Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics and Machine Learning
2020 | LIU-IDA/STAT-A--20/006--SE

Intra-Day Electricity Trading with Reinforcement Learning

Julius Kittler

Supervisor : Jose M. Peña
Examiner : Oleg Sysoev

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

Abstract

This thesis addresses the problem of trading electricity on the German intra-day electricity market. In particular, the continuous hourly intra-day market is considered, where electricity is traded that is to be delivered at a specific hour in the future on the same or next day. Electricity is traded continuously, meaning that orders can be placed at any given time until shortly before the specific hour at which the electricity is delivered to consumers.

The problem of trading electricity is addressed with reinforcement learning (RL). In particular, RL agents are trained to place both, buy and sell orders, with the goal to achieve large profits per traded volume. The RL agents also need to make decisions about the volume they want to buy or sell at each trade. The primary RL method applied in this thesis is a synchronous advantage actor-critic method (A2C) and the secondary RL method is a deep Q-network (DQN). Bayesian optimization is applied to optimize hyperparameters, features and reward functions. Training, validation and evaluation are conducted based on *static* data of historic market orders instead of a *dynamic* market simulation. This means that the actions of the RL agents do not affect the market prices in the context of this thesis.

Mainly, this thesis makes three contributions by a) comparing different *reward* functions, b) comparing different RL *methods* and c) comparing the final RL agent with *baseline* strategies. In regards to the reward functions, it is shown that a prevalent reward function from the financial domain, the Sharpe ratio, is not the most suitable reward function in the electricity domain if the goal is to optimize profits. In regards to RL methods, A2C and DQN resulted in similar behavioral strategies and their performance difference was not found to be significant. In regards to baseline strategies, the final RL agent was shown to outperform different baseline strategies: a random trader, a forecast trader and a moving average trader. However, the RL agent was not able to outperform a simple rule-based algorithm that was reverse engineered to mirror the RL agent's main strategy of selling at sudden price increases.

Overall, this thesis shows that RL is suitable for creating profitable trading agents and for identifying successful trading strategies. Moreover, this thesis also shows that successful trading strategies may be simple in the context of electricity trading, which allows to represent them with rule-based algorithms. Therefore, a possible approach might be to use RL to identify successful trading strategies and to then convert these strategies into rule-based algorithms to reduce complexity and improve interpretability.

Acknowledgments

I would like to thank my supervisor Ludwig Rauch at EXXETA for highly fruitful discussions, feedback and help to prioritize relevant aspects of the thesis. Above all, I would like to thank him for being a great and supportive colleague that was really fun to work with.

Furthermore, I would like to thank my supervisor Jose M. Peña for giving detailed feedback, for pointing me to relevant literature within the field of reinforcement learning and beyond and for inspiring me to combine ideas from different academic research fields. I was impressed by how closely Jose M. Peña followed along with the progress of this thesis.

Last but not least, I would like to thank my examiner Oleg Sysoev not only for his precise feedback from a statistical viewpoint but also for running the outstanding master's program in Statistics and Machine Learning at LiU, which to me was the best possible preparation for this master's thesis. I can hardly express how thankful I am for these 2 years full of challenges.

Contents

List of Figures

List of Tables

Acronyms

RL reinforcement learning

MDP Markov decision process

TD temporal difference

MC Monte Carlo

NN neural network

A3C asynchronous advantage-actor critic

A2C synchronous advantage-actor critic

DQN deep Q-network

OTE optimal trade execution

EPF electricity price forecasting

Reinforcement Learning Symbols

t discrete timestep

T final timestep of an episode

A_t a random variable representing the action at time t

S_t a random variable representing the state at time t

R_t a random variable representing the reward at time t

G_t a random variable representing the return following time t (until the end of the episode)

a an action (a value of the random variable A_t)

s, s' a state and it's subsequent next state (values of the random variables S_t and S_{t+1})

r a reward (a value of the random variable R_{t+1})

π a policy (a decision-making rule)

