Imperial College London

Imperial College London

Department of Mathematics

Reinforcement Learning Trading Strategies with Limit Orders and High Frequency Signals

Author: Arvid Bertermann (CID: 01912841)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2020-2021

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I would like to thank my supervisor Paul Bilokon for the time and effort he invested into supporting my thesis by providing feedback and invaluable advice over several months. Additionally, I would like to thank David Finkelstein and DQR Ltd for giving me access to an industry grade backtester.

Lastly, I would like to thank my friend Hendrik for motivating me throughout the project.

Abstract

When trading financial markets with limit orders at high frequency, multiple automated decisions have to be made within seconds to exploit suitable data patterns. Theoretical approaches to place limit orders in an optimal fashion oversimplify the market environment and hence fail to account for a number of properties that limit order markets possess.

Temporal Difference Learning and Deep Q Learning provide algorithms that are able to learn optimal behaviour by interacting with their environment. The interaction can be based on historical data, such that real experience drives the learning progress of the agent. Both sets of methods have previously been successfully applied in conjunction with limit orders in market making and optimal execution.

Assuming that a combination of different signals describes the state of the limit order book as well as the market order trading flow and price momentum, Temporal Difference Learning and Deep Q Learning may be able to transform the information into a profitable high frequency trading strategy based on limit orders.

In the following work we will give a theoretical overview of tabular methods and Deep Q Learning. Moreover, we focus on implementing numerous agents aiming at comparing their performance based on simulated signals and historical limit order book data.

Contents

List of Figures

List of Tables

Introduction

Due to the progress of technology and the immense amount of trades being initiated, a vast number of financial assets are traded electronically. This means that multiple (limit) orders can be submitted within a very short time frame. As humans are not able to learn and make decisions within nanoseconds based on higher dimensional input in a reliable way, automated systems can be leveraged to use the possible speed of financial markets and to invest profitably. One approach would be to formulate a mathematical problem, which aims at replicating the behaviour of the financial asset as a continuous stochastic process. This setup would then be further used to define an optimisation problem, which aspires to maximise the profit of a mathematically defined strategy. This has been done in [7] for numerous setups including market making and optimal execution. These models make strong assumptions and may oversimplify the complex environment of financial markets.

Contrary to stochastic control, (Deep) Reinforcement Learning is usually *model free*. The idea behind (Deep) Reinforcement Learning is to learn from experience by interacting with the environment. The agent receives information (state), that can be compared to features in a Linear Regression or Machine Learning problem. Unlike in Supervised Learning, the agent does not approximate a target variable but tries to maximise a numerical feedback that most likely varies based on the current state and the action that the agent takes. After making a decision the agent observes a new state, selects another action and receives another reward and a next state ... Following this scheme, the agent gradually explores its environment and adjusts its behaviour based on the details of the learning algorithm.

The application of (Deep) Reinforcement Learning to various trading problems has already been extensively researched. The paper by Kolm and Ritter [8] addresses the problem of how to optimally hedge a portfolio of option derivatives. In the field of optimal execution research by Ning, Lin and Jaimungal [9] and Schnaubelt [2] has been published. In detail, [9] demonstrates how to use a variant of Deep Q Learning to optimally liquidate large orders. The agent makes use of a fully connected neural network and Experience Replay. The paper by Schnaubelt focuses on the optimal placement of cryptocurrency limit orders. According to [2] proximal policy optimisation in combination with queue imbalances as part of the environment outperform several benchmarks. Moreover, two papers by Spooner et al. [10],[11] connect Reinforcement Learning Agents to the subject of market making. The findings in [10] mention that on-policy methods, in particular SARSA, and averaging instead of a learning rate perform better and more stably. One of the most recent papers from Cartea et al. [15] employs Double Deep Q Networks to trade in a Foreign Exchange setup. More precisely, the agents interact with the limit order book of three currency pairs (FX triplet), where the exchange rate of one pair is redundant due to arbitrage arguments.

In Chapter 1 we will discuss the foundations of Reinforcement Learning starting with Markov Decision Processes and Dynamic Programming. This is followed by an introduction to the actual learning methods that the trading agents use. In detail, we address the workings and properties of tabular agents such as Monte Carlo Methods and Temporal Difference Learning. Additionally, we present Eligibility Traces which combine the two different concepts into an updating mechanism that smoothly reduces the impact of future rewards on recently visited state–action pairs. Finally, we briefly mention how neural networks are used in Deep Reinforcement Learning to approximate Q-values and hence derive an optimal policy. The next section will be dedicated to describing the architecture, optimisation and training of feedforward neural networks. Lastly, we shed some light on the market microstructure of limit order markets by explaining the two basic types of orders

and how the limit order book facilitates trade.

The main goal of Chapter 2 is to gain a first understanding of how Q Learning and Deep Q Learning are able to derive an optimal policy based on a simulated mean-reverting signal. The signal is generated using an Ornstein–Uhlenbeck stochastic differential equation. We start with a simple setting and extend it to account for additional actions and states in a more complicated environment. In particular, we will pay attention to the convergence, aggressiveness and stability of the agent's performance.

Chapter 3 focuses on the application of tabular Reinforcement Learning agents to historical limit order book data. First, we theoretically design the agent and explain the intricacies of the backtesting environment. In order to use the valuable results from the second chapter, we emphasise mean-reverting signals. The intuition behind these signals is divided into four concepts: price momentum, trading momentum from market orders, trading momentum from limit orders and risk. The target is to find a combination of such features that enables one or more of the agents with sufficient information to profitably invest in a gradually realistic scenario. We do so by combining own ideas with promising research findings by Schaubelt [2] and Spooner et al. [10].

In the final section of the thesis we conclude our results and provide an outlook of what we think could be promising next steps to even further sharpen the performance and stability of our agents.

Chapter 1

Theory

1.1 Reinforcement Learning

We start with a famous quote from George Box:

> All models are wrong, but some are useful.

Especially in mathematics, theoretical models aim at approximating real-world environments and the interaction with them. Mostly, approaches to replicate processes with stochastic differential equations or complex functions do not account for all features of the true environment, which makes the model unrealistic and incorrect. Reinforcement Learning does not try to mirror the behaviour of processes. More precisely, it is mainly based on the idea of evaluating an agent's decision making within its environment and the agent's impact on its environment. In general, given some information (state) the agent gradually learns to choose the optimal decision (action) by maximising a numerical signal (reward). This signal may be designed in a way such that it focuses on short term or long term success of the agent's decision making. The exact setup of actions, states and a reward function is very flexible and contributes to the versatile and powerful application of Reinforcement Learning. Recent successes of this flexible set of learning methods are the AlphaGo agent, which beat the European Champion at the game of Go in 2015 [5] and the agent that learned to master different Atari games [4]. Furthermore, the most current paper by DeepMind combines tree-based methods with a learned model. Schrittwieser et al. [17] claim that their agents achieve superhuman performance in Atari, Go, Chess and Shogi without knowing the underlying dynamics of the complex domains. The theoretical introduction to Reinforcement Learning closely follows [1].

1.1.1 Markov Decision Processes

We start with Markov Decision Processes (MDP), as they model decision making in a discrete time setting and in a stochastic environment. MDPs are able to choose different actions in various scenarios (states) and then receive evaluative feedback (reward). Hence, the actions space $\mathcal{A}$, the state space $\mathcal{S}$ and the reward function $\mathcal{R}$ are the three main pillars of a MDP algorithm. We deal with finite MDPs, which means that the state space, the action space and the reward space are finite. The learning process is distinguished by a loop of state, action, reward, next state and next action. This sequence is often referred to as $SARSA$ or $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$.

How this theoretically endless sequence is translated into optimal decision making depends on the design of approximating either the value function $v(s), s \in \mathcal{S}$ or state–action values $q(s,a), s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. In very general terms, the value function $v(s)$ represents the (discounted) expected future return of a state, whereas the state–action value $q(s,a)$ identifies the (discounted) expected future return of a state when selecting action a. MDPs are the foundation of so-called tabular methods in Reinforcement Learning as they map values to states or state–action pairs that ultimately control the agent's decision making.

When considering finite MDPs, $\mathcal{R}$ and $\mathcal{S}$ can be interpreted as random variables. Their discrete probabilities merely depend on the action and the preceding state denoted by:

$$p(s', r|s, a) := \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$. As p defines a probability measure the following equation has to hold:

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

The probability function p completely characterises the dynamics of the MDP. We assume the Markov property, meaning that the state s contains all the information that can make a difference in the future [1, Chapter 3.1, Pages 47-49].

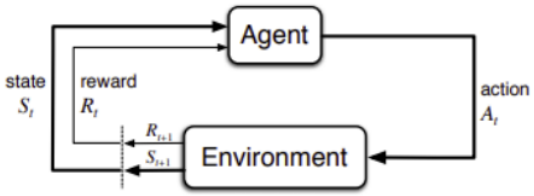

Figure 1.1: The agent–environment interaction in a Markov Decision Process as found in [1, Chapter 3.1, Figure 3.1].

1.1.2 Expected return, value functions and policies

At some point we have to precisely define the value of visiting a state $v(s)$ or even selecting an action in a certain state ($q(s, a)$). We start by defining the expected return G_t after time t. The agent should be flexible enough to emphasise short term as well as long term behaviour. The introduction of a discount rate γ guarantees that:

$$G_t := R_{t+1} + \gamma G_{t+1} = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

with $0 \leq \gamma \leq 1$ and R_{t+1} representing the numerical signal (reward) received after action A_t. If we deal with an episodic task T, which is potentially a random variable, determines the end of each episode. If we have a continuous task at hand T is replaced by ∞. The discount rate γ represents the impact that future rewards have on the expected return at time t. If $\gamma = 0$ the agent learns based on immediate reward, where as if $\gamma = 1$ future rewards heavily impact the agents behaviour [1, Chapter 3.3, Pages 54-55].

As previously mentioned, the value function (state–action function respectively) identifies how beneficial it is for an agent to be in a state (and choosing an action resp.). In detail, it is defined as the expected return starting from state s (and choosing an action resp.). The expected return G_t depends on future rewards $R_k, t \leq k \leq T$, which are feedback-values of selecting actions in different scenarios. Hence, the value function is influenced by the underlying probability distributions that define how likely it is for the agent to choose an action in a given state. This mapping of probabilities from state to actions is called a policy $\pi(a|s), s \in \mathcal{S}, a \in \mathcal{A}(s)$.

The value function starting in state s and following the policy π is defined as follows:

$$v_\pi(s) := \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} | S_t = s \right], \text{ for all } s \in \mathcal{S}.$$

If the agent reaches a state that can't be left or the state always ends the current episode, it is called a terminal state. The value of a terminal state is zero. Similarly, we can define the state–action function under a policy π:

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

One of the most prominent features of the value/state–action function is its recursive nature which leads us to the Bellman equation [1, Chapter 3.5, Pages 58-59]:

$$v_\pi(s) := \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{1.1.1}$$

and

$$q_\pi(s,a) := \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] = \sum_{s',r} p(s', r|s, a)r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s', a'). \tag{1.1.2}$$

The main goal of each agent is to find the optimal policy. An optimal policy achieves greater or equal expected return than all other policies in every state $s \in \mathcal{S}$.

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \tag{1.1.3}$$

If equation (1.1.3) holds for all policies π', we have the optimal policy π_*. There could be several optimal policies which all share the same optimal state–value and optimal action–value function:

$$\begin{aligned} v_*(s) &:= \max_\pi v_\pi(s), \\ q_*(s,a) &:= \max_\pi q_\pi(s,a) \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s). \end{aligned} \tag{1.1.4}$$

The equations in (1.1.4) can be rearranged to the so called Bellman optimality equations:

$$v_*(s) := \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

$$q_*(s,a) := \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')].$$

1.1.3 Dynamic Programming

Dynamic Programming (DP) assumes a perfect model of the environment as a finite Markov Decision Process including model dynamics $p(s', r|s, a)$. Hence, DP provides an important theoretical foundation for Reinforcement Learning, but is mostly not applicable due to its unrealistic assumption of a known environment and its great computational costs. DP focuses on the search for good policies by using value functions. As a first step we focus on policy evaluation, which computes v_π. π is an arbitrary policy. If we recall the Bellman equations ((1.1.2), (1.1.1)), we can see that given a completely known model, the value function is a linear equation system with $|\mathcal{S}|$ equations. Due to the normally high number of states, iterative policy evaluation is preferred [1, Chapter 4.1, Pages 74-76]:

$$\begin{aligned} v_{k+1} &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]. \end{aligned}$$

It can be shown that v_k converges to v_π as $k \to \infty$.

Apart from deriving good estimates for the value function, we would also like to optimise our policy. One way of doing that is to change to a deterministic action $a \neq \pi(s)$ in a certain state s with $a \in \mathcal{A}(s)$ and observe if $q_\pi(s,a) \geq v_\pi(s)$. If so, it is better to always select the action a deterministically than sticking to the policy π. This is proven by the *policy improvement theorem*:

Theorem 1.1.1. *Let π, π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$, $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. Then π' must be as good as or better than π. This also means that $v_{\pi'}(s) \geq v_\pi(s)$.*

Now, we would like to change the policy π at all states such that $q_\pi(s,a)$ is maximised:

$$\begin{aligned}\pi'(s) :&= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')].\end{aligned}$$

This strategy is called *greedy* and selects the actions that look to be the best ones. Making a policy greedy is called policy improvement, which is in line with the policy improvement theorem.

As a next step we can combine policy evaluation and policy improvement. First we evaluate a starting policy π_0 to get our first estimate of the value function v_{π_0}. Then we make our strategy greedy (π_1) and use π_1 to derive the second estimate of the value function (v_{π_1}). This alternating process of evaluation and improvement is called *policy iteration* [1, Chapter 4.3, Pages 80-81].

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Figure 1.2: The E stands for policy evaluation/prediction and I stands for policy improvement. This graphic can be found in [1, Chapter 4.3, Page 80].

Algorithm 1: Policy iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialisation
2. $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
3. $\Delta = \infty$
4. Set θ to a small positive number determining the accuracy of estimation
5. 2. Policy Evaluation
6. **while** $\Delta > \theta$ **do**
7. $\quad \Delta = 0$
8. $\quad$ **for** *each* $s \in \mathcal{S}$ **do**
9. $\quad\quad$ v = V(s)
10. $\quad\quad$ V(s) = $\sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
11. $\quad\quad$ $\Delta = \max(\Delta, |v - V(s)|)$
12. $\quad$ **end**
13. **end**
14. 3. Policy Improvement
15. policy-stable = True
16. **for** *each* $s \in \mathcal{S}$ **do**
17. $\quad$ old-action = $\pi(s)$
18. $\quad$ $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
19. $\quad$ **if** *old-action* $\neq \pi(s)$ **then**
20. $\quad\quad$ policy-stable =False;
21. $\quad$ **end**
22. **end**
23. **if** *policy-stable* **then**
24. $\quad$ STOP
25. $\quad$ Return $V \approx v_*$, $\pi \approx \pi_*$;
26. **end**
27. **if** *not policy-stable* **then**
28. $\quad$ Go to 2.
29. **end**

1.1.4 Monte Carlo Methods

So far we have introduced a theoretical foundation for Reinforcement Learning. This chapter presents the first actual learning technique. Before that, let us discuss the difference between on- and off-policy. An on-policy method only works with one policy that is responsible for sampling and learning. The policy that makes decisions is also the one that has to be improved. Contrary to that, off-policy methods separate evaluation and decision making. One policy explores the environment and makes decisions, whereas another one is optimised based on the experience coming from the first policy.

In contrast to MDPs and Dynamic Programming we do not assume complete knowledge of the model environment. In principle, Monte Carlo methods sample sequences of $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ and use this experience to form value estimates by averaging sample returns for each state or state–action pair. Value functions are updated only after an episode has terminated. The termination of an episode can either be determined by visiting certain states or by the number of time steps since the last update. Again, we start with the prediction (evaluation). Obviously, it is possible that a state s is visited multiple times following a policy π within one episode. The *first-visit MC* only takes the first visits per episode of a state into account, whereas the *every-visit MC* applies the averaging of returns to all states that have been visited. Both methods converge to $v_\pi(s)$ for all $s \in \mathcal{S}$ [1, Chapter 5 and 5.1, Pages 91-92].

Algorithm 2: MC prediction, for estimating $V \approx v_\pi$

```
Input: a policy π to be evaluated
Initialize:
V(s) ∈ ℝ, for all s ∈ 𝒮
Returns(s) = empty list for all s ∈ 𝒮
Decide if first-visit=True or first-visit=False for every visit MC
while True do
    Generate an episode following π : S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T
    G=0
    for each step of episode, t=T-1, T-2, ..., 0 do
        G = R_{t+1} + γG
        if S_t not in S_0, S_1, ..., S_{t-1} or not first visit then
            Append G to Returns(S_t)
            V(S_t) = Average(Returns(S_t))
        end
    end
end
```

In practice one would implement the updating rule in a different way. To store all the returns in a state specific vector is memory-wise unnecessary and inefficient. A different way to do this would be by online calculation:

$$V_{n+1}(s) = \frac{nV_n(s) + G}{n+1}$$

with $V_k(s)$ the estimated value function for state s using k experienced returns and G the current return.

The same procedure is also applicable to state–action values. Especially when we do not have knowledge of the model, the approximation of state–action values is useful in order to suggest a policy. One problem of estimating state–action values is that many state–action pairs may never be visited. This issue can be attacked by letting the agent explore and not act greedy all the time when sampling $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$. In episodal tasks *exploring starts* help the agent to visit all state–action pairs multiple times by assigning non-zero probabilities to all state–action pairs of being chosen to start an episode. In addition, when sampling the *SARSA* sequence, the agent could follow an ϵ-greedy strategy, which chooses the greedy action with probability $1 - \epsilon$ and a random action with probability ϵ:

$$\pi(s) := \begin{cases} \arg\max_{a \in \mathcal{A}(s)} q(s,a), \text{ with probability } 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} \\ a \neq \arg\max_{a \in \mathcal{A}(s)} q(s,a), \text{ with probability } \frac{\epsilon}{|\mathcal{A}(s)|}. \end{cases}$$

The so-called MC control works conceptually exactly as the policy iteration. Its an alternating process of first visit/every visit MC prediction and making the policy greedy with respect to the current value function [1, Chapter, 5.3, Pages 97-99].

When it comes to updating values there are two main concepts that may be followed. The first one, as already introduced, is averaging. The second one is using an error term and a learning rate α. The error term represents the difference between the new and the old estimate.This can also be applied to Monte Carlo:

$$V_{n+1}(s) = V_n(s) + \alpha[G - V_n(s)]$$

with n denoting how many updates have already been done and G the current expected return.

1.1.5 Temporal Difference Learning

Similar to Monte Carlo, Temporal Difference (TD) methods learn from experience. Again, for this to be successful we do not need to have knowledge of our environment. In addition, TD uses features of Dynamic Programming, as it makes use of bootstrapping: TD updates current estimates based on other learned values and does not wait for the end of an episode to do so. As in previous chapters, we first concentrate on predicting the value function. The one-step TD updating rule (TD(0)) looks one step ahead into the future and uses a learning rate to adjust the current estimate:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]. \tag{1.1.5}$$

It has been proven that for a fixed policy π the TD(0) method converges to the value function under the policy v_π.

We have seen before that when the model is unknown it is very useful to approximate state–action values as they contain more information about improving the policy. There are several TD(0) updating rules regarding Q-values. Most of them closely follow (1.1.5). One example would be

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{1.1.6}$$

In episodic tasks all terminal states s have $q(s,a) = 0$. Formula (1.1.6) uses five values: S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}. Hence, it is called *SARSA* and is an on-policy TD Control method. It uses only one policy that gradually develops thorough state–action values. Policy improvement is accomplished by making π greedy with respect to the learned Q-values [1, Chapter 6.1 and 6.4].

