[20] John Moody and Matthew Saffell. "Learning to trade via direct reinforcement". In: *IEEE transactions on neural Networks* 12.4 (2001), pp. 875–889.

[21] John Moody, Lizhong Wu, Yuansong Liao, and Matthew Saffell. "Performance functions and reinforcement learning for trading systems and portfolios". In: *Journal of Forecasting* 17.56 (1998), pp. 441–470.

[22] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[23] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[24] Ioannis Kanellopoulos and Graeme G. Wilkinson. "Strategies and best practice for neural network image classification". In: *International Journal of Remote Sensing* 18.4 (1997), pp. 711–725.

[25] Nazri Mohd Nawi, Walid Hasen Atomi, and MZ Rehman. "The effect of data preprocessing on optimized training of artificial neural networks". In: *Procedia Technology* 11 (2013), pp. 32–39.

[26] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[27] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Reinforcement Learning*. Springer, 1992, pp. 5–32.

[28] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample efficient actor-critic with experience replay". In: *arXiv preprint arXiv:1611.01224* (2016).

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 2. 1. available online at http://incompleteideas.net/book/bookdraft2018jan1.pdf. MIT press Cambridge, 2018.

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[31] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *Advances in neural information processing systems*. 2017, pp. 5285–5294.

[32] A Verdugo Lazo and P Rathie. "On the entropy of continuous probability distributions (corresp.)" In: *IEEE Transactions on Information Theory* 24.1 (1978), pp. 120–122.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[34] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[35] Erik Pärlstrand. "Comparing fast- and slow-acting features for short-term price predictions". MA thesis. KTH Royal Institute of Technology, 2017.

[36] Rama Cont, Arseniy Kukanov, and Sasha Stoikov. "The price impact of order book events". In: *Journal of financial econometrics* 12.1 (2014), pp. 47–88.

[37] Steven B Achelis. *Technical Analysis from A to Z*. McGraw Hill New York, 2001.

[38] John K Wald and Holly T Horrigan. "Optimal limit order choice". In: *The Journal of Business* 78.2 (2005), pp. 597–620.

[39] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I Jordan, and Ion Stoica. "Ray: A Distributed Framework for Emerging AI Applications". In: *arXiv preprint arXiv:1712.05889* (2017).

[40] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. Austin, Texas: ACM, 2015, 7:1–7:6. ISBN: 978-1-4503-4005-2. DOI: 10.1145/2833157.2833162. URL: http://doi.acm.org/10.1145/2833157.2833162.

[41] Mark D Griffiths, Brian F Smith, D Alasdair S Turnbull, and Robert W White. "The costs and determinants of order aggressiveness". In: *Journal of Financial Economics* 56.1 (2000), pp. 65–88.

[42] Hans Degryse, Frank De Jong, Maarten Van Ravenswaaij, and Gunther Wuyts. "Aggressive orders and the resiliency of a limit order market". In: *Review of Finance* 9.2 (2005), pp. 201–242.

[43] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[44] Yoav Benjamini and Yosef Hochberg. "Controlling the false discovery rate: a practical and powerful approach to multiple testing". In: *Journal of the royal statistical society. Series B (Methodological)* (1995), pp. 289–300.