$\pi(s)$ action taken in state s under *deterministic* policy π

$\pi(a|s)$ probability of taking action a in state s under *stochastic* policy π

Trading Symbols

V_t total volume traded at time t (by all market participants)

V_t^b volume bought by the agent at time t

V_t^s volume sold by the agent at time t

P_t market price per volume at time t

C_t total cash of the agent at time t

Q_t total inventory, i.e. quantity of volume, owned by the agent at time t

$\mathcal{E}_t$ total equity of the agent at time t ($C_t + P_t \cdot Q_t$)

$\mathcal{P}_T$ total profit (or loss) of the agent for an episode of length T

1 Introduction

1.1 Motivation

Trading financial assets such as stocks, indices and currencies has received much attention in academic literature. Among others, the problem of trading financial assets has been approached with methods from the field of reinforcement learning (RL), with early successes by e.g. Moody et al. in 1998 [1], [2]. Recent advances in the field of reinforcement learning, e.g. [3], [4], have opened a whole new level of opportunities. For instance, reinforcement learning agents were used to beat human champions in games such as Go, Dota 2 and StarCraft II [5]–[7]. Recently developed successful reinforcement learning methods have also been applied for trading in the financial domain [8].

In the electricity domain, the problem of trading has also been approached with reinforcement learning [9]–[11]. However, the focus has been on addressing the problem of optimal trade execution (OTE), where a certain amount of electricity is either to be sold or to be bought in order to maximize revenue or minimize cost respectively. In contrast, this thesis addresses the problem of optimizing profit by doing both, buying and selling electricity, as in the financial domain. This problem has received little attention from the research community in the electricity domain. Nevertheless, trading electricity by doing both, buying and selling, is practically possible. Just like in the financial domain, this problem may be approached with reinforcement learning. This leads to the questions: what is reinforcement learning and why it is a suitable methodological choice for trading electricity?

In the field of statistical learning, we generally distinguish between unsupervised and supervised learning [12]. Reinforcement learning may be seen as a third paradigm. The main idea of reinforcement learning is to train an *agent* to take *actions* in an *environment* in order to optimize some *objective function*. The agent learns by actively gaining experience through exploration: in some environment state s, the agent takes an action a and receives a reward r (and continues this process from the next state s'). Learning by taking actions and receiving a feedback from the environment is what distinguishes reinforcement learning from supervised learning. In supervised learning, we estimate functions $f : x \to y$ based on some given, labeled training data (X, y). In reinforcement learning, we learn functions $f : s \to a$ based on experience in form of tuples (s, a, r, s') (representing state, action, reward, next state), which are not given *as is* but instead *collected by the agent itself*. By making use of exploration, the agent learns to take sequences of actions that are optimal without ever being

given which actions are optimal (not even in the form of training data). This motivates the choice of reinforcement learning for this thesis.

Importantly, RL agents are usually trained in an environment which dynamically responds to the agents' actions. In video games such as Dota 2, the agent might attack the opponent, and as a consequence, the opponent will fight back or try to defend itself. However, it is also possible to train an agent in an environment that does *not* change as a consequence of the agent's actions. In the context of trading for example, one could make the assumption that the agent's trades do not affect the market price of whatever product is traded. In this case, the environment, which is mainly based on the market price, would not be affected by the actions of the agent. Hence, the environment may be represented with a *static* data set consisting of historic time series of the market price. Please refer to section A.1 for more details. Using static data is a common approach in research about reinforcement learning for trading and was, for instance, used in [8], [13], [14]. Also in this thesis, RL agents were trained in an environment where states were represented with *static* data as described in chapter 4.

1.2 Objective

The underlying purpose of the thesis is to create a RL agent that can profitably trade electricity. In particular, the RL agent is supposed to maximize *profit per volume traded*. This means that each volume of electricity traded by the agent should result in large incremental profits. In this thesis, the agent is supposed to make trading decisions not only about whether to *buy or sell* but also about the *volume* to buy or sell.