Algorithm 3: SARSA (on-policy TD control) for estimating Q $\approx q_*$

```
Algorithm Parameters: step size α ∈ (0, 1], small ε > 0
Initialize Q(s, a), for all terminal states s⁺ ∈ S⁺, a ∈ A(s), Q(s⁺, a) = 0
for each episode do
    Initialize S
    Choose A from S using policy derived from Q (e.g. ε-greedy)
    for each step of the episode do
        Take action A, observe R, S′
        Choose A′ from S′ using policy derived from Q (e.g., ε-greedy)
        Q(S, A) = Q(S, A) + α[R + γQ(S′, A′) − Q(S, A)]
        S = S′
        A = A′
    end
end
```

The most common off-policy TD Control method, that is used in later stages of the thesis is Q Learning [1, Chapter 6.5, Pages 131-132].

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{1.1.7}$$

The approximated Q-values are independent of the policy that generates the data. However, the use of the maximum sometimes develops too optimistic state–action values. It can be shown that Q Learning converges to q_* with probability 1. The algorithm is the same as the SARSA one except from replacing line 9 with the new updating rule shown in equation (1.1.7). Instead of using the learning rate approach one could apply averaging of values. For SARSA:

$$Q_{n+1}(S_t, A_t) = \frac{nQ_n(S_t, A_t) + R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})}{n+1}. \tag{1.1.8}$$

1.1.6 TD(λ) — Eligibility Traces

So far we have seen two main concepts of how an agent can learn its state or state–action values. *Eligibility Traces* (ET) combine those two methods, which may lead to more efficient learning. The ET-parameter $\lambda \in [0, 1]$ decides if the learning method is closer to Monte Carlo ($\lambda = 1$) or TD ($\lambda = 0$). An Eligibility Trace E_t can be interpreted as a vector that tracks a value for each state or state–action pair. For TD(λ) that value starts at 0 for all $s \in \mathcal{S}$ and potentially $a \in \mathcal{A}(s)$. $E_t(s) = E_{t-1}(s) + 1$ ($E_t(s, a) = E_{t-1}(s, a) + 1$) if the agent is in the state s at time t (and chooses action a respectively). After each discrete time step, the whole Eligibility Trace is decayed by the discount rate γ and λ : $E_{t+1} = E_t \lambda \gamma$, meaning the value is high if the state (state–action pair) was recently visited and decays exponentially if the state (state–action pair) has not been visited. The TD error $\delta_t = R_{t+1} + \gamma W(S_{t+1}) - W(S_t)$ impacts all states (state–action pairs) in the following way

$$W_{t+1} = W_t + \alpha E_t \delta_t$$

with W representing a vector of state or state–action values. Thus, the current error is projected backwards in time and modifies especially those state or state–action values which have recently been visited.

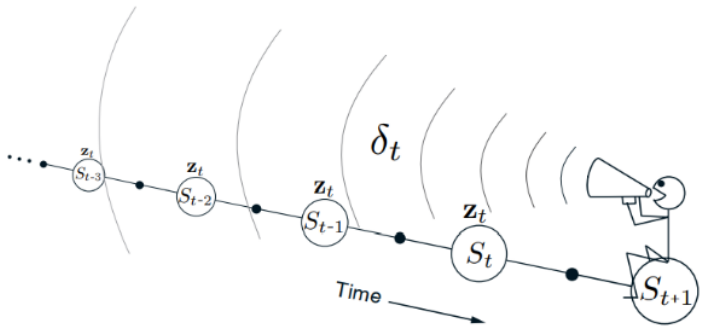

Figure 1.3: The backward view of TD(λ) as found in [1, Chapter 12.2, Figure 12.5]. Each update depends on the current δ. However, the Eligibility Trace (z_t in the picture) of past events decides how large the impact of the current TD error is on the previously visited states.

This procedure is computationally efficient as only one value per state or state–action pair has to be saved and the update at time t does not involve waiting for future rewards. This also reduces

the complexity of the algorithm [1, Chapter 12.2, Pages 292-294].

Algorithm 4: SARSA (on-policy TD(λ) Eligibility Trace) for estimating Q $\approx q_*$

Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
$E(s,a) = 0, s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $\alpha, \gamma \in [0,1], \lambda \in [0,1]$ **for** *each episode* **do**
    Initialize S
    Choose A from S using policy derived from Q (e.g. ϵ-greedy)
    for *each step of the episode* **do**
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $\delta = R + \gamma Q(S', A') - Q(S, A)$
        $E(S, A) = E(S, A) + 1$
        $Q = Q + \alpha \delta E$
        $E = E\gamma\lambda$
        $S = S'$
        $A = A'$
    end
end

To change the algorithm to the Q Leaning Off policy TD(λ) the error δ would be defined as the term in square brackets in equation (1.1.7).

1.1.7 Deep Q Learning

Until this point we have only considered tabular methods as states or state–action pairs were mapped to a value. This also means we indirectly assumed a discrete and finite state space $\mathcal{S}$ and action space $\mathcal{A}(s)$. What if we want to include continuous features in the environment? The infinite number of possible states prevents us from using tabular methods: First we could not implement a table that contains infinite values. In addition, tabular Reinforcement Learning relies on the fact that states are visited multiple times and then form a rough estimate based on multiple updates. With a continuous state space $\mathcal{S}$ that would hardly be possible.

The idea of Deep Q Learning is to use a neural network to predict state–action values. The input to the network is the state s, usually consisting of continuous variables. Output is a vector of state–action values (one for each action) that could have been chosen given the input state s. Before we focus on the learning itself we introduce the concept of *Experience Replay*. At each time step t the agent's experience consisting of $e_t = (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ is appended to a replay buffer M of limited size. The replay buffer is normally a queue such that it follows the "First In First Out" or "FIFO" principle. After a pre-defined number of time steps a minibatch is randomly sampled from the replay buffer M. This can be considered training data for the neural network. As the experience sequence is likely to be correlated, random sampling reduces the risk of overfitting. The last question to be solved is the determination of our target vector Y_t. All actions that are not selected are assumed to be correctly predicted. The state–action value belonging to the chosen action A_t is modified in the following way:

$$Y(A_t) := \begin{cases} R_t, \text{ for terminal } S_t \\ R_t + \gamma \max_a Q(S_{t+1}, a, \theta_t), \text{ for non-terminal } S_t \end{cases} \tag{1.1.9}$$

with $Q(S_{t+1}, a, \theta_t)$ representing the predicted state–action value of action a by the network taking the state S_{t+1} as input. The parameter θ_t identifies the weights of the network at time t. After the determination of the target vector Y_t the network is fitted to the new observation using stochastic

gradient descent. This section is based on [20, Chapter 3, Pages 54-86].

Algorithm 5: Deep Q Learning with Experience replay after each episode for estimating $Q \approx q_*$

Initialize replay memory M with capacity N
Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for *each episode* **do**
    Initialize S
    Choose A from S using policy derived from Q (e.g. ϵ-greedy)
    for *each step of the episode* **do**
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        Store transition (S, A, R, S', A') in replay buffer M
        $S = S'$
        $A = A'$
    end
    Sample random minibatch of transitions (S, A, R, S', A') from M
    Determine target vector Y_k for all transitions using equation (1.1.9)
    Fit network to data via gradient descent using backpropagation
end

The formula to calculate Y_t could also be adjusted to the SARSA updating rule.

1.2 Feedforward Neural Networks

Neural networks are at the heart of Deep Learning algorithms, which are a subset of Machine Learning techniques. In this chapter we will briefly introduce the structure of *feedforward neural networks* (FNN) and emphasize how the training process works. The definitions and algorithms are based on [6] and [13].

A neural network takes an I ($I \in \mathbb{N}$) dimensional input and leverages complex optimisation techniques to approximate an O ($O \in \mathbb{N}$) dimensional output. A FNN consists of an input layer, hidden layers and an output layer, which each consist of neurons (or units). In a fully connected FNN all neurons from the n-th layer are only connected to each neuron of the $n+1$-th layer. A connection means that data multiplied by a weight $w, w \in \mathbb{R}$, is sent from one neuron to the other. At each neuron itself the activation function modifies the data as well. There exists one activation function per layer, but different layers may apply different activation functions. The weights (resp. biases) belonging to each connection (resp. neuron) are represented via a weight matrix W (resp. bias vector b). Formally we define a *feedforward neural network* in the following way:

Definition 1.2.1. Let $I, O, r \in \mathbb{N}$. A function $\mathbf{f}$: $\mathbb{R}^I \to \mathbb{R}^O$ is a feedforward neural network (FNN) with $r-1 \in 0, 1, \ldots$ hidden layers, where there are $d_i \in \mathbb{N}$ units in the i-th hidden layer for any $i = 1 \ldots, r-1$, and activation functions σ_i: $\mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$, for any $i = 1, \ldots, r$ where $d_r = O$, if

$$\mathbf{f} = \sigma_r \circ L_r \circ \cdots \circ \sigma_1 \circ L_1,$$

where L_i: $\mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$, for any $i = 1, \ldots, r$ is an affine function

$$L_i = W^i x + b^i, x \in \mathbb{R}^{d_{i-1}},$$

parameterised by weight matrix $W^i = [W^i_{j,k}]_{j=1,\ldots,d_i, k=1,\ldots,d_{i-1}}$ and bias vector $b^i = (b^i_1, \ldots, b^i_{d_i}) \in \mathbb{R}^{d_i}$, with $d_0 = I$.

FNNs are applicable to regression and classification problems. Similar to a Linear Regression, neural networks minimise a loss function, which measures the distance of the predicted values to the true values. In contrast to a Linear Regression and due to the complex structure of networks, there does not exist a closed-form solution of matrix multiplications.

Usually, the training of a neural network lasts multiple epochs. Each epoch e randomly samples disjoint minibatches B^e_i. These minibatches are used to calculate the stochastic gradients. First let

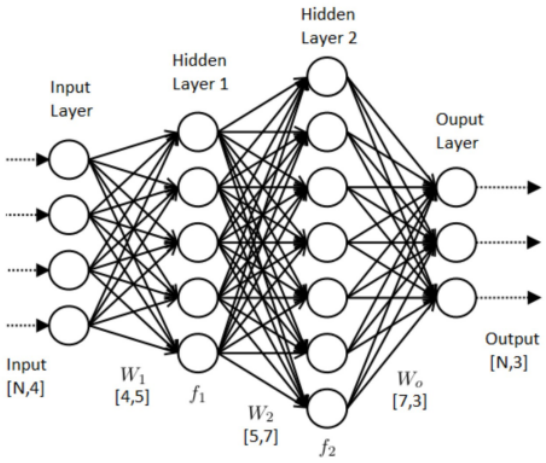

Figure 1.4: Graphical representation of a feedforward neural network with 2 hidden layers consisting of 5 and 7 units. The input is 4 dimensional whereas the output is 3 dimensional [14]. f_1 and f_2 denote the activation functions, N the number of observations the network is trained with and W the weight matrices.

us imagine that the loss function is subject to a very high-dimensional input. This input consisting of weights and biases represents the parameter set to be optimised such that the loss function is minimised. The search for a global minimum is conducted by the Stochastic Gradient Descent (SGD).

Algorithm 6: Stochastic Gradient Descent

Initialize minibatch size $m \in \mathbb{N}$ such that $N = km$ for some $k \in \mathbb{N}$
Initialize learning rate $\alpha > 0$, initial weights and biases θ_0 and number of epochs E
for $e = 1, \dots, E$ **do**
  Sample disjoint $B_1^e, \dots, B_k^e$ such that $|B_j^e| = m, \forall j = 1, \dots, k$
  if *e=1* **then**
    $\theta_0^e = \theta_0$
  end
  else
    $\theta_0^e = \theta_0^{e-1}$
  end
  for $i = 1, \dots, k$ **do**
    $\theta_i^e = \theta_{i-1}^e - \alpha \nabla_\theta \mathcal{L}_{B_i^e}(\theta_{i-1}^e)$
  end
end
return θ_k^E

with $\nabla_\theta \mathcal{L}_{B_i^e}(\theta_{i-1}^e)$ identifying the stochastic gradient of the loss function based on the i-th minibatch within the e-th episode. The predictions for B_i^e are calculated using the previous parameter set (θ_{i-1}^e). The stochastic gradient in each direction is derived via backpropagation. As the name indicates, this technique starts at the output layer of the network and sequentially calculates the stochastic gradient (details can be found in [6]).

1.3 Limit Order Markets

In this chapter we introduce limit order markets and explain how different types of orders can be used to buy and sell assets. As this section is based on Bouchaud et al. [21, Chapter 3, Pages 44-57] we can recommend this book for a more advanced theoretical study of market microstructure and limit order markets.

A limit order market's goal is to facilitate trade in an organised and secure way. The main features of such a market are the limit order book and the matching algorithm. The *limit order book (LOB)* is a discrete price grid. The distance between two prices is called *ticksize*. The relative ticksize (ticksize/mid-price) may indirectly impact the shape of a LOB. At each price, traders can submit a special type of order called limit order to fill up the LOB. There are two types of limit orders: bid and ask limit orders. A bid (ask) limit order corresponds to the obligation to buy (sell) an asset at a specified price and a specified quantity if the order is matched. A limit order is uniquely defined by four features: direction d, time t, price p and quantity q. A limit order can be interpreted as an offer to buy/sell and hence provides liquidity to the market. For this offer, traders receive a slightly better price as bid limit orders are below the mid-price and ask limit orders are above p_{Mid}. Hence, a trader submitting a limit order has the chance to buy lower and sell higher than p_{Mid}. The mid-price is calculated in the following way:

$$p_{Mid}(t) := \frac{a_0(t) + b_0(t)}{2} \tag{1.3.1}$$

with $a_0(t)$ being the best (lowest) ask and $b_0(t)$ the best (highest) bid price in the LOB at time t. A bid (ask) limit order with a price higher (lower) than or equal to the lowest ask (highest bid) is treated as a market order.

Normally, there is no guarantee that a limit order is going to be executed as the incoming order flow of market orders determines which orders in the LOB are going to be matched. A buy (sell) market order is immediately matched with ask (bid) limit orders in the queue of the lowest ask (highest bid) price. This means, that for the guarantee of execution a trader using market orders pays a small premium, as he/she sells below and buys above the mid-price. A market order can be identified by three features: direction d, time t and quantity q. In contrast to limit orders the price is automatically detected based on the shape of the limit order book.

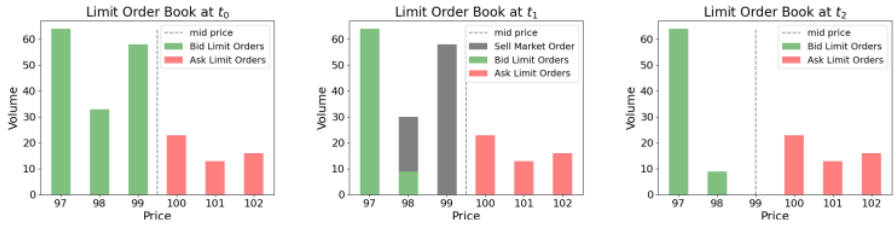

Figure 1.5: Graphical representation of a limit order book at three times. *Left:* Initial LOB; *Center:* Sell market order consuming liquidity; *Right:* LOB after executed trade.

Figure 1.5 illustrates the development of the limit order book following the price-time priority algorithm, which is the most common to be responsible for matching orders at an exchange. The algorithm automatically looks for the best price and matches limit orders according to the"First In First Out" (FIFO) principle. Limit orders that have been submitted earlier will be executed before limit orders that have been submitted at later times assuming they are quoted at the same price. This means orders are queueing up. Thus, the volume at any price in the LOB is also referred to as a queue.

The left graphic in Figure 1.5 represents the initial scenario (t_0). At t_1 a relatively large sell market order arrives at the exchange and consumes the whole liquidity at the best bid ($b_0(t_1) = 99$) and also about 70% of the volume at the second best bid price $b_1(t_1) = 98$. The ask side remains unchanged as no buy market orders or ask limit orders have been submitted or cancelled. At t_2 we can observe that the mid-price has moved down and liquidity has been consumed on the bid-side of the LOB. This example points out that buy market orders drive the mid-price up while sell market orders decrease p_{Mid}.

1.3.1 Stochastic Control and Market Making

In this section we briefly present the stochastic control setup for market making proposed in [9, Chaper 10], as it is the closest theoretical model to what we are trying to achieve in the *model free* Reinforcement Learning setting. A market maker quotes limit orders on both sides of the LOB and tries to maximise its profit by earning the spread $a_0(t) - b_0(t)$. The control of the market maker is the depth at which he quotes prices. We define the following processes:

- $S = \{S_t\}_{t\geq 0}$ denotes the mid-price, with $S_t = S_0 + \sigma W_t, \sigma > 0$ and $W = \{W_t\}_{t\geq 0}$ is a standard Brownian motion.
- $\delta^{+,-} = \{\delta_t^{+,-}\}_{t\geq 0}$ describes the distance to the mid-price S. Buy (sell) limit orders are posted at $S_t + \delta_t^+$ $(S_t - \delta_t^-)$.
- $M^{+,-} = \{M_t^{+,-}\}_{t\geq 0}$ is the Poisson process with rates $\lambda^{+,-}$ counting the arrival of other market participants buy (+) and sell (-) market orders.
- $N^{\delta,+,-} = \{N_t^{\delta,+,-}\}_{t\geq 0}$ denote the counting of filled sell (+) and buy (-) limit orders.
- Conditional on market order arrival, the posted limit order is filled with probability $e^{-\kappa^{+,-}\delta_t^{+,-}}$, with $\kappa^{+,-} \geq 0$.
- $X^\delta = \{X_t^\delta\}_{t\geq 0}$ defines the Market makers cash following :

$$dX^\delta = (S_{t^-} + \delta_t^+)dN_t^{\delta,+} - (S_{t^-} - \delta_t^-)dN_t^{\delta,-}.$$

- Finally, the agents inventory is the difference of the counting processes $Q^\delta = \{Q_t^\delta\}_{t\geq 0}$ with $Q_t^\delta = N_t^{\delta,-} - N_t^{\delta,+}$.

It should be mentioned that $N^{\delta,+,-}$ are not Poisson processes, as they do not always jump up when market orders $M_t^{+,-}$ increase. The fill rate of limit orders can be written as $\Lambda_t^{\delta,+,-} = \lambda_t^{+,-} e^{-\kappa^{+,-}\delta_t^{+,-}}$.

As mentioned before the market maker's strategy depends on $\delta^{+,-}$ and will be defined over an finite time horizon $[0, T]$. The terminal inventory Q_T^δ is liquidated with a worse price than the mid-price. In general the inventory process moves within $q_- \leq Q_t^\delta \leq q_+, q_- < 0 < q_+$. The objective it to maximise the following expression:

$$H^\delta(t, x, S, q) := \mathbb{E}_{t,x,q,S}\left[X_T + Q_T^\delta(S_T^\delta - \alpha Q_T^\delta) - \phi\int_t^T (Q_u^\delta)^2 du\right]$$

with $\phi, \alpha > 0$ constants. Hence, the value function is:

$$H(t, x, S, q) := \max_{\delta\in\mathcal{A}} H^\delta(t, x, S, q)$$

with $\mathcal{A}$ not denoting the discrete space of actions but the set of admissible strategies. Next, we present the Hamilton-Jacobi-Bellmann (HJB) equation. More information about the theoretical foundation of solving stochastic control problems can be found in [12].

$$\begin{aligned}
0 = {} & \partial_t H + \frac{1}{2}\sigma^2\partial_{SS}H - \phi q^2 \\
& + \lambda^+ \sup_{\delta^+}\{e^{-\kappa^+\delta_t^+}(H(t, x + (S + \delta^+), q - 1, S) - H(t, x, q, S))\}\mathbf{1}_{\{q>q_-\}} \\
& + \lambda^- \sup_{\delta^-}\{e^{-\kappa^-\delta_t^-}(H(t, x - (S - \delta^-), q + 1, S) - H(t, x, q, S))\}\mathbf{1}_{\{q<q_+\}}
\end{aligned}$$

corresponding to the terminal condition $H(T, x, S, q) = x + q(S - \alpha q)$. In order to simplify the HJB we define the ansatz

$$H(T, x, S, q) := x + qS + h(t, q)$$

which transforms the terminal condition into $h(T,q) = -\alpha q^2$. The HJB reduces to

$$\begin{aligned} 0 &= \partial_t h - \phi q^2 \\ &+ \lambda^+ \sup_{\delta^+} \{e^{-\kappa^+ \delta_t^+} (\delta^+ + h(t,q-1) - h(t,q)\} \mathbf{1}_{\{q > q_-\}} \\ &+ \lambda^- \sup_{\delta^-} \{e^{-\kappa^- \delta_t^-} (\delta^- + h(t,q+1) - h(t,q)\} \mathbf{1}_{\{q < q_+\}} \end{aligned}$$

It can be shown that the optimal depths result in

$$\delta^{+,*} = \frac{1}{\kappa^+} - h(t,q-1) + h(t,q), q \neq q_-$$

$$\delta^{-,*} = \frac{1}{\kappa^-} - h(t,q+1) + h(t,q), q \neq q_+.$$

The boundary conditions for the stochastic control δ are $\delta^{+,*}(t,q_+) = \delta^{-,*}(t,q_-) = \infty$. If we set $\phi = \alpha = 0, q_+ = |q_-| = \infty$ the optimal depth is given by $\delta^{+,-} = \frac{1}{\kappa^{+,-}}$. Hence, $-h(t,q+-1)+h(t,q)$ can be identified as the inventory cost. Moreover,with no inventory restrictions the function h solving the HJB can be shown to be

$$h(t) = e^{-1} \left(\frac{\lambda^+}{\kappa^+} + \frac{\lambda^-}{\kappa^-} \right) (T-t).$$

Hence, the market maker neglecting inventory boundaries and liquidating at the mid-price at T simply maximises the probability of filling his/her limit orders at every time step independent of time and inventory.

Chapter 2

Trading the Ornstein–Uhlenbeck Process

We start the application of Reinforcement Learning with agents trading on a mean-reverting process in a very simplified setting. The price or signal process is simulated with the following Ornstein–Uhlenbeck stochastic differential equation (SDE):

$$dX_t = \theta(\mu - X_t)dt + \sigma W_t \tag{2.0.1}$$

with W_t a Brownian Motion, the local standard deviation σ, the mean-reverting speed θ and the mean-reverting value μ around which the process X_t oscillates. The closed form of the SDE can be shown to be

$$X_t = X_0 e^{-\theta t} + \mu(1 - e^{-\theta t}) + \sigma \int_0^t e^{-\theta(t-s)} \mathrm{d}W_s. \tag{2.0.2}$$

From now on we interpret the simulated Ornstein–Uhlenbeck process as a stock price, which the agent can instantly buy or sell. As the process described by (2.0.1) fluctuates around the mean-reverting value μ, the ultimate goal of the agents would be to simply buy low and sell high. All simulated processes use the same parameter setup:

$$\mu = 0 \ , \ \theta = 10 \ , \ \sigma = 0.2$$

In order to simulate instances of the process we first divide the time interval $[0, 10]$ into a partition $t = [t_0, t_1, ..., t_{98}, t_{99}]$ with equally distanced time points and $\Delta = t_1 - t_0 = ... = t_{99} - t_{98}$. The following iterative equation based on (2.0.2) is used to determine a realisation of the Ornstein–Uhlenbeck process:

$$\begin{aligned} &X_0 = \mu \\ &X_{t_{i+1}} = X_{t_i} \cdot e^{-\theta\Delta} + \mu(1 - e^{-\theta\Delta}) + \tfrac{\sigma}{\sqrt{2\theta}}\sqrt{1 - e^{-2\theta\Delta}} \cdot Z_i \\ &0 \leq i \leq 99, i \in \mathbb{N} \end{aligned}$$

with $Z_i \sim \mathcal{N}(0, 1)$ independent and identically distributed standard normal random variables. In order to simplify the notation, we refer to $t_i \equiv i$.

In this chapter we introduce 3 conceptually different agents, which either use Q Learning or Deep Reinforcement Learning. In addition, we consider two different inventory restrictions for the agents. The simplest environment only allows an exposure of one stock in both directions (long/short) of the market, whereas the more advanced setting allows a maximum long/short position of 5 stocks. Apart from finding an optimal strategy each agent aims at performing better than the Bollinger Band strategy. If $X_t > BOLU_t$ the Bollinger Band strategy sells and if $X_t < BOLD_t$ the strategy

buys with

$$
\begin{aligned}
&BOLU_t := EWMA_t + \sqrt{EWMVAR_t} \\
&BOLD_t := EWMA_t - \sqrt{EWMVAR_t} \\
&EWMA_t := \alpha \sum_{i=0}^{t} X_i (1-\alpha)^{t-i} \\
&EWMA_{t+1} := \alpha X_{t+1} + (1-\alpha) EWMA_t \\
&EWMVAR_{t+1} := (1-\alpha)(EWMVAR_t + \alpha(X_{t+1} - EWMA_t))^2.
\end{aligned} \tag{2.0.3}
$$

The main goal of the agents is to maximise the profits from trading (Profit and Loss, PnL) over a number of 100 time steps. Accordingly, the problem is setup as an episodic task and $0 \leq t < 100$, $t \in \mathbb{N}$. In this chapter the agents action will not have a market impact on the price of the stock. All agents act ϵ-greedy, meaning that at each time step with probability ϵ a random action and with probability $1-\epsilon$ the action corresponding to $\max_a Q(s,a), s \in \mathcal{S}, a \in \mathcal{A}(s)$ is chosen. The initial-ϵ is always 1. Thus, especially in the beginning the agents mainly explore. After each episode ϵ is decayed: $\epsilon_{t+1} = \epsilon_t \cdot \eta$, $\eta \in [0,1]$. In the following, η is also called ϵ-decay. The pseudo code looks as follows:

Algorithm 7: Reinforcement Learning Framework for all agents

```
Initialize agent (Q Learning or DQN) including the parameters γ, ϵ, η, minimum-ϵ,
  Number Episodes, α/ learning rate, Q-tabular/Network architecture;
Episode=1
while Episode ≤ Number Episodes do
    Generate Ornstein–Uhlenbeck process X_t
    Initialize first state S_0
    PnL = 0 ;
    done = FALSE;
    t=0;
    while TRUE do
        ϵ-greedy agent selects action A_t
        Determine next state S_{t+1}
        Calculate reward R_{t+1}
        Update PnL;
        if Q Learning agent then
            Update Q-values;
        end
        if DQN agent then
            Append (S_t, A_t, R_{t+1}, S_{t+1}) to replay memory;
        end
        t+=1 ;
        if t == 100 then
            Save PnL ;
            break;
        end
    end
    if DQN agent then
        Sample batch size from replay memory;
        Determine target vector Y for batch size;
        Fit network(s) of the DQN agent;
    end
    ϵ_{t+1} = min(ϵ_t * η, minimum-ϵ)
    Episode+=1
end
```

The implementation is done with the programming language `Python`. The code is publicly available on `https://github.com/arvidbertermann/RL_lib`. A mathematical analysis and practical guidelines of how to optimally trade the Ornstein–Uhlenbeck process can be found in [19].

2.1 One Long/Short Agents

In this chapter we introduce the agents with the one long/short inventory restriction. A description of the general setup is followed by detailed information about the different agents including first qualitative insights into the learning success of the agents. In the end a quantitative analysis evaluates and compares the different agents.

When setting up a Reinforcement Learning algorithm one has to specify the state space $\mathcal{S}$, the action space $\mathcal{A}$ and the reward-function $\mathcal{R}$. In this setting a single state consists of three components $S_t = (S_t^1, S_t^2, S_t^3) \in \mathcal{S}$. Depending on the Reinforcement Learning algorithm the specific information/values delivered by a state vary. In general:

- S_t^1 gives information about the current value of the stock price.
- S_t^2 gives information about the value of the stock price at the last point of time when the exposure was switched from either -1 to 1 (short to long) or 1 to -1 (long to short).
- $S_t^3 \in \{-1, 1\}$, with -1 (1) indicating the short (long) exposure of the agent.

The state space $\mathcal{S}$ can be divided into two different sub-spaces $\mathcal{S}^+$ and $\mathcal{S}^-$, solely depending on the third component S_t^3 of the current state S_t. In detail, $S_t^3 = -1 \implies S_t \in \mathcal{S}^-$, $S_t^3 = 1 \implies s_t \in \mathcal{S}^+$. The inventory of the agent directly impacts the action space $\mathcal{A}(s)$:

$$S_t^3 = -1 \implies A_t \in \{n, b\},\ S_t^3 = 1 \implies A_t \in \{s, n\} \tag{2.1.1}$$

with n ("nothing") meaning that the exposure does not change, b ("buy"/"short to long") resulting in neutralising the short position and buying another share and similarly s ("sell"/"long to short") resulting in neutralising the long position and selling another share. Thus, the exposure S_t^3 remains either -1 or 1. The reward function is defined by the following map $\mathcal{R} : \mathcal{S} \times X \times X \longrightarrow \mathbb{R}$:

$$R_{t+1} = S_{t+1}^3 (X_{t+1} - X_t). \tag{2.1.2}$$

Interpreted from a financial market perspective, the reward is the current exposure times the stock performance from time t to $t+1$. As an example, if the agent is long ($S_{t+1}^3 = 1$) and the stock price jumps from $X_t = 99$ to $X_{t+1} = 100$ the reward is 1. The PnL is defined as follows

$$PnL_e = 2 \sum_{t=1}^{99} X_t (\mathbb{1}_{\{A_t = s\}} - \mathbb{1}_{\{A_t = b\}}) + S_{99}^3 X_{99}. \tag{2.1.3}$$

At $t = 99$ the agent neutralises its position.

2.1.1 Q Learning

The first, most simplified agent (Q agent), uses Q Learning to learn optimal investment decision making. As mentioned in section 1.1.5 this algorithm adjusts its Q-values according to the following update rule:

$$Q^{new}(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \cdot (R_t + \gamma \cdot \max_a Q(S_{t+1}, a) - Q(S_t, A_t)).$$

In the introduction to this chapter the action space $\mathcal{A}$, the reward function $\mathcal{R}$ and the third component of the state space $\mathcal{S}$ have already been explicitly defined. Q Learning is a tabular method, which lists all state–action pairs and keeps track of $Q(s, a)$ over time. Independent of the exposure, we have two different actions for each state as explained in (2.1.1). The memory needed to store the Q-values depends on the number of possible combinations of S_t^1 and S_t^2. If one would choose the first two components to be continuous values, Q Learning would not be a suitable method to train an agent, because a single state would most likely be visited at most once due to the infinite number of possible combinations. Hence, we discretise the state space in the following way: We instantiate a process-memory $M_{t,e}^X$, which contains the last 1000 observations

of the price process across episodes e at time t. Based on a vector $b_{t,e}$ of barriers, which represent quantile-values of $M^X_{t,e}$, the bucket numbers of S^1_t and S^2_t are determined. In detail,

$$b_{t,e} := [\min M^X_{t,e}, q_{0.1}^{M^X_{t,e}}, q_{0.2}^{M^X_{t,e}}, q_{0.4}^{M^X_{t,e}}, q_{0.6}^{M^X_{t,e}}, q_{0.8}^{M^X_{t,e}}, q_{0.9}^{M^X_{t,e}}, \infty] \ , \ 0 \leq t < 100 \ , \ t \in \mathbb{N} \tag{2.1.4}$$

$$S^1_t = i + 1, \text{ such that } X_t \in [b_{t,e}[i], b_{t,e}[i+1]) \text{ with } b_{t,e}[0] = \min M^X_{t,e} \ , \ 0 \leq t < 100 \ , \ t \in \mathbb{N} \tag{2.1.5}$$

$$S^2_t := S^1_k, \text{ such that } \max_k S^3_t \neq S^3_k \ , \ 0 \leq k < t < 100 \ , \ k, t \in \mathbb{N} \tag{2.1.6}$$

where $q_p^{M^X_{t,e}}$ represents the $p \cdot 100\%$ quantile value of $M^X_{t,e}$.

In summary, S^1_t and S^2_t take integer values from 1 to 7 resulting in 49 possible combinations with two different exposure values $(-1, 1)$ and two different actions (b/n, s/n). Hence, this setup consists of $|\mathcal{S}| = 98$ states and $|Q(s, a)| = 196$ Q-tabular values with $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$. All Q-values are initialised to 0. The first state S_0 of each episode always starts with $(S^1_t, S^2_t) = (4, 4)$ and $S^3_t = Y$, $Y \in \{-1, 1\}$ with $\mathbb{P}(Y = -1) = \mathbb{P}(Y = 1) = 0.5$.

As mentioned before, in this section, we would like to emphasise the learning progress of the agent from a qualitative perspective. At later stages, the PnL of several agents and properties of their convergence will be investigated quantitatively. All results shown regarding the Q agent were generated with the following parameter setup:

$$\alpha = 0.1 \ , \ \gamma = 0.9 \ , \ \text{episodes} = 2500 \ ,$$
$$\text{initial-}\epsilon = 1 \ , \ \text{minimum-}\epsilon = 0.01 \ , \ \epsilon\text{-decay} = 0.998$$

Figure 2.1 illustrates that in the beginning the exposure-changing actions have similar distributions with respect to the value of the process. This is due to the high initial exploration of the agent. The upper right graph shows, that the agent starts to learn a strategy as the action-histograms move away from each other. The lower left graph confirms that the agent clearly develops a tendency to "buy low and sell high" after 1400 episodes by separating the values of the stock price into a "buy" and "sell" dominated region. To further evaluate the agent, Figure 2.2 displays the detailed investment process of the first and last episode including the PnL development over time. Matching the initial observations from Figure 2.1, the decisions taken by the agent in the left graph are mixed, whereas the right plot shows a clear distinction between "long to short" and "short to long". The learned strategy unsurprisingly results in a higher profit.

Figure 2.1: Action splitting of Q agent demonstrated by histograms of long to short/short to long investment decisions.

Finally Figure 2.3 reveals the state–action values $Q(s, a)$. The number on the x-axis can be interpreted as follows: The sign determines the exposure, meaning that -/+ indicates a short/long position. The first/second digit represents S^1_t/S^2_t of the state S_t. For instance, $S_t = -57$ means

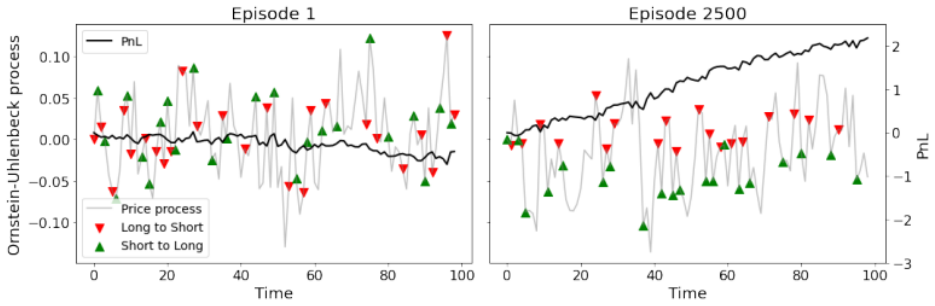

Figure 2.2: Profit and Loss (2.1.3) including investment decisions of Q agent for the first and last episode.

that at time t the value of the process is in bucket number 5 and that the last "long to short" action was taken in bucket number 7. The upper plot shows that given a long exposure, the agent prefers doing "nothing" for low values $(11 - 37)$ of S_t, whereas for high values $(50 - 77)$ the agent decides to change its exposure. These observations are independent of the second digit. The lower plot displays similar patterns : The agent holds its position for low states $((-77) - (-51))$ and changes its exposure for high S_t $((-37) - (-11))$ given an inventory of -1.

The middle bucket 4 is unsurprisingly the only one, where the action of the agent depends on the last exposure change. Most likely all values of the process X_t represented by the middle bucket are close to the mean-reverting value μ. Thus, it is the most unpredictable value-region of the process. This might be the reason for the mixed strategy of the agent and the dependence on the second digit. In addition, there are large differences between Q-values given the first digit and the exposure of the state. In detail, high (low) values of S_t^2 and $S_t^3 = 1$, $(S_t^3 = -1)$ result in significantly lower $Q(s, a)$, which intuitively makes sense as the agent bought (sold) the stock at a relatively high (low) price.

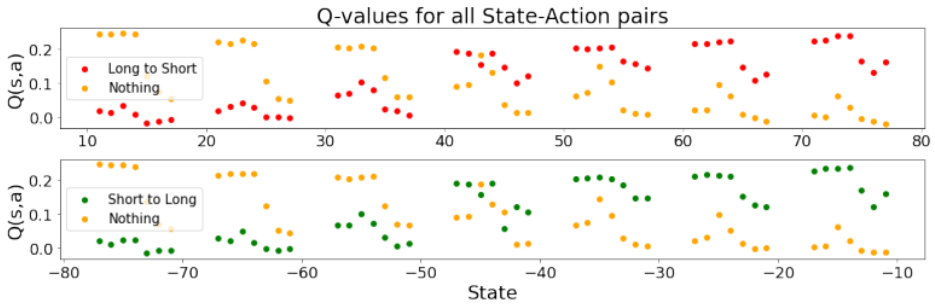

Figure 2.3: Q-values for all state–action pairs after 2500 episodes of training.

In conclusion, the bucket representing the current value of X_t and the exposure dominate the strategy of the agent. However, S_t^2 clearly affects the expected return of a state–action pair but mostly not to the extent that it would change the agents decision given S_t^1. The second component of the state only impacts the decision making for the most unpredictable part of the price process. Hence, the agent might learn as successfully without the S_t^2 information. This would simplify the task even more to only $|\mathcal{S}| = 14$ states and $|Q(s, a)| = 28$. In summary, the agent:

- Changes its exposure from "long to short" given high stock prices and positive exposure.
- Changes its exposure from "short to long" given low stock prices and negative exposure.
- Holds its position given high stock prices and negative exposure.

- Holds its position given low stock prices and positive exposure.

2.1.2 Deep Q Network

In the last chapter we discretised the Ornstein–Uhlenbeck process into 7 buckets and used a tabular method to keep track of the Q-values. In this chapter we use Deep Q Learning. Hence, the agent (*DQN* agent) uses a neural network to approximate state–action values. As a neural network is able to work with continuous values, the first and second component of the state are defined as follows:

$$S_t^1 = X_t \ , \ 0 \leq t < 100 \ , \ t \in \mathbb{N}$$

$$S_t^2 = S_k^1, \text{ such that } \max_k S_t^3 \neq S_k^3 \ , \ 0 \leq k < t < 100 \ , \ k, t \in \mathbb{N}.$$

The input to the network is the state S_t and output is a vector of predicted $Q(s,a)$-values, $s \in \mathcal{S}, a \in \mathcal{A}(s)$. Each tuple of $(S_t, A_t, R_{t+1}, S_{t+1})$ (state, action, reward and next state) is saved into a replay memory containing 2000 observations. After every episode (100 time steps) a batch of 32 observations is randomly sampled from the replay memory. A neural network optimises its predictions by minimising a loss function, which compares predictions $\hat{Y}_t$ with the target values Y_t. In our case the target values are determined by:

$$\begin{aligned} Y_t(A_t) &:= R_t + \left(\gamma \cdot \max_{a \in \mathcal{A}(s)} Q(S_{t+1}, a)\right) \cdot \mathbf{1}_{\{t<99\}} \\ Y_t(a) &:= \hat{Y}_t(a) \ , \ a \neq A_t \end{aligned} \tag{2.1.7}$$

The loss function used is the mean squared error:

$$MSE(Y, \hat{Y}) := \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.$$

The neural network itself consists of the 3-dimensional input layer, 3 hidden layers with 48 neurons using the tanh activation function and a three dimensional output layer using the linear activation function. The dimension of the output layer equals the number of actions. It is crucial to remember that only two of these three actions will be taken into account in the algorithm itself depending on the exposure S_t^3 of the state S_t (see (2.1.1)). For example, if $S_t = (2.2, 6.1, -1)$, the action to be selected is either "buy" or "nothing". Lets assume the *DQN* agent decides to change its exposure from short to long (buy), $A_t = b$, which results in $S_{t+1} = (5.4, 2.2, 1)$. The determination of the target value Y_t as shown in (2.1.7) uses the $\max_a Q(S_{t+1}, a), a \in \mathcal{A}(S_{t+1})$ function over all "activated" actions. In this particular example $a \in \mathcal{A}(S_{t+1}) = \{s, n\}$.