Trades are to be placed on the continuous hourly intra-day market. In this market, electricity is traded that is to be delivered at a specific hour in the future on the same or next day. Moreover, electricity is traded continuously, meaning that orders can be placed at any given time until shortly before the specific hour at which the electricity is delivered to consumers.

1.3 Research Questions

This thesis aims to answer three research questions. These questions are relevant not only from a practical perspective but also from an academic perspective. First, it is important to define the reward function for the RL system because, in the context of RL, the reward function determines which actions the agent learns to take. Second, it is relevant to understand how different methodological approaches may affect the performance. Third, it is important to compare the results with a baseline in order to understand how well the RL agent performs relative to simple approaches. In particular, the first and the second question have received attention in academic literature, for instance in [2] and [15] respectively.

1. **Reward**: Is the Sharpe ratio, a prevalent reward function from the financial domain, the best reward function for intra-day trading in the electricity domain?
2. **Method**: Does the RL agent achieve a better performance when trained with an actor-critic method than when trained with a critic-only method?
3. **Baseline**: Does the RL agent achieve a better performance in trading electricity than baseline strategies that are commonly used for trading?

1.4 Delimitations

The thesis focuses exclusively on the German continuous intra-day electricity market and it focuses exclusively on hourly contracts (for electricity that is delivered at a specific hour). Furthermore, the agent only places market orders (and not limit orders or any other type of orders). This means that orders specify the volume to buy or sell (but not the price at which to buy or sell since all orders are executed at market price).

2 Theory

While it is possible to use supervised learning for trading, reinforcement learning provides a much more elegant approach [15]. This is because trading applications are essentially applications that perform actions to optimize a certain objective and in reinforcement learning, an agent is trained to perform actions to optimize a certain objective as well.

In contrast, the approach of supervised learning is less elegant. In supervised learning, prediction models are trained to make forecasts of e.g. future prices. These forecasts subsequently need to be mapped to actions explicitly. This approach may not be optimal for various reasons, such as failure to optimize the objective [13], [15]. For instance, the objective may be to maximize profits, but forecasts are made to minimize forecast error. In addition, the forecasts may not be mapped to actions in a way that maximizes profits. Finding an optimal mapping from predictions to actions is non-trivial, and therefore, it is common to use heuristics such as: *buy* some volume if the forecast corresponds to a price *increase* by 3% and *sell* some volume if the forecast corresponds to a price *decrease* by 3% [16]. Reinforcement learning on the other hand, allows to learn sequences of actions that optimize the objective directly, without any intermediate steps such as predictions. Importantly, reinforcement learning does not require any prior knowledge about which actions are optimal. Therefore, this thesis mainly uses reinforcement learning instead of supervised learning.

In this chapter, theoretical concepts relevant for this thesis are introduced. Section 2.1 introduces the elements of reinforcement learning, section 2.2 introduces neural networks, section 2.3 provides an overview of RL methods, section 2.5 introduces the first RL method applied in this thesis, section 2.4 introduces the second RL method applied in this thesis, and section 2.6 introduces reward functions that are used for reinforcement learning in the context of trading. Throughout this thesis, efforts were made to maintain the notation of the text book *Reinforcement learning: An introduction* by Sutton and Barto [17]. This text book is also the main reference for sections 2.1 and 2.3.

2.1 Introduction to Reinforcement Learning

The purpose of this section is to give an overview of the elements of reinforcement learning. This section does not explain actual methods but rather introduces the terms and concepts that are necessary to understand RL methods.