All results shown below use the following parameter setup:

$$\begin{aligned} &\text{learning rate } = 0.001 \ , \ \gamma = 0.9 \ , \ \text{episodes} = 2500 \ , \\ &\text{initial-}\epsilon = 1 \ , \ \text{minimum-}\epsilon = 0.01 \ , \ \epsilon\text{-decay} = 0.998 \end{aligned}$$

Figure 2.4 confirms again that the *DQN* agent learns to buy low and sell high. In comparison to the *Q* agent, the neural network seems to need more time to detect the buy and sell region of the Ornstein–Uhlenbeck process. The peaks of the histograms between episode $2400 - 2500$ have moved further away from each other than in the Q Learning setup (Figure 2.1). This results most likely in fewer investments as extreme values are less frequently visited. In addition, the range of values covered by the "long to short"/"short to long" histograms after 2500 episodes is larger. This could lead to smaller profits from trading as buying/selling at relatively high/low values of the process is on average not profitable.

Figure 2.5 reveals that the *DQN* Agent seems to be more "passive" than the *Q* agent. The left plot clearly indicates that doing nothing dominates the exposure changing actions. Moreover, by comparing episode 2485 and 2500, the *DQN* agent seems to follow a different strategy. An optimal agent is expected to find the optimal strategy and stop changing its investing approach. The memory replay, which is responsible for fitting the neural network after each episode, appears to constantly change the approximation of the Q-values, such that the agent follows a significantly

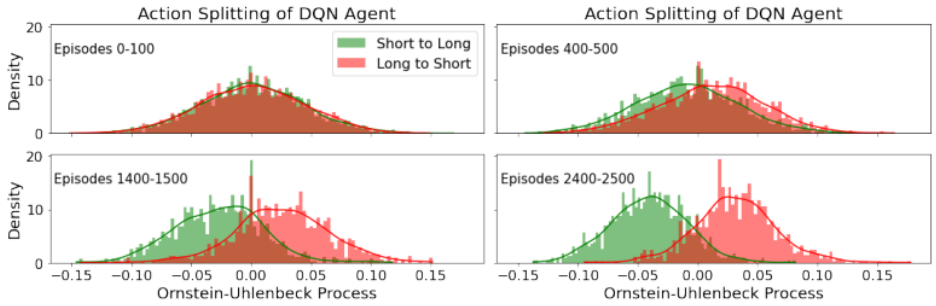

Figure 2.4: Action splitting of *DQN* agent demonstrated by histograms of long to short/short to long investment decisions.

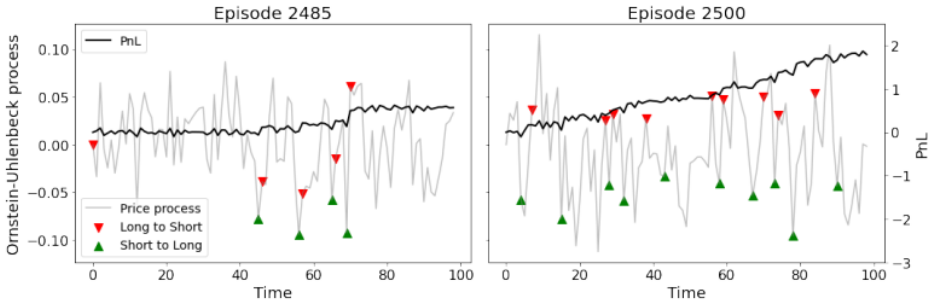

Figure 2.5: Profit and Loss (2.1.3) including investment decisions of *DQN* agent for the 2485-th and last episode.

different policy. Both simulations of the mean-reverting process offer a higher investment potential than the *DQN* agent is able to exploit. A possible reason, why the *DQN* struggles to trade more actively, might be the ignoring of the third action depending on the current exposure of the state. Given $S_t^3 = 1$ ($S_t^3 = -1$) the approximated $Q(S_t, \text{“}buy\text{”})$ ($Q(S_t, \text{“}sell\text{”})$) is always assumed to be correctly predicted but will never be used at any point of time. Hence, one third of the networks approximations of $Q(S_t, a)$ are not taken care of but are implicitly assumed to agree with the target value. In the next section we attack this point of neglect by slightly changing the architecture of the *DQN* agent.

2.1.3 Interactive Deep Q Networks

This chapter presents an experimental approach to avoid the aforementioned problem of approximating the vector of $Q(s, a)$ values, $s \in \mathcal{S}, a \in \mathcal{A}(s)$ of which not all actions are “activated”. Before, we set up an agent consisting of a single neural network responsible for all decisions independent of the exposure. In this chapter the *iDQN* agent is defined by two sub agents, the *Buy iDQN* and *Sell iDQN* agent, each using a neural network. As the name suggests, the *Buy iDQN* agent is activated when the exposure $S_t^3 = -1$. It is able to hold or to change the exposure to $S_{t+1}^3 = 1$ by neutralising the short position and buying another share of the stock. Vice versa, the *Sell iDQN* agent is activated when the exposure $S_t^3 = 1$. It is able to hold or to change the exposure to $S_{t+1}^3 = -1$ by neutralising the long position and selling another share of the stock. Both sub-agents have their own replay memory with up to 2000 observations and are trained after each episode with one batch of size 32. Obviously, the *Buy iDQN*/*Sell iDQN* agents replay memory only consists only of states S_t with $S_t^3 = -1/S_t^3 = 1$. In order to determine the target values Y_t the exposure of the next state S_{t+1}^3 is crucial:

$$Y_t(A_t) := R_t + \left(\gamma \cdot \max_{a \in \{n,b\}} Q_{Buy}(S_{t+1}, a)\right) \cdot \mathbb{1}_{\{t<99\ ,\ S^3_{t+1}=-1\}} \left(\gamma \cdot \max_{a \in \{s,n\}} Q_{Sell}(S_{t+1}, a)\right) \cdot \mathbb{1}_{\{t<99\ ,\ S^3_{t+1}=1\}}$$

and

$$Y_t(a) := \hat{Y}_t(a)\ ,\ a \neq A_t$$

with Q_{Buy}/Q_{Sell} indicating that the neural network of the *Buy iDQN/Sell iDQN* sub-agent is used to approximate the state–action values. As both sub-agents are responsible for the investment process and due to the collaboration we name this approach *Interactive DQN (iDQN)*. The structure of the neural networks as well as the parameter setup is exactly the same as in the DQN approach.

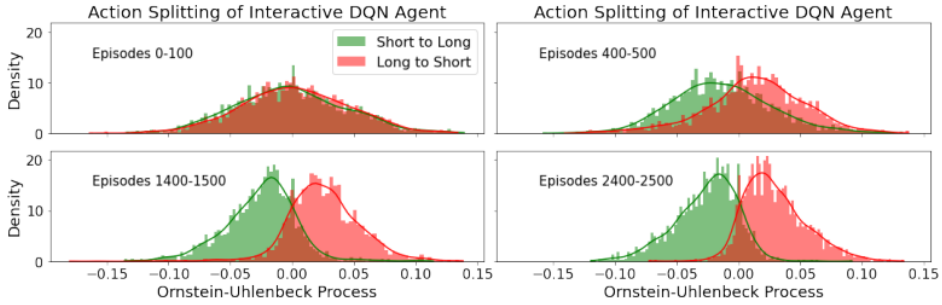

Figure 2.6: Action splitting of *iDQN* agent demonstrated by histograms of long to short/short to long investment decisions.

Figure 2.6 displays that the *iDQN* agent develops the "buy low sell high" strategy quicker than the *DQN* agent. Moreover, the shape and positioning of the histograms looks similar to the *Q* agent in Figure 2.1. Figure 2.7 again shows two examples of how the algorithm traded. After 250 episodes the *iDQN* agent was still selling the stock at relatively low values, which partially might be caused by exploration. The right graph proves that the *iDQN* agent optimised its decision making and also realises the majority of the investment opportunities.

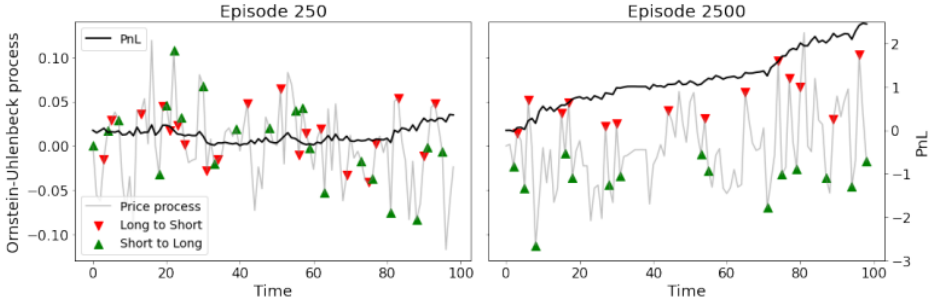

Figure 2.7: Profit and Loss (2.1.3) including investment decisions of *iDQN* agent for the 250-th and last episode.

2.1.4 Evaluation and Comparison

After investigating the learning progress of the agents from a qualitative perspective, we now inspect the quantitative progress. Figure 2.8 shows the rolling mean and standard deviation of the

episodic PnL with a window of 100 observations for the three agents and the deterministic Bollinger Bands strategy as a benchmark. The left graph illustrates that the Q as well as the $iDQN$ agent beat the benchmark after approximately 750 episodes, whereas the normal DQN agent matches the Bollinger Bands performance after 2000 episodes. The right graph indicates that the rolling standard deviation of episodic PnLs is similarly decreasing for the Q and the $iDQN$ agent. In contrast, the fluctuation of episodic PnLs increases for the DQN agent. As the Bollinger Bands is a deterministic strategy it does not surprise that its standard deviation is lower than the one of the Reinforcement Learning agents.

Table 2.1 summarises the trends seen in Figure 2.8 but also gives some information about the aggressiveness and efficiency of the agents. In the beginning all three agents average roughly 50 investments per episode. An investment is considered to be any exposure changing action (long to short, short to long). After 2500 episodes the DQN agent averages 21 investment decisions, whereas the Q and $iDQN$ agent changes its exposure almost 40 times per 100 time steps. Hence, the probability of observing an exposure changing action when investing with the Q and $iDQN$ agent is almost twice as high. The profitability per exposure changing action of the DQN agent is significantly higher than the one for the Q Learning and iDQN agent.

Q Learning	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	0.1671	0.4839	49.46	0.0034
Episodes 400-500	1.2840	0.4183	46.51	0.0276
Episodes 1400-1500	2.0687	0.4418	39.61	0.0522
Episodes 2000-2100	2.1129	0.3542	39.09	0.0541
Episodes 2400-2500	2.1381	0.3956	38.64	0.0553

DQN	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	-0.0061	0.4916	49.15	-0.0001
Episodes 400-500	0.6417	0.5354	38.74	0.0166
Episodes 1400-1500	1.1383	0.7223	21.89	0.0520
Episodes 2000-2100	1.4255	0.6826	21.79	0.0654
Episodes 2400-2500	1.5758	0.6206	20.98	0.0751

Interactive DQN	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	0.1909	0.5203	49.08	0.0039
Episodes 400-500	1.1925	0.4364	45.28	0.0263
Episodes 1400-1500	1.9681	0.4131	37.51	0.0525
Episodes 2000-2100	2.1081	0.4566	39.53	0.0533
Episodes 2400-2500	2.1227	0.3341	38.37	0.0553

Table 2.1: Statistics Table for the three Agents : $\overline{PnL}$ is the average PnL (2.1.3) and σ_{PnL} is the standard deviation of the episodic PnLs considering the range of episodes in the first column. An investment is considered to be an exposure changing action.

In summary,

- *Convergence:* The Q and the $iDQN$ agent learn faster than the DQN agent and seem to have converged after 2500 episodes. The DQN agent still changes its investment strategy after 2500 episodes but temporarily finds a comparable optimal strategy (Figure 2.5).
- *PnL:* The Q and the $iDQN$ agent beat the Bollinger Bands but their PnLs have a higher standard deviation. The DQN agent matches the benchmarks performance on average but the PnL has a significantly higher standard deviation.
- *Aggressiveness:* The Q and the $iDQN$ agent invest twice as much on average compared to the DQN agent, which partially explains the better PnL performance.

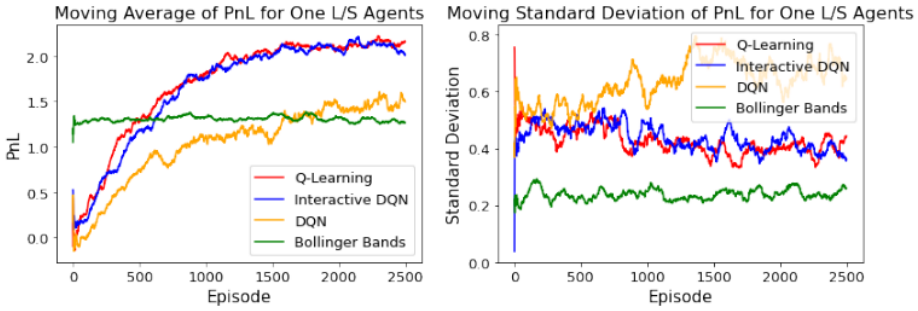

Figure 2.8: *Left:* Comparison of the three agent's moving average PnL (2.1.3). *Right:* In order to get a feeling for the convergence/stability of the learned strategy, the rolling standard deviation of episodic PnLs is shown. Both statistics use a rolling window of 100 observations. The Bollinger Bands (2.0.3) are used as a benchmark.

- *Profitability per trade:* The profit per exposure changing action for the *DQN* agent is $\approx 35\%$ higher when compared to the *Q* and *iDQN* agent.

As a conclusion, for the first restriction setup of one long/short the more sophisticated Deep Q Networks are not performing better than the simple tabular method.

2.2 Inventory Agents

In this chapter we increase the complexity of the learning task by allowing the agent to build up an inventory of up to 5 shares in either direction of the market. The first two components of a state $s \in \mathcal{S}$ remain unchanged. The third component S_t^3 can now take values $i \in [-5, 5], i \in \mathbb{Z}$. The action space $\mathcal{A}$ consists of five actions. In the previous setup we allowed the agent to double buy, double sell and do nothing. We add the actions single buy and single sell. The third component of a state determines which actions are going to be activated. If $|S_t^3| \leq 3$ all actions are available as the agent will stay within the inventory restrictions independent of the action. If $S_t^3 = 4$ ($S_t^3 = -4$), holding the exposure, single buy, single sell and double sell (buy) will be activated. If the inventory reached the upper (lower) boundary it is not longer allowed to buy (sell) resulting in three available actions with holding the exposure, single sell (buy) and double sell (buy). When updating $Q(S_t, A_t)$, one cautiously needs to look at the exposure of the next state S_{t+1}^3 as it determines $\hat{\mathcal{A}}(S_{t+1})$ with $\max_a Q(S_{t+1}, a), a \in \hat{\mathcal{A}}(S_{t+1})$:

$$\begin{aligned} &|S_{t+1}^3| < 4 \implies a \in \{ds, s, n, b, db\}\ , \\ &S_{t+1}^3 = -5 \implies a \in \{n, b, db\}\ , \\ &S_{t+1}^3 = -4 \implies a \in \{s, n, b, db\}\ , \\ &S_{t+1}^3 = 4 \implies a \in \{ds, s, n, b\}\ , \\ &S_{t+1}^3 = 5 \implies a \in \{ds, s, n\}\ , \end{aligned} \tag{2.2.1}$$

with s ≡ sell, b ≡ buy, d ≡ double and n ≡ nothing.

The reward function is the same as in (2.1.2). Every episode starts with an exposure $S_0^3 = 0$ and with $S_t^1 = S_t^2 = 4$ for the Q Learning and $S_t^1 = S_t^2 = 0$ for the deep Reinforcement Learning agents. All other structural aspects of the Reinforcement Learning algorithm are set up as in the previous learning task in section 2.1. As a consequence the reward function does not measure the change in PnL but the isolated immediate return of the action. The PnL in this task differs slightly from (2.1.3):

$$PnL_e = 2\sum_{t=1}^{99} X_t(\mathbb{1}_{\{A_t=ds\}} - \mathbb{1}_{\{A_t=db\}}) + \sum_{t=1}^{99} X_t(\mathbb{1}_{\{A_t=s\}} - \mathbb{1}_{\{A_t=b\}}) + S_{99}^3 X_{99}. \tag{2.2.2}$$

2.2.1 Q Learning

We start with the tabular method Q Learning in the more complex setting. The first two components of a state s are computed as shown in (2.1.4) - (2.1.6). Due to the larger action space and the new inventory restrictions we have $|\mathcal{S}| = 539$ states and $|Q(s,a)| = 2401$ state–action values. Hence, more than 10 times memory is required when compared to the previous learning task.

In Figure 2.9 histograms for a range of episodes of the four exposure changing actions are shown. In comparison to the simple Q agent, the peaks of the double sell and double buy histograms are further away from each other, meaning that the agent only buys (sells) when the price is extremely low (high). The space between the extreme actions is filled with the single buy and single sell actions. Hence, for the unpredictable region of the process the Q *Inv* agent invests more carefully. Interestingly, one can observe a fat left (right) tail of the single buy (sell) action, which could be due to the inventory restrictions. Imagine the exposure is 4 and the process is in bucket 2. The right decision would be to buy but as the maximum exposure is 5, only the single buy action is activated. Figure 2.10 confirms the previously mentioned patterns.

In summary, the Q *Inv* agent learns to double buy low and to double sell high. The single buy and single sell options seem to be used in order to optimise the inventory and staying within the inventory restrictions but also to invest in more uncertain situations.

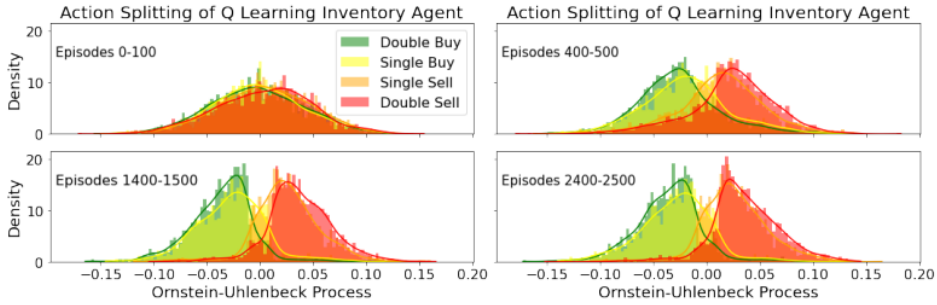

Figure 2.9: Action splitting of Q *Inv* agent demonstrated by histograms of exposure changing actions.

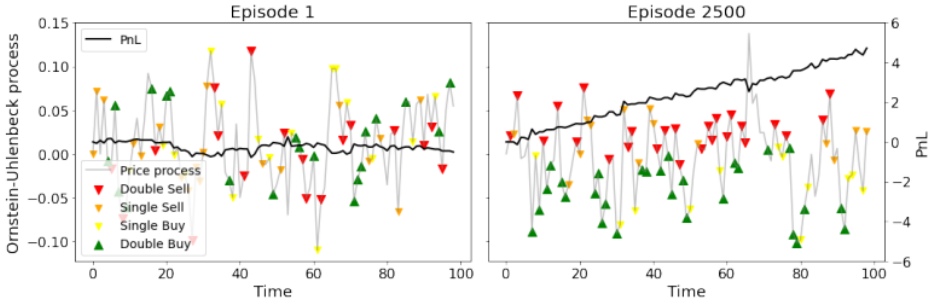

Figure 2.10: Profit and Loss (2.2.2) including investment decisions of Q *Inv* agent for the first and last episode.

2.2.2 Deep Q Network

The next algorithm to investigate is the *DQN Inv* agent. As the learning task is more complex we increase the number of neurons to 64 in the first, 128 in the second and 64 in the third fully connected layer of the network. The input layer, all activation functions and the loss function remain

unchanged. The output layer consists of a 5 dimensional vector approximating the Q-values. In order to determine the target vector for each observation in the memory replay equations (2.1.7) and (2.2.1) are used.

Comparable to the simple learning task, the *DQN Inv* agent needs longer to separate the actions than the *Q Inv* agent (Figure 2.9,2.11) . Furthermore, after 1500 episodes the algorithm seems to not distinguish between double and single buy as the histograms look almost identical (Figure 2.11). After 2500 episodes, similar to the *Q Inv*, double buy and double sell have a similar shape and seem to be mirrored at $X_t = 0$. However, the histograms of the single buy and single sell action have a completely different location and shape. The right plot in Figure 2.12 confirms that the *DQN Inv* agent prefers to single sell around the mean-reverting value μ, whereas the single buy action is hardly used. To build up a positive exposure the double buy action appears to be clearly dominating.

As a result, the *DQN Inv* agent recognises the buy low and sell high pattern but seems to learn an asymmetric investing behaviour for the single buy and sell actions.

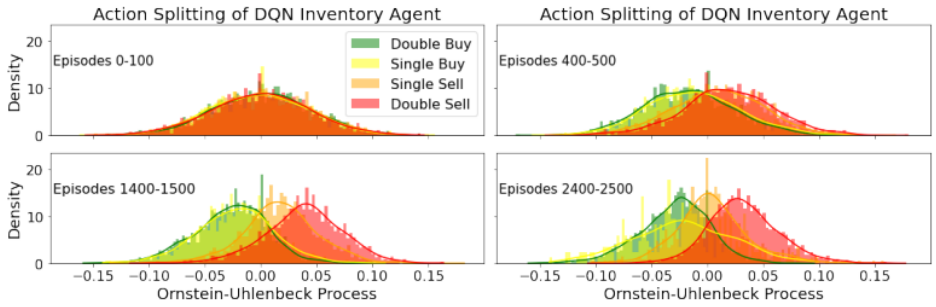

Figure 2.11: Action splitting of *DQN Inv* agent demonstrated by histograms of exposure changing actions.

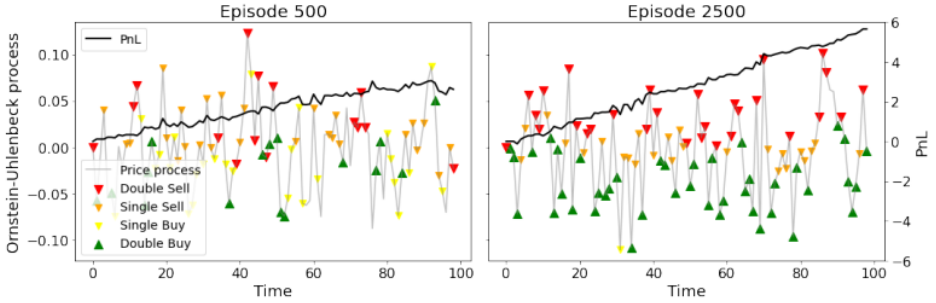

Figure 2.12: Profit and Loss (2.2.2) including investment decisions of *DQN Inv* agent for episode 500 and 2500.

2.2.3 Interactive Deep Q Networks

The *iDQN Inv* agent consists again of a *Buy iDQN Inv* and *Sell iDQN Inv* sub-agent. Apart from the output layer each of their networks has the same structure as the one for the *DQN Inv* agent. The output layer for the interactive sub-agents is three-dimensional. The *Buy iDQN Inv* network can hold the exposure, buy one and buy two shares and the *Sell iDQN Inv* network can also hold the exposure, sell one and sell two shares given that the action would result in a new state that does not violate the inventory restrictions. Apart from states where the maximum exposure is

reached, the buy and the sell sub-agent are going to be activated. If the agent acts greedily, the maximum of all Q-values will be chosen to be the next action:

$$A_t = \underset{a_1 \in \mathcal{A}_{Buy}(S_t), a_2 \in \mathcal{A}_{Sell}(S_t)}{\arg\max} \Big(Q_{Buy}(S_t, a_1), Q_{Sell}(S_t, a_2) \Big)$$

where $\mathcal{A}_{Buy}$ and $\mathcal{A}_{Sell}$ are derived from the same logic as in (2.2.1) and the maximum set of possible actions for each sub-agent.

Both sub-agents have their own replay memory. If a state S_t is followed by $A_t \in \{b, db\} / A_t \in \{ds, s\}$ the tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ is appended to the buy/sell replay memory. If $A_t = n$ and $|S_t^3| < 5$ the tuple is added to both replay memories. Otherwise the data will be added to the agent that is the only one active. Remember that when the overall interactive agent holds the maximum exposure of $5/-5$ only the *Sell iDQN Inv*/*Buy iDQN Inv* agent is activated in order to stay within the inventory restrictions. During the memory replay the target vector Y_t is determined as follows:

$$Y_t(A_t) := R_t + \Big(\gamma \cdot \max_{a_1 \in \mathcal{A}_{Buy}(S_{t+1}), a_2 \in \mathcal{A}_{Sell}(S_{t+1})} \Big(Q_{Buy}(S_{t+1}, a_1), Q_{Sell}(S_{t+1}, a_2) \Big) \Big) \cdot \mathbf{1}_{\{t<99\}}$$

$$Y_t(a) := \hat{Y}_t(a) \ , \ a \neq A_t.$$

Figure 2.13 illustrates once more the action splitting. The agent seems to learn symmetrically to distinguish between the double buy (sell) and single buy (sell) action. Similar to the *Q Inv* agent, around the mean-reverting value μ the most popular exposure changing actions are the single buy and sell. Both single action histograms have a fat tail, which could be due to maximise the exposure potential when $S_t^3 = 4/S_t^3 = -4$.

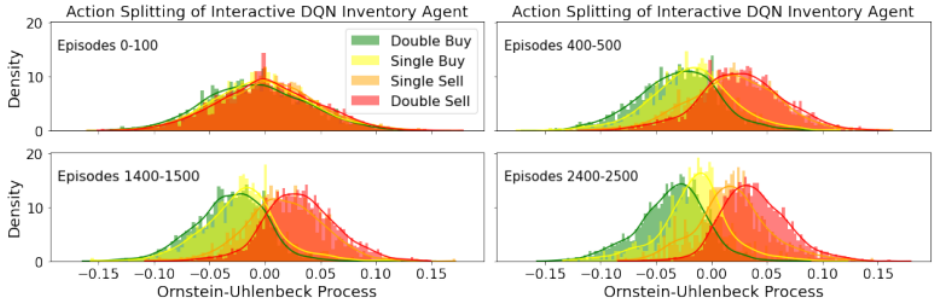

Figure 2.13: Action splitting of *iDQN Inv* agent demonstrated by histograms of exposure changing actions.

Figure 2.14 indicates that the agent may change its strategy over time. After 1500 episodes the double buy and sell actions are the only exposure changing actions the agent makes use of. Contrary to that, after 2500 episodes regarding exposure reducing decisions, the single sell action is approximately chosen as often as the double sell action. Interestingly, the single buy action seems to be very unpopular at both stages of the learning process.

The agent successfully manages to buy low and sell high. Especially because of Figure 2.14 the agents behaviour might be dominated by both double actions. This seems to be a repeating phenomena as we have seen similar behaviour in the *DQN Inv* example.

2.2.4 Evaluation and Comparison

Finally, we proceed with the quantitative evaluation. From the individual analysis of the agents, it seems that the only one that has certainly converged to a not longer changing strategy is the *Q Inv* agent. The almost constant PnL performance and standard deviation after 1000 episodes in Figure 2.15 confirm that. In contrast, both DQN agents are still improving their average performance

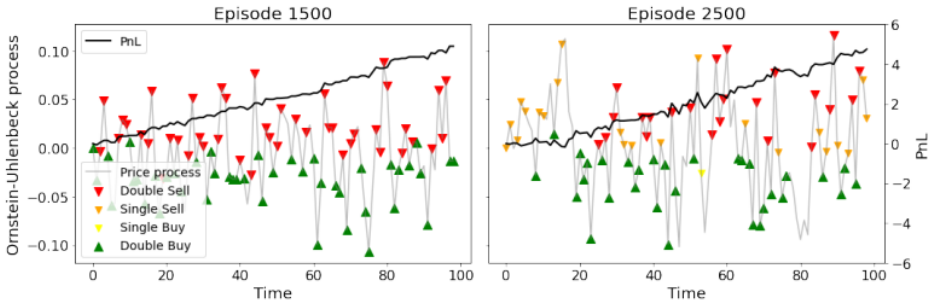

Figure 2.14: Profit and Loss (2.2.2) including investment decisions of *iDQN Inv* agent for episode 1500 and 2500.

after 2500 episodes and beat the tabular method after $\approx$ 1750 episodes. However, the standard deviation of the agents using neural networks is still higher even though it starts to decreases after 1000 episodes. Clearly, all agents beat the Bollinger Bands performance but the episodic PnL fluctuates more heavily. In comparison to the simple learning task (One L/S), the opportunity for the agent to build up inventory results in a PnL that is twice as high.

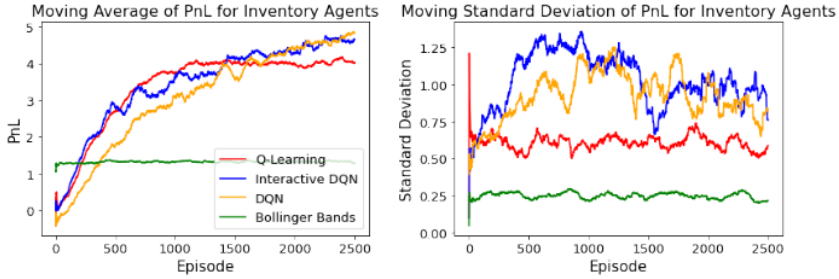

Figure 2.15: *Left:* Comparison of the three inventory agent's moving average performance. *Right:* In order to get a feeling for the convergence/stability of the learned strategy, the rolling standard deviation of episodic PnLs (2.2.2) is shown. Both statistics use a rolling window of 100 observations. The Bollinger Bands (2.0.3) are used as a benchmark.

If an agent has converged it can be assumed that over multiple episodes the action probabilities remain relatively constant (in a simulated environment). In addition, due to the previously made observation that the DQN agents hardly and asymmetrically use the single buy and single sell actions we investigate the development of action probabilities:

$$\mathbb{P}_e(a) := \frac{\sum_{i=e-99}^{e} \sum_{j=0}^{99} \mathbb{1}_{\{A_j^i = a\}}}{\#a}$$

with A_t^e the t-th action of episode e and $\#a$ = Total number of actions = Episodes * Actions per episode = 100^2. Let us interpret $\mathbb{P}_e(a)$ as a random, continuous process following a classical drift-diffusion stochastic differential equation.

$$dP(a)_t := \mu(t)dt + \sigma(t)dW_t$$

with W_t a standard Brownian Motion. In order to confirm that an action probability has converged, we would like to find a time t_C such that the local standard deviation is constant and optimally small, e.g. $\sigma(t) = C, C \in \mathbb{R}_+, \forall t \geq t_C$ and there is no drift, $\mu(t) = 0, \forall t \geq t_C$.

Figure 2.16 reveals that for both actions (double sell, single buy) after $\approx$ 1250 episodes *Q Inv* agent has converged. The action probabilities of the *DQN Inv* agent still show a trend and high,

non-constant local standard deviation. Between the strong and stable convergence of the tabular method and the heavily fluctuating *DQN Inv* metrics, the behaviour of the interactive agent has to be classified. After 2000 episodes both actions show zero-drift and a constant standard deviation. However, especially $\mathbb{P}_e(SingleBuy)$ fluctuates between $\approx 0.05 - 0.12$. Another important key result is that the double actions dominate the single actions as the developments of double buy and single sell are similar to double sell and single buy (see A.1).

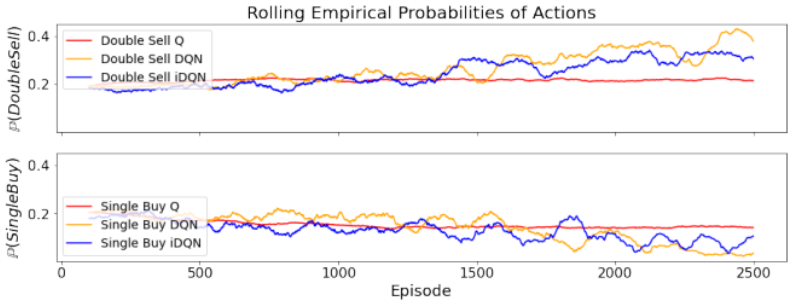

Figure 2.16: Development of action probabilities $\mathbb{P}_e(a)$ for the double sell and single buy action for the three inventory agents.

Q Learning Inv	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	0.3741	0.6114	77.32	0.0048
Episodes 400-500	2.6842	0.5286	78.4	0.0342
Episodes 1400-1500	4.0114	0.6404	74.37	0.0539
Episodes 2000-2100	3.9185	0.5810	74.1	0.0529
Episodes 2400-2500	4.0185	0.5818	74.63	0.0538

DQN Inv	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	-0.0301	0.6246	77.79	-0.0003
Episodes 400-500	1.7701	0.8684	76.62	0.0231
Episodes 1400-1500	3.5457	0.9775	82.18	0.0431
Episodes 2000-2100	4.5491	0.7968	85.91	0.0529
Episodes 2400-2500	4.8312	0.8232	88.68	0.0545

Interactive DQN Inv	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	0.4734	0.6941	75.95	0.0062
Episodes 400-500	2.7844	1.1791	69.37	0.0401
Episodes 1400-1500	4.0997	0.7988	80.06	0.0512
Episodes 2000-2100	4.5318	0.9991	82.78	0.0547
Episodes 2400-2500	4.6422	0.7538	76.66	0.0606

Table 2.2: Statistics Table for the three inventory agents : $\overline{PnL}$ is the average PnL and σ_{PnL} is the standard deviation of the PnLs considering the range of episodes in the first column. An investment is considered to be an exposure changing action.

Table 2.2 underlines the PnL related observations from Figure 2.15 and also sheds some light on the aggressiveness and efficiency/profit per investment of the inventory agents. In contrast to the previous learning task in section 2.1, the *DQN Inv* agent is the most aggressive agent as the probability of an exposure changing action $\mathbb{P}(a \neq n) = 88.68\%$ is over 10% higher when compared to the other two inventory agents. Although the *Q Inv* agent is the only one that has converged after 2500 episodes its average PnL as well as the efficiency/profit per investment are lower than

the metrics of the agents using neural networks. In line with the finding that the *iDQN Inv* agent shows more promising convergence patterns is its higher profit per investment among the two deep Reinforcement Learning agents.

In summary,

- *Action preference:* The double actions dominate the single actions. If the agent detects a sub-space of $\mathcal{S}$ in which buying/selling is profitable on average, it make sense to choose the more extreme action as it results in twice as much return.
- *Convergence:* The *Q Inv* and the *iDQN Inv* agent learn faster than the *DQN Inv* agent. The *Q Inv* agent converges after 1500 episodes, *iDQN Inv* shows first signs of convergence after 2000 episodes and *DQN Inv* still changes its policy after 2500 episodes.
- *PnL:* The agents using neural networks beat the *Q Inv* performance after 1750 episodes.
- *Aggressiveness:* The *DQN Inv* agent invests most aggressively.
- *Profitability per trade:* The profit per exposure changing action for the *iDQN Inv* agent is the highest when compared to the *Q Inv* and *DQN Inv* agent.

One of the main findings of this chapter is that in the more complex setting Deep Reinforcement Learning may result in a performance advantage but does not converge as stably as Q Learning does. That includes the policy/investment strategy as well as a higher standard deviation of episodic PnLs for all (interactive) DQN agents. Apart from pushing the current inventory to the maximum exposure, the single actions seem to be redundant. In the next chapter, we start to use noisy, real world data. Given the instability of the DQN agents and the small advantage of extending the actions space, we use tabular methods such as Q Learning and focus on 3 actions (buy, sell, nothing). Additionally, as we experienced at later stages, the training and testing of Deep Q Networks on millions of observations requires immense computational power, which makes the task even more demanding.

Chapter 3

Trading on historical Limit Order Book Data

Before starting to construct Reinforcement Learning agents that trade on high frequency signals, we recommend studying the book by Bilokon et al. [18] to develop a profound understanding of Machine and Reinforcement Learning applications to finance.

After using a simulated mean-reverting signal to train agents, we start focusing on historical high frequency limit order book data. First, we explain the backtesting and agent setup. Then we introduce the main features that are going to form the state space and the intuition why these features might be able to help the agent invest profitably. Finally, we will look at the performance and behaviour of the agents. The findings in the previous chapter will play a vital role when it comes to generating (mean-reverting) features and designing the agents. During the course of this chapter, particularly when evaluating our agents, we aim at comparing the methodology and results to known research in similar areas from Schnaubelt [2] and Spooner et al. [10].

3.1 Agent Design

The design of the agents is very similar to the one in the previous chapter. We will solely focus on tabular methods including Q Learning, SARSA and Eligibility Traces.

3.1.1 Actions, state space and reward

We have seen before that in a rather simple setup with simulated data the agent was not able to identify the difference between selling (buying) and double selling (buying). Hence, we only implement three actions: placing a bid "b" or an ask "s" limit order and doing nothing "n". The quantity of the limit order is 1 for inventory agents and 2 for the agent that is only able to have an exposure of 1 or -1. As all agents start with an inventory of 0, the first investment decision of the "One" agents is of size 1. The main idea behind using limit orders is that if executed we receive a better price and they create less market impact. In addition, investing with limit orders can quickly be transformed into market making and is sometimes rewarded by exchanges with rebates.

The state space $\mathcal{S}$ can be divided into two different sets of information: Market and agent features. The only agent feature that we use is the current position/exposure of the agent. All other features come from LOB data. As before we discretise continuous variables based on their historical distribution into buckets. We choose 7 buckets with the following quantile values, defining the intervals that determine the buckets:

$$b_t^{s_i} := \left[\min M_t^{s_i}, q_{0.1}^{M_t^{s_i}}, q_{0.2}^{M_t^{s_i}}, q_{0.4}^{M_t^{s_i}}, q_{0.6}^{M_t^{s_i}}, q_{0.8}^{M_t^{s_i}}, q_{0.9}^{M_t^{s_i}}, \infty\right] \ , t \in \mathbb{T}$$

$$S_t^i = j+1, \text{ such that } S_t^i \in [b_t^{s^i}[j], b_t^{s^i}[j+1]) \text{ with } b_t^{s^i}[0] = \min M_t^{s^i} \text{ and } t \in \mathbb{T}$$

with the memory $M_t^{s^i}$ of the i-th state variable at time t and q_x^V the $100 * x\%$ quantile of vector/memory V. We set the maximum number of values stored in one memory to 5000. $\mathbb{T}$ represents the set of times at which there exist observations in the data frame. Learning from the results of the previous chapter, we do not include the bucket-value of a state variable at the last time the exposure changed its sign.