2.1.1 Finite Markov decision process

Finite Markov decision processes (MDPs) represent the foundation of reinforcement learning theory. A finite MDP consists of four elements: a finite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a finite set of rewards $\mathcal{R}$ and the probability distribution displayed in equation 2.1. In general, states contain the information available to the decision maker (also called *agent*), actions represent the decisions taken by the agent and rewards represent the feedback that the agent receives in response to taking a certain action in a certain state.

$$p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a), \tag{2.1}$$

The probability distribution from equation 2.1 is also referred to as $p(s', r|s, a)$ in short. S_{t+1}, S_t, A_t and R_{t+1} are random variables representing states, actions and rewards respectively and $s', s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ and $r \in \mathcal{R}$ are values of these random variables. $\mathcal{A}(s)$ denotes the set of valid actions for a state with value s. The prime $'$ denotes the value at the next timestep, meaning that s is the state value at the current timestep and s' is the state value at the next timestep.

There are two main assumptions of the finite MDP [18]. First, it is assumed that the *Markov property* is fulfilled and second, it is assumed that the *temporal homogenous transition probability property* is fulfilled. Both are reflected in the probability distribution $p(s', r|s, a)$.

1. The *Markov property* states that the random variables at time $t + 1$ only depend on random variables at time t (but not on any random variables before that, such as $t - 1$). In other words, S_{t+1} and R_{t+1} only depend on S_t and A_t (but not on S_{t-1} and A_{t-1} or any other random variables before).

2. The *temporal homogenous transition probability property* states that the random variables S_{t+1} and R_{t+1} do not depend on the value of time t itself. In other words, the probability $Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ is the same when $t = 1$ as when $t = 100$ (as long as the values s and a in the condition are the same).

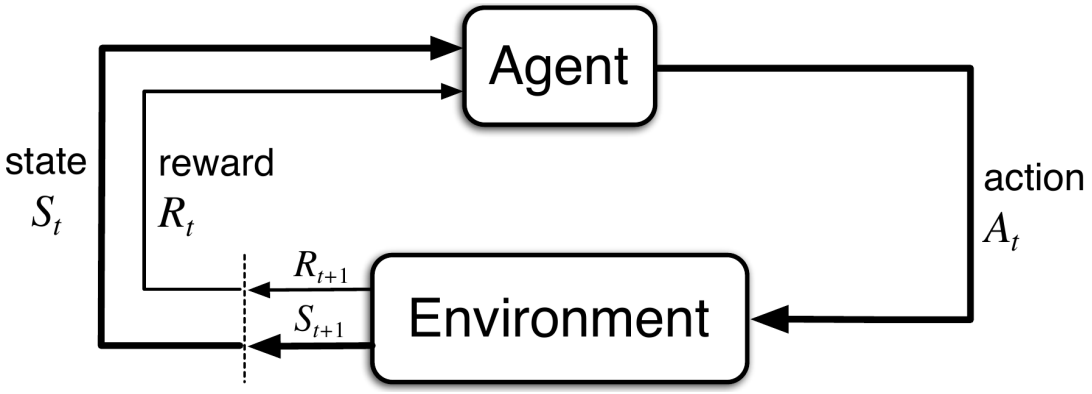


Figure 2.1: Agent-environment interaction in a Markov decision process [17]

2.1.2 Agent-environment interaction

In the context of RL, MDPs describe the interaction between an agent and its environment. At every timestep t, the agent is in a certain state S_t and takes an action A_t. Subsequently, the environment issues a reward R_{t+1} and the next state of the agent S_{t+1}. This process is visualized in figure 2.1. It repeats until termination if the environment is episodic like in this thesis (or until infinity if the environment is continuing). In an episodic environment, the following *trajectory* is observed for one episode of length T:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ..., S_{T-1}, A_{T-1}, R_T, S_T \tag{2.2}$$

The definition of episodes depends on the RL application. In the context of board games, an episode might represent one game of, for instance, chess or Go. Similarly in video games, an episode might represent one game session that lasts until the player looses or wins. In the context of intra-day trading, an episode might correspond to one day of trading.