The exposure e_t at time t of the agent is not treated as a continuous variable but the transformation from actual exposure to state-variable is very similar. The agent that only allows one long/ one short indicates its current position by multiplying the state with the sign of the exposure (1 for long, -1 for short). When the state is < 0 (> 0) only the actions n and b (s) are available (see (2.1.1)). For example the state $S_t = -117$ means that at time t the agent is short one stock, the first and second LOB features are in bucket 1 whereas the third state variable is in bucket 7. For the inventory agent we define a maximum long/short position of 3 units. The buckets of the exposure variable are equal to the exposure itself:

$$S^a(e_t) := \begin{cases} 3, \, e_t = 3 \\ 2, \, e_t = 2 \\ 1, \, e_t = 1 \\ 0, \, e_t = 0 \\ -1, \, e_t = -1 \\ -2, \, e_t = -2 \\ -3, \, e_t = -3. \end{cases}$$

If the exposure is equal to the (negative of) maximum exposure only actions n and (b) s are available. Otherwise all actions can be chosen. For the inventory agents, instead of the sign we add the position itself to the identifier of a state. Hence, the previously mentioned state of $S_t = -117$ for the one long/short agent, would be transformed into $S_t = -1117$.

The main goal of the agent is to end up with a positive PnL:

$$\begin{aligned} PnL(t) = &\sum_{\tau \leq t} \mathbf{1}_{\{A_\tau = s\}} ALO\left(\tau, a_0(\tau), q_\tau\right) - \mathbb{1}_{\{A_\tau = b\}} BLO\left(\tau, b_0(\tau), q_\tau\right) \\ &+ e_t \left(\mathbf{1}_{\{e_t > 0\}} b_0(t) - \mathbb{1}_{\{e_t < 0\}} a_0(t)\right) \end{aligned} \tag{3.1.1}$$

with A_τ as usual the decision of the agent at time τ. $ALO\left(\tau, a_0(\tau), q_\tau\right)$ represents an executed ask limit order with price $a_0(\tau)$, quantity q_τ that was submitted to the exchange at time τ.

The most naive way of designing the reward function $\mathcal{R}$ is to use the change in PnL. In order to only trade when the agent is very confident to make a profit, we can also introduce a trading penalty. When the agent takes an action, we measure the change in PnL x seconds ($x \in \{1, 10\}$ depending on the setup) after the limit order submission. This delayed change in PnL better extracts the impact of the action as usually there is a significant time gap between limit order submission and execution.

$$R_t(A_t) := PnL(t + x) - PnL(t) \tag{3.1.2}$$

or

$$R_t(A_t) := PnL(t + x) - PnL(t) - \mathbb{1}_{\{A_t \in \{s,b\}\}} * p$$

with p denoting a fixed or dynamic trading penalty. One problem that may occur is that the PnL reflects the whole exposure and not only the isolated decision of the agent. Hence, we introduce another reward function:

$$R_t(A_t) := \begin{cases} 0, \text{ if } A_t = n \\ b_0(t + x) - b_0(t), \text{ if executed within x seconds ; } A_t = b \\ a_0(t) - a_0(t + x), \text{ if executed within x seconds ; } A_t = s \\ b_0(t) - a_0(t), \text{ else} \end{cases} \tag{3.1.3}$$

with $b_0(t + x)/a_0(t + x)$ the best bid/ask price at time t plus x seconds. If the limit order is not executed, a penalty (minus the spread $s = a_0(t) - b_0(t)$) is returned. If an agent with the

isolation reward function (3.1.3) generates a positive reward on average, we automatically know that the agent is profitable. If an order is submitted and executed the reward function simulates the immediate neutralisation of the executed limit order by a market order of the other side.

3.1.2 Backtesting Algorithm

When training Reinforcement Learning agents on high frequency data, the implementation is essential. On the one hand we would like to have a flexible data pipeline, that guarantees the access, storage and manipulation of all the data. On the other hand we deeply care about the efficiency and the short run time of our code.

The backtester that we used first derives a vector of positions, that is then passed on to the execution, that places the orders, figures out the execution and calculates the PnL. The partition of computing the position and execution results in quicker code but is not able to directly feedback the execution information to the agent. The "theoretical" exposure of the agent can differ from the actual inventory due to the uncertainty of being matched when using limit orders. This problem leads to a partially not transparent flow of information that is fed to the agent and might result in inefficient learning.

Algorithm 8: Reinforcement Learning Framework for all agents learning on LOB data

```
1  Load limit order book and perform feature extraction
2  Select features to form state space 𝒮
3  Slice remaining data into y-hourly batches b_1, ..., b_n, y ∈ ℕ
4  Initialize tabular agent including the parameters γ, ϵ, minimum-ϵ, η, α, tabular
5  Set deactivation d = x seconds, x ∈ {1, 10}
6  for batch in b_1, ..., b_n do
7      i=0
8      Initialize state S_{t_i}
9      for observation O_{t_i} in batch do
10         Determine action A_{t_i} based on (soft) greedy policy
11         if A_{t_i} == b or A_{t_i} == s then
12             Deactivate agent for d seconds
13         end
14         Observe next state S_{t_{i+1}}
15         Save triple (S_{t_i}, A_{t_i}, S_{t_{i+1}}) to memory M
16         if i modulo 100 == 0 then
17             ϵ = min(ϵ * η, minimum-ϵ)
18         end
19         i+ = 1
20     end
21     Convert vector of actions into position vector
22     Plug position vector into execution handler
23     Use execution data and PnL to calculate reward vector R
24     i=0
25     for observation O_{t_i} in batch do
26         Take triple (S_{t_i}, A_{t_i}, S_{t_{i+1}}) from M, R_{t_{i+1}} from R
27         Load A_{t_{i+1}} from next triple
28         Update Q(S_{t_i}, A_{t_i}) given update rule (Q Learning, SARSA using learning rate,
             averaging or Eligibility Traces)
29         i+ = 1
30     end
31 end
```

During testing the for loop starting at line 25 is neglected, as the learning of the policy/behaviour will happen during training.

3.1.3 Execution

To develop a better understanding of the agents, this chapter explains the execution. As mentioned before, we use limit orders, which are always submitted at the best bid or ask. All agents automatically cancel orders that have been longer in the LOB than 10 seconds. Additionally, the execution only allows one order in the limit order book. For instance, if an ask limit order is not executed until the next limit order is submitted by the agent, the first ask order is immediately cancelled and replaced by the second order. We will try to avoid this effect by deactivating the agent for x seconds. The execution is flexible with respect to where in the queue of the best bid or ask price the order is placed. In a realistic scenario the order will be appended to the end of the queue. However, in the beginning we will start with a simplified, unrealistic setting and "jump" the queue.

3.2 Feature Extraction from Limit Order Book Data

We use LOB data from the Chicago Mercantile Exchange (CME) of the commodity future *WTI Crude Oil* with physical delivery in April 2021. We have one observation of bid/ask prices and volume every time a market order has entered the exchange or the LOB has changed due to cancellations or new submissions. Theoretically, we can access the first 12 bid and ask prices. Most of our signals will use only the first and second layers of prices and volumes per market side.

Especially for tabular methods there exists the trade-off between adding new features to the environment to give the agent additional information and the curse of dimensionality. For instance, imagine we have 4 market variables with 7 buckets each and the agents exposure with 7 different exposure buckets. This multiplies to $7^5 = 16,807$ states and $7^5 * 3 = 50,421$ state–action pairs for the agent to learn. Imagine we add another feature to the environment. Then we end up with $7^6 = 117,649$ states and $7^6 * 3 = 352,947$ state–action pairs.

In the case of simulated data in Chapter 2, we saw that the agents converged after 2500 episodes. Each episode had 100 observations. In total the agents received $250,000$ states. Taking into account that the agents consist of ≈ 200 state–action pairs, each state was visited on average 1250 times. Continuing the example from above, we would expect the agent with 4 market and one agent variable(s) to converge after 63 Million observations, whereas the agent using 5 market and one agent variable(s) is expected to receive 441 Million states before convergence. Assuming that we receive 1 million observations per week, we would need over one year of high frequency data for the "smaller" agent and 8 and a half years for the "larger" agent to converge on expectation.

Given that these calculations are based on a noise-free, simulated environment and assume that every observation is used, it becomes clear that the agents trading the limit order book might need even more data to succeed. Hence, feature extraction and selection is critical.

In order to leverage the success from the chapter before, we mainly use mean-reverting signals. However, this time the reward function will not be the signal reward but first and foremost connected to the change in PnL or the isolated reward function from (3.1.3). We group the signals into four different groups: price momentum signals, trade momentum signals from market orders, trade momentum signals from limit orders and risk signals. These four groups of features can mostly be even further decomposed into long, mid and short term signals.

In the following we will denote $a_i(t)/b_i(t)$ as the i-th best ask/bid price and $v_{a_i}(t)/v_{b_i}(t)$ as the volume at the i-th best ask/bid price. The mid-price at time t is defined as shown in (1.3.1). In order to measure price momentum, we apply a simple moving average to the mid-price over two different backward looking time windows. One *long* and one *short* time window. Finally, we substract the short term from the long term moving average. We denote this feature as $MAVGD(x, y, t)$ for $x(y)$ denoting the long (short) time window in t time units.

Figure 3.1 shows two differences of moving averages. The left plot is based on time windows of multiple hours and the right plot is based on time windows of multiple minutes. Clearly, both signals fluctuate around 0. However, their mean-reverting speed is quite different. Unsurprisingly,

the signal based on short term windows reverts at a much higher speed. This also guarantees that all state combinations of buckets are going to be explored as the correlation seems to be close to 0. Generally speaking, one assumes positive momentum when the signal changes from positive to negative as the short term average jumps above the long term average.

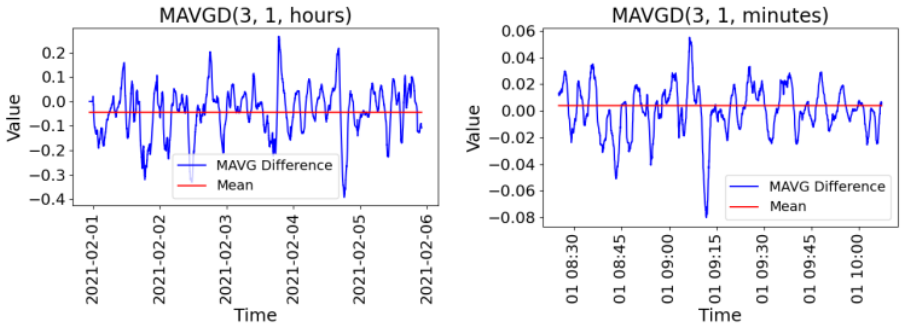

Figure 3.1: Price momentum long and short term mean-reverting signals using the difference of two moving averages based on different time windows. *Left:* 3 hours - 1 hour. The graphic presents data from 1^{st} to 6-th February 2021. *Right:* 3 minutes - 1 minute. The graphic presents data from 1^{st} February 2021 from $\approx 8:30am - 10:00am$.

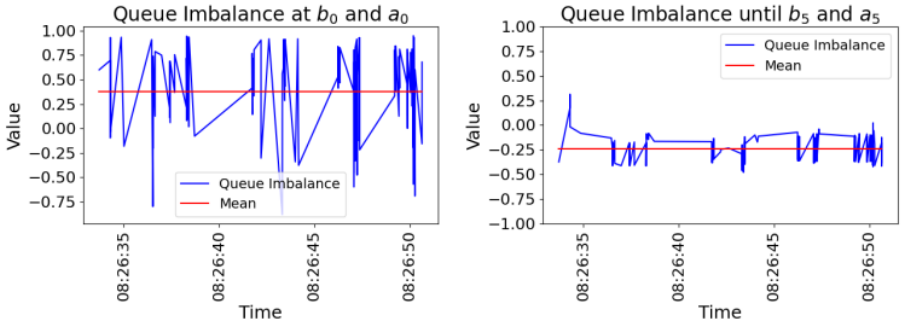

Figure 3.2: Trading momentum signals based on the LOB : *Left:* Queue imbalance using only the volume at the best bid and ask price. *Right:* Queue imbalance using the first five queues of limit order volume of bid and ask prices. Both plots presents data from 1^{st} February 2021 from $\approx 8:26:35am - 8:26:50am$.

Next, we focus on trading momentum signals that use information of submitted limit orders to the LOB. The probably most famous example would be the queue imbalance, which focuses on cumulative volume of bid and ask prices up to the i-th best bid/ask price. We define the metric as done by [2]:

$$QI_i(t) := \frac{\sum_{j=0}^{i} v_{a_j}(t) - v_{b_j}(t)}{\sum_{j=0}^{i} v_{a_j}(t) + v_{b_j}(t)}.$$

The intuition for $i = 0$ would be that values close to -1 (1 respectively) indicate that the mid-price shifts up (resp. down) as the best ask moves up (resp. best bid moves down) due to a small volume in the queue. Over multiple layers of limit order volume the metric focuses on the depth of the LOB and emphasises the price movement in the longer run. Figure 3.2 illustrates the queue imbalance at the best bid and ask prices (left) and up to the 6-th layer (right). Interestingly, both signals are distinguished by large spikes. Although both signals are calculated based on the

same data, they oscillate in different regions of the value spectrum. Whereas QI_0 is mostly positive (stronger ask queue), $QI_5 < 0$ almost the whole 15 seconds, which represents more liquidity on the bid side of the LOB. In addition, the signals including several layers of volume seems to be more stable in terms of fluctuation.

After extracting information using limit orders, we now focus on market orders. In order to measure trading momentum from a market order perspective, we compute trade count and trade volume imbalances as shown in [2]. Trade count imbalances focuse on the number of buy and sell trades during a backward looking time window and trade volume imbalances utilise the cumulative buy and sell volume over the last seconds, minutes or hours. The trade count imbalance at time t looking back x minutes is defined as:

$$TCI(x,t) := \frac{\#SMO(x,t) - \#BMO(x,t)}{\#SMO(x,t) + \#BMO(x,t)}$$

with $\#BMO(x,t)$ ($\#SMO(x,t)$) denoting the number of buy (sell) market orders during the last x minutes looking back from time t. Similarly, $v_{BMO(x,t)}$ ($v_{SMO(x,t)}$) describes the aggregated buy (sell) market order volume looking back x minutes from time t. Using the exact same operation (difference divided by sum) we define the trade volume imbalance at time t looking back x minutes:

$$TVI(x,t) := \frac{v_{SMO(x,t)} - v_{BMO(x,t)}}{v_{SMO(x,t)} + v_{BMO(x,t)}}$$

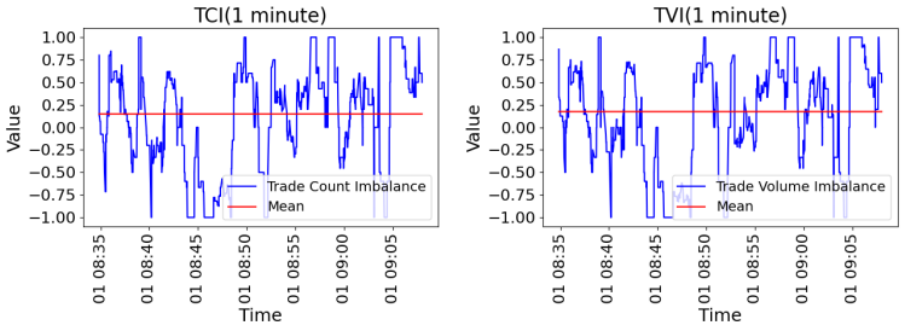

Figure 3.3: Trading momentum signals based on market order trading flow: *Left:* Trade count imbalance of last minute. *Right:* Trade volume imbalance of last minute. Both plots use data from the 1^{st} February 2021 from $\approx 8:35am - 9:05am$.

If the market is dominated by buyers (sellers), the metrics are close to -1 (1), whereas a balanced market with $v_{BMO(x,t)} \approx v_{SMO(x,t)}$ would result in values close to 0. Figure 3.3 depicts $TC(1,t)$ and $TC(1,t)$ with t ranging from 8:25 a.m.- 9:10 a.m. on the 1^{st} February 2021. Both signals look mean-reverting and are hard to separate. One would assume that the correlation between these features is close to one. It can be assumed that both signals provide the agent with the same information. Due to the curse of dimensionality as pointed out earlier we will only include one of those variables.

Lastly, we demonstrate a risk signal and a combination of price and trading momentum component called Kyle's λ. We compute the volatility as the standard deviation of mid-price returns during the last minute. The returns are calculated with two consecutive observations from the LOB data frame. The volatility measures the oscillation of returns from its mean. If this is large, prices have been fluctuating a lot, which can be identified with more uncertainty but also more opportunities when it comes to trading. The formula for Kyle's λ at time t with window W is defined in [3]:

$$\lambda(W,t) := \frac{p_t - p_{t-W}}{\sum_{i=t-W}^{t} b_i V_i}$$

with p_t the mid-price at time t, V_t the traded volume at time t and $b_t = sign(p_t - p_{t-1})$. Kyle's λ is harder to interpret as there are two different factors at play. The numerator measures price momentum over a lookback time window W in seconds. The denominator focuses on trading momentum from markets orders. The larger the denominator, the more price changes and trades have been taking place simultaneously into the same direction. Hence, values close to 0 indicate a buy/sell dominated market whereas larger values suggest a balanced market environment. Furthermore, Kyle's λ is an indicator for the depth of the LOB. Both signals in Figure 3.4 demonstrate mean-reverting behaviour at different speed.

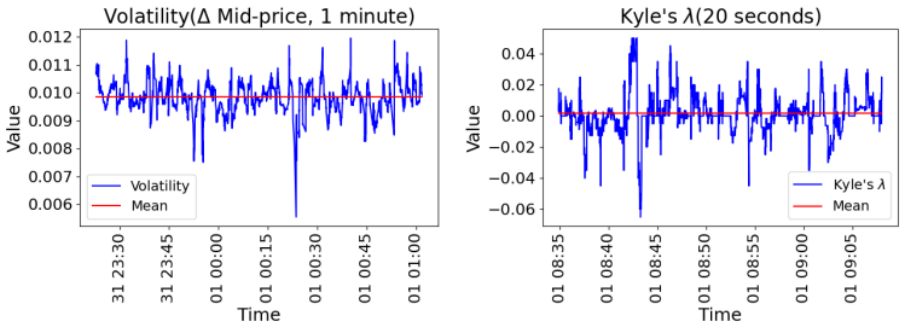

Figure 3.4: *Left:* Volatility of mid-price returns using the last minute of observations. The graphic presents data starting on the 31^{st} January 2021 at 23:30 to 1^{st} of February $1:00am$. *Right:* Kyle's λ. The graphic presents data from 1^{st} February 2021 from $\approx 8:35am - 9:05am$.

Finally, lets have a look at the correlation of the variables shown so far. As mentioned before, we want to avoid a high positive or negative correlation between variables as the agent will effectively explore less states and learn less situations to optimally overcome. Figure 3.5 shows that all features have a correlation close to 0 apart from the trade count and trade volume imbalance.

Apart from Kyle's λ, all previously introduced signals are constructed in a very simple fashion. By looking into recent literature (e.g. [3]) we find that also Rolls' Measure has been used to extract information from LOB data:

$$
\begin{aligned}
R_t &:= 2\sqrt{|cov(\Delta P_t, P_{t-1}|} \\
\Delta P_t &:= [\Delta p_{t-W}, \Delta p_{t-W+1}, \dots, \Delta p_t] \\
\Delta P_{t-1} &= [\Delta p_{t-W-1}, \Delta p_{t-W}, \dots, \Delta p_{t-1}]
\end{aligned}
$$

with Δp_t being the change in mid-price $p_t - p_{t-1}$ and W a lookback window. *Roll's impact* a time t further scales by $p_t * V_t$, where V_t is defined as the cumulative volume executed during a short time interval.

3.3 Agent Evaluation

In this chapter we will present details and results regarding the setup and performance of the numerous agents. First, we have a superficial look at the data that all agents will train and test on. As previously mentioned, we use high frequency limit order book data from a *WTI Crude Oil* future. The future is traded in units of 1000 barrels, whereas the price reflects the price of one barrel. This means that all our results are actually in thousand of £. Moreover, in the execution we do not assume costs or rebates for submitting limit orders.

Unfortunately we only have one month (February 2021) of data available. This results in roughly ≈ 3.8 million observations. This is a significant difference to other research in this area: Schnaubelt [2] uses 18 month of crypto-currency limit order book data and Spooner et al. [10] feed

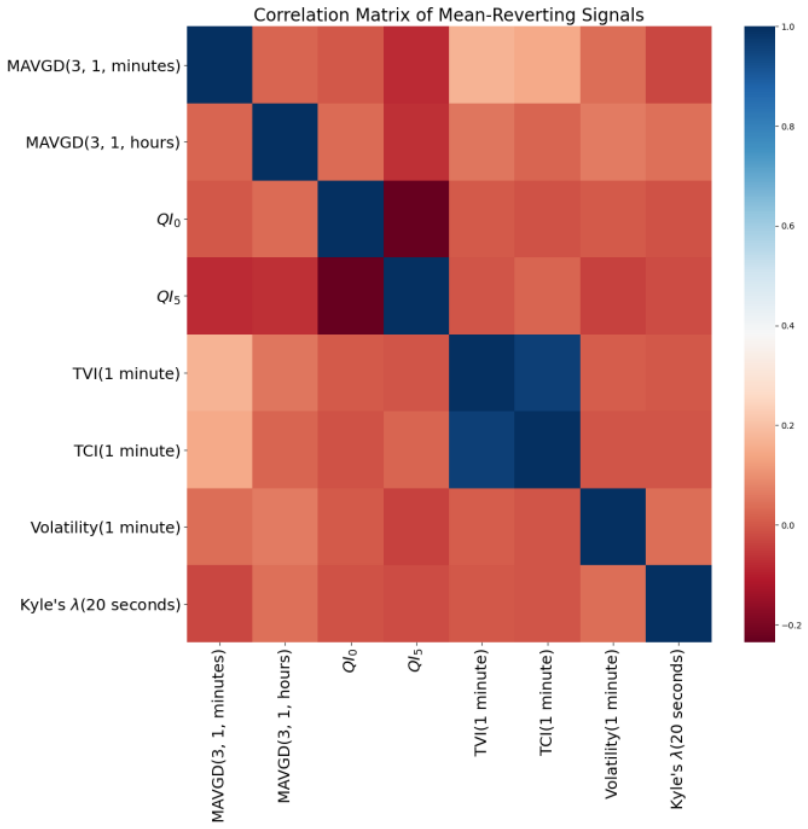

Figure 3.5: Correlation matrix of features based on data from the 1^{st} February 2021 of the *WTI Crude Oil* future.

8 month of LOB data to the agents. As a consequence, it is much more likely that their agents have converged, whereas in the following work we hope for first indications that optimal behaviour has started to be learned. The training and testing periods are defined as follows:

Type	Start	End
Training	01.02.2021	19.02.2021
Testing	22.02.2021	26.02.2021

Table 3.1: Training and Testing period.

As one can observe from Figure 3.6, during the first two weeks of the training data, the traded volume as well as the volatility was much lower than in the third week. Unfortunately, it seems that the 3 weeks of data mostly contain an upward trend. For the agent it is important to explore all different kinds of scenarios to develop a robust policy. From a trading volume point of view, the test data is very similar to the last week of the training data.

During training the exploration parameter ϵ starts at $1(100\%)$ and decays to a minimum of $0.2(20\%)$. Every 100 states that the agent receives, ϵ decays to $0.999 \cdot \epsilon$. The learning rate is set to $\alpha = 0.01$. If the agents are exposed to out of sample test data, they act greedily. The greedy policy is defined by $\pi(s) := \arg\max_{a \in A(s)} Q_\pi(s, a)$ for every state $s \in \mathcal{S}$ that the agent receives. Let us define two parameter sets that are going to appear repeatedly.

The idea behind both selection of features and the discount factor γ is that the *PnL* setup (see

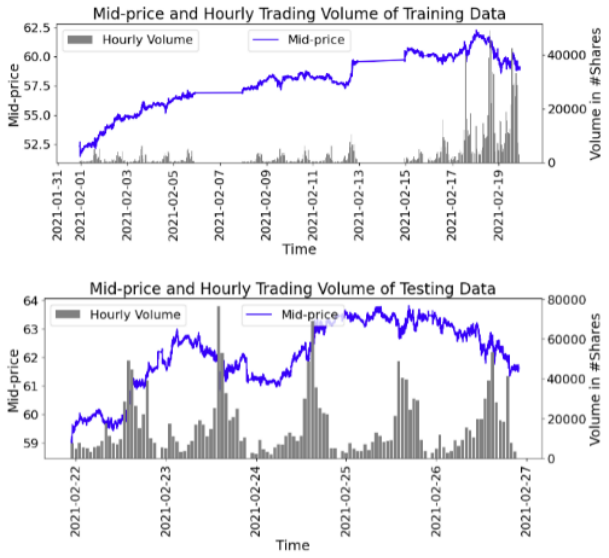

Figure 3.6: Mid-price process and executed hourly trading volume of the *WTI Crude Oil* future. *Upper:* Training data ranging from 1^{st} to 19-th of February 2021. *Lower:* Out of sample test data ranging from $22nd$ to 26-th of February 2021.

Reward function	s^1	s^2	s^3	s^4	γ
PnL	QI_5	TVI(1 minute)	MAVGD(3, 1, minutes)	MAVGD(3, 1, hours)	0.95
Iso	QI_0	TVI(1 minute)	MAVGD(3, 1, minutes)	Volatility(1 minute)	0.1

Table 3.2: Parameter sets for agents

equation (3.1.2)) focuses on a long-sighted return and parameters, whereas the *Iso* setting (see equation (3.1.3)) tries to extract short term opportunities combined with signals that describe a more immediate, quickly evolving picture of the trading environment.

In order to extract the isolated effect of an investment decision on the PnL, we deactivate the agent for 1 second after each submitted limit order. The orders are cancelled after 10 seconds.

3.3.1 Jump the queue

The fist Q Learning agents are able to jump the queue. This means that submitted limit orders are allocated to the front and not to the end of the volume queue at the best bid or ask price. This unrealistic modification simplifies the trading environment and hence the task at hand to learn for the agent. We do this based on previous experience, that small steps towards a realistic setting provide a lot of useful findings regarding which information or setup fits the agent best. The Q agents are only allowed to be long or short one unit of the financial asset, whereas the suffix *Inv* (e.g. *Q Inv*) means that the inventory process of the agent moves within $[-3, 3]$. In general, with training we refer to learning and updating Q-values, which are transformed into a greedy policy in the out of sample test.

Figure 3.7 shows the training and testing performance of 4 agents that jump the queue. Clearly, all agents need roughly two weeks to learn a strategy that is profitable depsite the minimum exploration of 20%. The out of sample testing performance is consistently profitable.

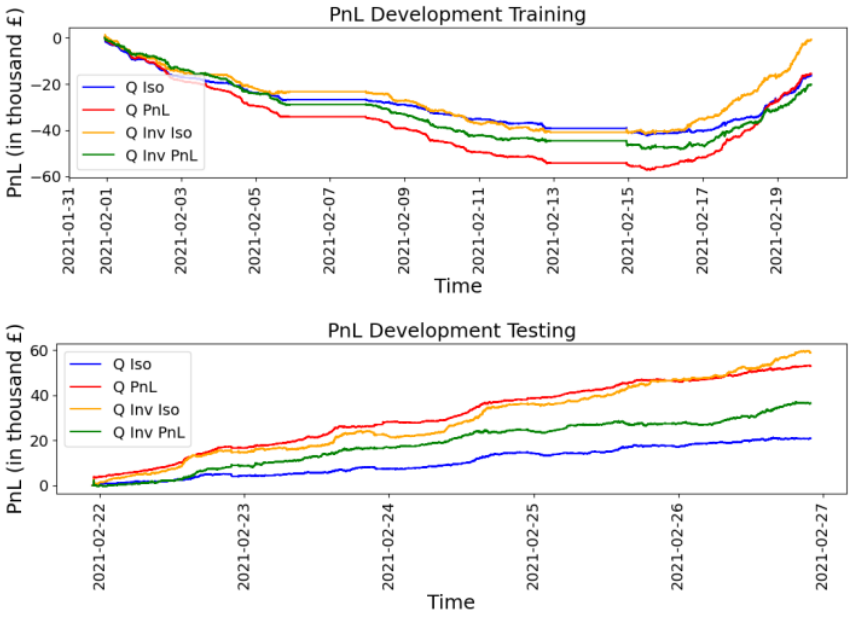

Figure 3.7: Training and testing performance (3.1.1) of Q agents with and without inventory. All submitted limit orders jump the queue at the best bid or ask price.

Table 3.3 and 3.4 indicate that the *Iso* agents are less aggressive, meaning they submit less orders. They are also better at timing their limit order submission, as the filling probability is higher in training and testing. This is not a surprise as the reward function punishes not executed limit orders, which is completely neglected in the *PnL* setup. Moreover, the aggressive behaviour of the *PnL* agents leads to a lower number of visited states, as order submission deactivates the agent for 1 second, which automatically skips observations that could have been fed to the agent.

Especially for the inventory agents, the number of state–action pairs that have never been visited is very high (more than 10% of all $Q(s, a)$). In addition, not a single state–action pair has been visited more than 500 times. In summay, apart from the stable performance, the convergence metrics are not satisfying.

Agent	Set	#V	#V/#states	#No V	$\mathbb{P}(> 500)$	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$
Q	Iso	1,045,621	108.87	4	1.9 %	17.57 %	23.68 %
Q	PnL	570,847	59.44	131	0.74%	52.14 %	18.76 %
Q Inv	Iso	868,430	17.22	8590	0.0%	25.91 %	25.81%
Q Inv	PnL	546,724	10.84	9437	0.0%	56.00 %	21.76%

Table 3.3: Training evaluation for Q Learning agents with and without inventory taking into account two parameter sets. All submitted limit orders jump the queue at the best bid or ask price. #V denotes the total number of visits aggregated over all states. # No V is the number of state–action pairs that have not been visited during training. $\mathbb{P}(> 500)$ is the number of state–action pairs that have been visited more than 500 times divided by the total number of state–action pairs. $\mathbb{P}(Invest)$ reflects the number of "buy" and "sell" decisions divided by the total number of states that the agent received (# V). $\mathbb{P}(Fill)$ is computed by dividing the number of executed by the number of submitted limit orders.

Agent	Set	Avg. profit/day (+- std. deviation)	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$
Q	PnL	4.21 (+-1.70)	16.35%	26.44 %
Q	Iso	10.69 (+-3.04)	51.23%	24.87%
Q Inv	PnL	11.88 (+-2.74)	24.66 %	29.90%
Q Inv	Iso	7.34 (+-2.51)	55.24%	27.01%

Table 3.4: Test evaluation for Q Learning agents with and without inventory taking into account two parameter sets. All submitted limit orders jump the queue at the best bid or ask price. For $\mathbb{P}()$ explanation see Table 3.3.

3.3.2 Middle of the queue

Next, we modify the queue parameter from 0 to 0.5. This means that the agents place the order in the middle of the volume queue. Taking into account that orders of other market participants are frequently cancelled, this setting is still optimistic but certainly closer to reality. Due to a worse position in the queue, we would like to allow a longer deactivation time for the agents when submitting limit orders. This will better extract the isolated impact of the decision on the profitability of the agent. Unfortunately, deactivating for a longer time means that more observations in the data frame are going to be skipped. This results in less exploration. We decided to set the deactivation time to 10 seconds.

We start with the Q agent allowing an exposure of 1 and -1, which has 9604 state–action pairs. This is a significant difference to the *Inv* agents, which need to explore ≈ 50.000 states–action pairs. As a results, all Q Learning agents showed only little signs of optimal behaviour and non-positive out of sample performance. One problem of Q Learning is known to be its optimistic estimation of the Q function by using the maximum Q-value of the next state. To reduce the overestimation of $Q(s, a)$, Moskovitz et al. [16] suggests to use a pessimistic updating rule by replacing the maximum with the minimum.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \min_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

This will probably lead to a pessimistic estimation of $Q(s, a)$. In our work we will follow the findings from Spooner et al. [10]. They claim that on-policy methods, such as SARSA, develop a more realistic approximation of the Q function.

Agent	Set	#V	#V/#states	#No V	$\mathbb{P}(> 500)$	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$
Q	Iso	506,502	52.74	70	0.0%	13.05%	20.17%
SARSA	Iso	477,519	49.72	38	0.0%	14.21%	21.61 %
Q	PnL	176,462	18.37	404	0.0%	48.87%	20.41 %
SARSA	PnL	177,279	18.46	389	0.0%	48.71%	20.20%

Table 3.5: Training evaluation for Q Learning and SARSA agents without inventory taking into account two parameter sets. All submitted limit orders are placed in the middle of the queue at the best bid or ask price. For $\mathbb{P}()$/ #V explanations see Table 3.3.

Agent	Set	Avg. profit/day (+- std. deviation)	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$
Q	Iso	-0.2 (+-1.83)	0.13%	21.07 %
SARSA	Iso	0.28 (+-0.92)	0.95%	24.26%
Q	PnL	-2.23 (+-0.87)	37.98%	23.17%
SARSA	PnL	-2.13 (+-1.63)	35.98%	22.93%

Table 3.6: Test evaluation for Q Learning agents with and without inventory taking into account two parameter sets. All submitted limit orders are placed in the middle of the queue at the best bid or ask price. For $\mathbb{P}()$ explanation see Table 3.3.

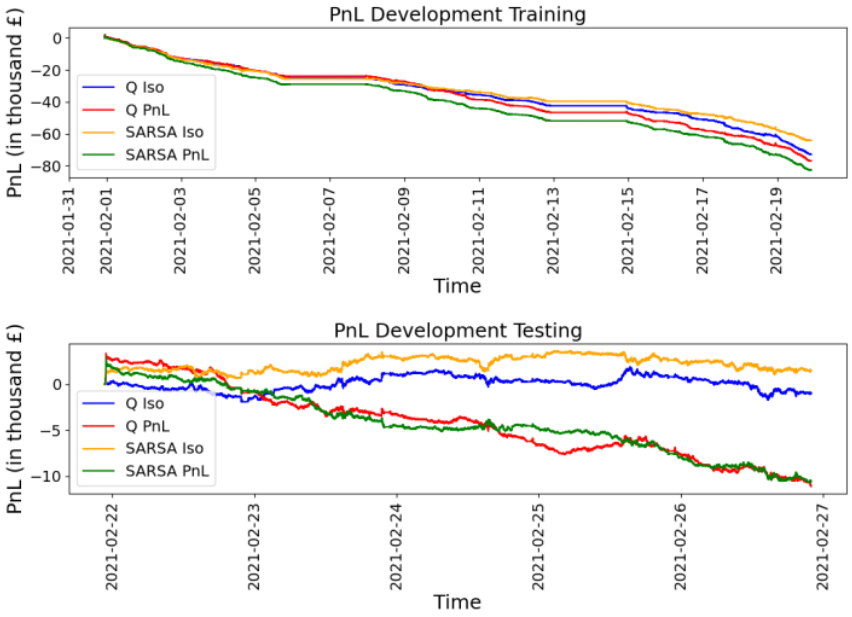

Figure 3.8: Training and testing performance (3.1.1) of Q and SARSA agents without inventory. All submitted limit orders enter the queue in the middle.

Figure 3.8 shows that during training all agents fail to derive a profit at any time. Interestingly, the out of sample test reveals a performance gap between the agents using the *Iso* and *PnL* setup. Apart from different information that is fed to the agents, this may be explained by the number of states, that each agent has explored. In general, all agents explored less than before due to the longer deactivation time (Table 3.5). It is reasonable to assume that this slows down the convergence of the agents. As before, the *PnL* agents are more aggressive and visit even less state–action pairs. Moreover, the change in the queue parameter significantly reduces the limit order submission of the *Iso* agents (Table 3.6). This may be due to the increased uncertainty about execution (as we are further behind in the queue) and the penalty for non-execution. In contrast to the *PnL* setup, the *Iso* agents show first signs of optimal behaviour. Especially the *SARSA Iso* agent invests profitably, which confirms the findings of Spooner et al. [10]. Furthermore, we tried averaging rewards instead of using the error term and learning rate (see (1.1.7),(1.1.8)). That has also been suggested in [10]. The key metrics and performance did not significantly change.

At this point we try to evaluate, how the so far most successful *SARSA Iso* agent in the more realistic scenario (queue = 0.5) transforms the information into a decision. Figure 3.9 illustrates the distribution per decision given either the queue order imbalance (left) or the moving average difference (right). Clearly, by also considering the distributions of the other variables in Figure A.2 (see Appendix), we come to the conclusion that QI_0 dominates the agents decision making. In detail, the agent submits buy limit orders when the queue imbalance is close to -1 and prefers to submit ask limit orders for QI_0 close to 1. This corresponds to the intuitive interpretation of the queue order imbalance at the best bid and ask prices. Additionally, this aligns with Schnaubelt's [2] findings. He highlights the predictive power of queue order imbalances. For MAVGD(3, 1, minutes), TVI(1 minute) and Volatility(1 minute), the violin plots of "Nothing","Buy" and "Sell" look very similar. It appears that the discretised bucket value of these variables do not contain enough information to drive the agent's decision making.

Recalling the visualisation of QI_0 in Figure 3.2, one can observe that the signal spikes rapidly and jumps around its mean multiple times within seconds. This quick reaction to the market

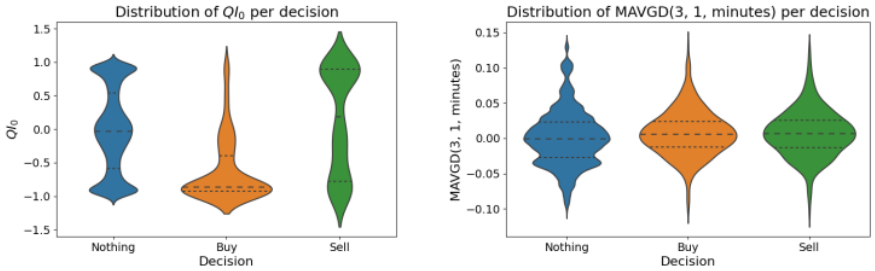

Figure 3.9: Distribution for queue order imbalance (QI_0) and moving average difference (3 minutes -1 minute) per decision based on the *SARSA Iso* agent's test run.

environment significantly separates QI_0 from other signals, which behave in a more controlled way. As this seems to be the powerful type of indicator, we try to extract other signals with similar properties.

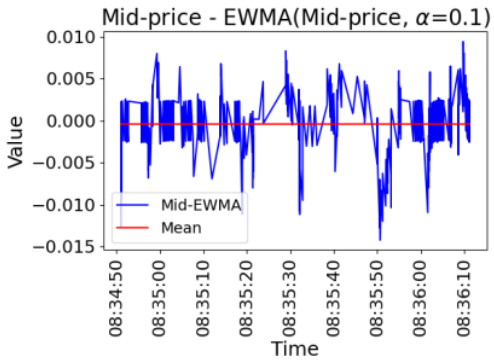

Figure 3.10: Price momentum, short term mean-reverting signal using the difference of the mid-price and an exponentially weighted moving average with $\alpha = 0.1$. The graphic presents data from 1^{st} February 2021 from $\approx 8:34:50am - 8:36:10am$.

Figure 3.10 shows the difference of the mid-price and an exponentially weighted moving average of the mid-price, which depends on the parameter $\alpha = 0.1$. Given a time series $X_0, X_1, X_2, ...$ the exponential moving average is given by:

$$EWMA(X_t, \alpha) = \alpha \sum_{i=0}^{t} X_i (1-\alpha)^{t-i}.$$

Due to the previous findings we will change the setup in the following way: We include the signal shown in Figure 3.10 and exchange it with the moving average difference parameter MAVGD(3, 1, minute). In order to be less attracted to large orders, we change from TVI(1 minute) to TCI(2 minutes). As we would like achieve better convergence metrics with the same data, we neglect the risk parameter and use only 3 LOB features. Additionally, previous *Iso* agents have shown very careful submission behaviour. This might be due to the pure punishment of not executed orders, which automatically reduces the state–action values for the investment decisions. As we aim for

agents, which quote on a more frequent basis, we slightly modify the reward function:

$$R_t(A_t) := \begin{cases} 0, \text{ if } a_t = n \\ b_0(t+x) - b_0(t), \text{ if executed within x seconds ; } A_t = b \\ a_0(t) - a_0(t+x), \text{ if executed within x seconds ; } A_t = s \\ b_0(t+x) - a_0(t), \text{ if not executed within x seconds ; } A_t = b \\ b_0(t) - a_0(t+x), \text{ if not executed within x seconds ; } A_t = s. \end{cases} \tag{3.3.1}$$

Instead of purely punishing non-executed limit orders after 10 seconds, we first calculate the same reward as if it would have been executed and then subtract the spread at time t. This results in the simplified expressions shown in (3.3.1). We call this reward function *Adjusted Iso*.

Reward function	s^1	s^2	s^3	γ
Adjusted Iso	QI_0	TCI(2 minute)	Mid-price - EWMA(Mid-price, $\alpha = 0.1$)	0.1
Complete Iso	QI_0	TCI(2 minute)	Mid-price - EWMA(Mid-price, $\alpha = 0.1$)	0.1

Table 3.7: Adjusted Iso (3.3.1) and Complete Iso (3.3.2) parameter sets using the Adjusted Iso and Conplete Iso reward functions and 3 LOB features.

Agent	Set	#V	#V/#states	#No V	$\mathbb{P}(> 500)$	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$
Q	Adj. Iso	515,232	375.53	0	27.4%	12.66%	21.99%
SARSA	Adj. Iso	515,609	375.80	0	26.89%	12.66%	21.68%
Q Inv	Adj. Iso	415,685	57.71	70	0.71%	17.14%	26.75%
SARSA Inv	Adj. Iso	408,352	56.69	2	0.65%	17.55%	27.22%
Q Inv ET	Comp. Iso	220,848	30.66	28	0.53%	38.24%	23.34%

Table 3.8: Training evaluation for Q Learning (ET) and SARSA agents with and without inventory using the *Adjusted Iso* and the *Complete Iso* parameter set. All submitted limit orders are placed in the middle of the queue at the best bid or ask price. For $\mathbb{P}()$/ #V explanations see Table 3.3.

Agent	Set	Avg. profit/day (+- std. deviation)	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$
Q	Adj. Iso	3.02 (+-0.41)	1.76%	22.55%
SARSA	Adj. Iso	1.46 (+-1.18)	1.29%	21.74%
Q Inv	Adj. Iso	-1.95(+-1.04)	3.73%	28.94%
SARSA Inv	Adj. Iso	1.70 (+-4.40)	3.08%	31.23%
Q Inv ET	Comp. Iso	0.84(+-4.11)	6.49%	24.50%

Table 3.9: Test evaluation for Q Learning (ET) and SARSA agents with and without inventory using the *Adjusted Iso* and the *Complete Iso* parameter set. All submitted limit orders are placed in the middle of the queue at the best bid or ask price. For $\mathbb{P}()$ explanation see Table 3.3.

Table 3.8 and Table 3.9 show that both agents with two inventory states $(-1, 1)$ show promising signs of convergence in training and testing. Figure 3.12 in section 3.3.3 is the corresponding performance plot. Due to the reduced dimension in the state space, each state is visited more frequently. This leads to more robust Q-values. Contrary to the previous setup (3.6), the Q agent performs more profitably and more stably than the *SARSA* agent. The change in reward function caused the agent to act more aggressively as we hoped to incentivise. A look at Figure 3.11 illustrates a clear decision pattern for QI_0 and Mid-price - EWMA(Mid-price, α=0.1). Hence, the modified information appears to help the agent to exploit data patterns in a more reliable and consistent fashion. Interestingly, extremely positive (negative) values of the queue order imbalance are combined with slightly positive (negative) values of the new price momentum parameter. The distributions of the trade count imbalance for buying and selling look almost identical (A.3). However, there is a visible difference between the distributions of holding and changing the exposure.

Due to the larger amount of states, the inventory agents show weaker signs of convergence in all parameters and also do not convince with reliable positive performance (Table 3.8, Table

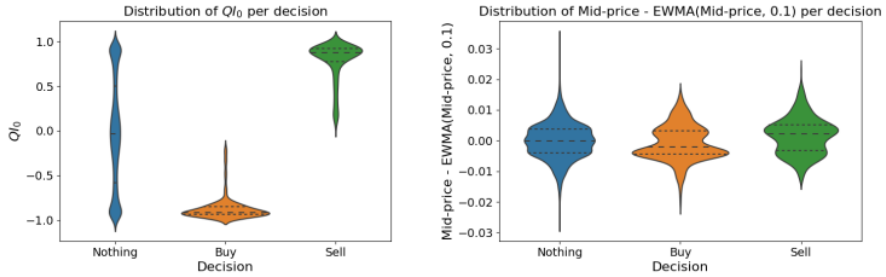

Figure 3.11: Distribution for queue imbalance (QI_0) and the new price momentum signal (Mid-price - EWMA(Mid-price, $\alpha = 0.1$) per decision based on the *Q Adjusted Iso* agent's test run.

3.9). However, the inventory agents develop an efficient order submitting behaviour as their filling probability is higher.

So far we assign a value of 0 to all actions "n", independent of the state or LOB development. As the inventory agents react more sensible to changes in prices and need longer to revert their position, we specify a new reward function called *Complete Iso* with η_t denoting the exposure at time t. The parameter η_t can be interpreted as a risk indicator as it may reward but also punish high exposures.

$$R_t(A_t) := \begin{cases} b_0(t+x) - b_0(t), \text{ if executed within x seconds ; } A_t = b \\ a_0(t) - a_0(t+x), \text{ if executed within x seconds ; } A_t = s \\ b_0(t+x) - a_0(t), \text{ if not executed within x seconds ; } A_t = b \\ b_0(t) - a_0(t+x), \text{ if not executed within x seconds ; } A_t = s \\ \eta_t(b_0(t+dt) - b_0(t)), \text{ if } A_t = n \text{ and } \eta_t < 0 \\ \eta_t(a_0(t+dt) - a_0(t)), \text{ if } A_t = n \text{ and } \eta_t > 0 \\ 0, \text{ otherwise} \end{cases} \tag{3.3.2}$$

with $t + dt$ indicating the next observation in the data frame after t and $a_0(t + x)$ the best ask price x seconds after time t.

The goal is to use this reward function in combination with Eligibility Traces (ET) to stabilise the performance. The strength of ETs is the backward impact of future rewards on recently visited state–action pairs. To leverage this property even more, we select a high discount factor $\gamma = 0.95$ and ET-parameter $\lambda = 0.95$.

The *Q Inv ET* agent using the *Complete Iso* reward function acts more aggressively than the *Q Inv* agent (Table 3.9). Again, due to the deactivation time this leads to less exploration. Nevertheless, with half of the exploration the performance of *Q Inv ET* is positive but very unstable. This indicates that the agent has started to learn optimal behaviour but requires more training to reduce PnL oscillations.

3.3.3 End of the queue

In order to investigate the robustness of the so far most promising strategy, we test the learned tabular of Q-values from the *Q Adj. Iso* agent with a queue parameter of 1.0. This means we enter the LOB at the end of the queue. Considering limit order cancellations, this scenario is realistic if not even more difficult to master than reality would be. Waiting further behind in the queue also means longer waiting times until execution. Hence, we expect worse filling probabilities. We intentionally do not increase the deactivation or limit order cancellation time. As the state of the LOB constantly changes, we assume that increased deactivation could result in non-optimal investment decisions.

Table 3.10 shows that when further increasing the queue parameter there is a performance drop, but impressively the behaviour learned in the "easier" trading environment is still successful in a

more complicated setup. As expected, the filling probability and the number of executed trades decrease with increasing queue parameter. The average number of matched limit orders per day decreases from ≈ 550 to 475. This means that by increasing the queue parameter from 0.5 to 1.0, $\approx 85\%$ of the previously executed limit orders were still executed. This emphasises the timing ability of the agent. In 3.12 we compare the performance curves of three previously mentioned agents, that demonstrate promising results considering the amount of data they trained on.

Agent	Set	Queue	Avg. profit/day (+- std. deviation)	$\mathbb{P}(Invest)$	$\mathbb{P}(Fill)$	#E
Q	Adj. Iso	0.5	3.02 (+-0.41)	1.76%	22.55%	2,782
Q	Adj. Iso	1.0	0.92 (+-0.46)	1.76%	19.28%	2,378

Table 3.10: Test evaluation for Q Learning without inventory using the *Adjusted Iso* parameter set. All submitted limit orders are placed at the best bid or ask price. For $\mathbb{P}()$ explanation see Table 3.3. #E denoting the number of executions.

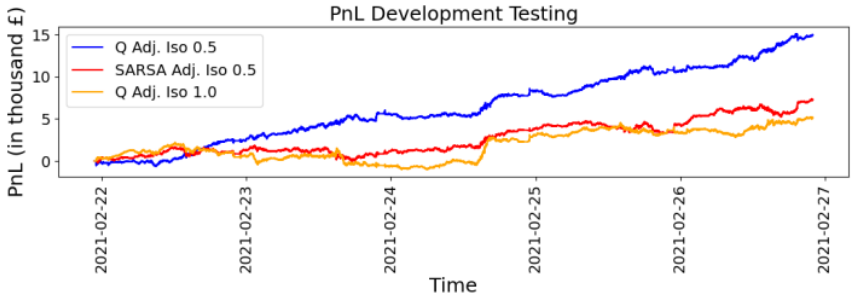

Figure 3.12: Testing performance (3.1.1) of Q and SARSA agents without inventory.

Conclusion

The noticeable recent success of Reinforcement Learning in different fields, naturally suggests its application to financial problems. Chapter 2 shows that tabular methods as well as Deep Q Networks are able to learn optimal behaviour in a simulated environment, given that the reward function transparently reflects the agent's objective. Especially Q Learning agents demonstrated fast convergence and consistent performance when trained and tested on the Ornstein–Uhlenbeck process.

Despite the small number high frequency LOB observations, we were able to design profitable Reinforcement Learning trading agents, that are solely based on submitting limit orders. By taking small steps towards a realistic trading scenario, we gradually optimised the state space and the reward function. As a result, rapidly fluctuating signals seem to convey the most precious information for the agents. Surprisingly, our findings indicate that the change in PnL is not a suitable candidate for the reward function. Other concepts, such as isolating the impact of specific actions and deactivating the agent showed a larger impact on the learning process. The introduction of inventory significantly increases the complexity of the learning task. The higher number of states and the risk of higher exposures require more training and more robust methodologies. Eligibility Traces seem to be a suitable candidate. Our findings partially agree with those mentioned by Spooner et al. [10]. In detail, the application of on-policy methods and risk adjusted reward functions has proven to be a potential improvement, whereas averaging Q-values instead of applying the gradient did not enhance the agent's performance. It should be emphasised that our results were derived from only one financial asset and one month of high-frequency data.

Due to the computational challenges of training and predicting with deep neural networks, in Chapter 3 we were only able to focus on tabular methods. Especially the promising results of the *Interactive* Deep Q Networks in the simulated environment should be applied to real-world financial data in the future to robustly measure its potential. In general, the implementation of Reinforcement Learning agents turned out to be a complex task, that requires not only sufficient knowledge of software (such as `Python`), but also a solid hardware setup.

While we concentrated on sharpening the reward function and selecting only a few traditionally extracted signals, there are many directions that can be explored. For example, the combined usage of limit and market orders or the application of autoencoders to reflect the state of the LOB are possible avenues that are open to explore. Furthermore, based on our findings and the research shown in [10], the design of a *Multiagent* that leverages a combination different agents to form the final Q-values, seems a real prospect.

Lastly, taking into account the flexibility and recent successes of Reinforcement Learning in finance but also in other areas, the use of such methods seems highly promising.

Appendix A

Supplementary Figures

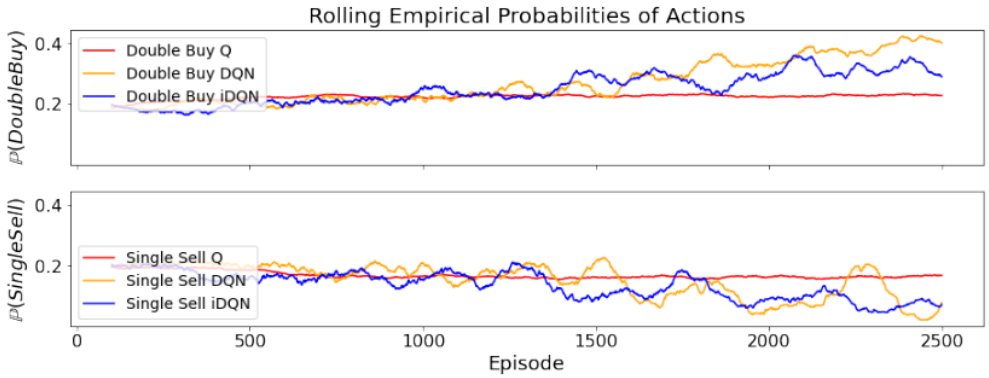

Figure A.1: Development of action probabilities $\mathbb{P}_e(a)$ for the double buy and single sell action for the three inventory agents.

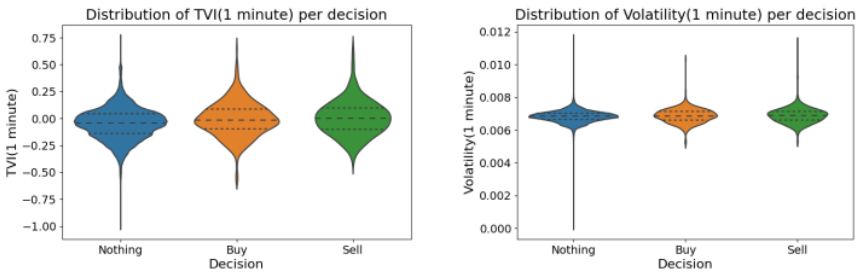

Figure A.2: Distribution for trade volume imbalance (TVI(1minute)) and volatility ($Volatility$(1 minute)) per decision based on the *SARSA Iso* agent's test run.

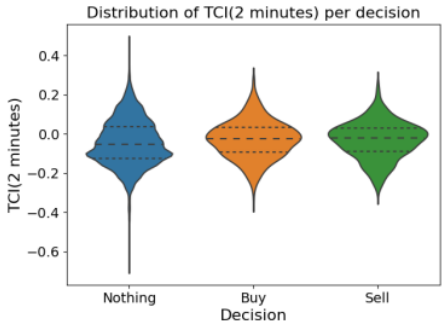

Figure A.3: Distribution for trade count imbalance (TCI(2 minutes))per decision based on the Q *Adj. Iso* agent's test run.

Bibliography

[1] Richard S. Sutton, Andrew G. Barto *Reinforcement Learning: An Introduction*
The MIT Press, Cambridge, Massachusetts

[2] Schnaubelt, Matthias
Deep reinforcement learning for the optimal placement of cryptocurrency limit orders
FAU Discussion Papers in Economics, No. 05/2020, Leibniz-Informationszentrum Wirtschaft

[3] David Easley, Marcos López de Prado, Maureen O'Hara, and Zhibai Zhang
Microstructure in the Machine Age
https://ssrn.com/abstract=3345183, February 2019

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller
Playing Atari with Deep Reinforcement Learning
https://arxiv.org/pdf/1312.5602v1.pdf
DeepMind Technologies, January 2013

[5] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis
Mastering the game of go without human knowledge
https://doi.org/10.1038/nature24270
Nature, October 2017

[6] F. Higham and D. J. Higham (2018)
Deep learning: an introduction for applied mathematicians https://arxiv.org/abs/1801.05894, Section 5, 2018

[7] Alvaro Cartea, Sebastian Jaimungal, Jose Penalva *Algorithmic and High Frequency Trading*
Camebridge University Press, 2015

[8] Petter N. Kolm and Gordon Ritter
Dynamic replication and hedging: A reinforcement learning approach
The Journal of Financial Data Science, 2019

[9] Brian Ning, Franco Ho Ting Lin, and Sebastian Jaimungal
Double deep Q-learning for optimal execution
arXiv, 2018, https://arxiv.org/abs/1812.06600.

[10] Thomas Spooner, John Fearnley, Rahul Savani, Andreas Koukorinis
Market Making via Reinforcement Learning
AAMAS 2018, July 10-15, 2018, Stockholm, Sweden

[11] Thomas Spooner, Rahul Savani
Robust Market Making via Adversarial Reinforcement Learning
https://www.researchgate.net/publication/341104642

[12] Huyen Pham
Continuous-time Stochastic Control and Optimisation with Financial Applications
Springer-Verlag Berlin Heidelberg 2009

[13] I. Goodfellow, Y. Bengio and A. Courville
Deep Learning
MIT Press, Cambridge, MA, 2016

[14] Link: https://medium.com/@srijaneogi31/exploring-multi-class-classification-with-deep-learning-239cb42e69bf, Accessed: 29.08.2021

[15] Cartea, Álvaro and Jaimungal, Sebastian and Sánchez-Betancourt, Leandro
Deep Reinforcement Learning for Algorithmic Trading
March 2021; https://ssrn.com/abstract=3812473

[16] Ted Moskovitz, Jack Parker-Holder, Aldo Pacchiano, Michael Arbel, Michael I. Jordan
Tactical Optimism and Pessimism for Deep Reinforcement Learning
https://arxiv.org/abs/2102.03765v3, May 2021

[17] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, David Silver
Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model
Nature 588, 604–609 (2020), December 2020

[18] Matthew F. Dixon, Igor Halperin, Paul Bilokon
Machine Learning in Finance
Springer, 2020

[19] Tim Leung, Xin Li
Optimal Mean Reversion Trading: Mathematical Analysis and Practical Applications
World Scientific Publishing Co., Doi.org/10.1142/9839, Available at SSRN: https://ssrn.com/abstract=2664588, September 23, 2015

[20] Alexander Zai, Brandon Brown
Deep Reinforcement Learning in Action
Manning, 2020

[21] Jean-Phillipe Bouchaud, Julius Bonart, Jonathan Donier, Martin Gould
Trades, Quotes and Prices; Financial Markets Under the Microscope
Cambridge University Press, 2019

BERTERMANN_ARVID_01912841

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60