While the finite MDP seems suitable for formalizing the problem of reinforcement learning, its assumptions are generally relaxed for RL problems in practice. Sutton and Barto encourage to "think of the state as whatever information is available to the agent about its environment" [17, p. 7]. In the context of trading for instance, the state S_t may contain information about past market prices from $t-k, ..., t$ instead of only information about the market price at time t. Thereby, the *Markov property* would be relaxed. Furthermore, the time t may be encoded in the state space as well if it provides valuable information to the agent. Thereby, the *temporal homogenous transition probability property* would be relaxed as well. In addition, many practical problems, such as the problem of trading electricity, are best represented not by a discrete state space with a finite set of states but instead with a continuous state space, which is still denoted as $\mathcal{S}$. In summary, the MDP is used to formalize the problem of reinforcement learning, but its assumptions may be relaxed in practice.

2.1.3 Return

The return G_t is the sum of discounted future rewards, as shown in equation 2.3. This equation is generally used for both, continuing environments (with infinitely many timesteps) and episodic environments (with T timesteps per episode). However, for episodic environments, we may simply express the return as in equation 2.4. Note that the return G_t can also be expressed recursively as in equation 2.5. The discount factor $\gamma \in [0, 1]$ is generally used to assign less weight to rewards in the far future than to rewards in the near future. This is done because there is more uncertainty for rewards in the far future.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad \text{general case} \tag{2.3}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \qquad \text{episodic case} \tag{2.4}$$

$$G_t = R_{t+1} + \gamma G_{t+1} \qquad \text{recursive expression} \tag{2.5}$$

Importantly, the return is of relevance because the objective of the agent is to maximize the expected return $\mathbb{E}[G_t|\cdot]$, as represented by the state-value and action-value functions in equations 2.7 and 2.8 respectively.

For understanding, it may be helpful to reflect on the differences between the reward R_t and the return G_t. The reward R_t is given directly by the environment at time t, whereas the return G_t is not yet known at time t because it depends on the rewards from future timesteps. Further note that while the reward R_t depends on the past action A_{t-1} (and possibly other past actions), the return G_t depends on the action A_t and on *future* actions (that result in the future rewards $R_{t+1}, R_{t+2}, ..., R_T$).

For example, consider the last reward and the last return of an episode of length T with a trajectory as in equation 2.2. The last reward, R_T, is simply a reward like any other reward during the episode. It is given by the environment upon the action A_{T-1} taken by the agent. The return G_T, however, is *zero*. This is because in an episodic environment, no future rewards will be observed after timestep T, and the return is computed based on future rewards.

2.1.4 Policy

Before introducing the state-value function and the action-value function in the subsequent sections, the term *policy* needs to be introduced. A policy π, also known from decision theory [19], is a probability distribution. It describes the probability that the action A_t takes the value a given that the state of the agent S_t takes the value s. Note that a policy may be deterministic, namely in the case where a particular action value a occurs with a probability of 1, i.e. $\pi(A_t = a|S_t = s) = 1$, and all remaining actions take probabilities of 0.

$$\pi(a|s) = \pi(A_t = a|S_t = s), \tag{2.6}$$

where $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, i.e. s is any valid state and a is any valid action for the state s.

2.1.5 State-value function

The state-value function $v_\pi(s)$ represents the expected return conditioned on the current state S_t, assuming that the agent follows the policy π from S_t onwards. In reinforcement learning, we generally aim to find a policy π which maximizes the state-value function (or the action-value function from the next section).

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t|S_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{2.7}$$

2.1.6 Action-value function

The action-value function $q_\pi(s, a)$ represents the expected return conditioned on the current state S_t and conditioned on the action A_t taken by the agent in this state, assuming that the agent follows the policy π from S_{t+1} onwards. The difference between the state-value function and the action-value function is the following: in the action-value function, the action A_t taken in state S_t does not necessarily have to conform to the policy π, whereas in the state-value function it does.

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t|S_t = s, A_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{2.8}$$

2.1.7 Optimal policy, state-value function and action-value function

An optimal policy π^* is a policy that maximizes the state-value function $v_\pi(s)$ and the action-value function $q_\pi(s, a)$ as shown in equation 2.9. Reversely, the optimal state-value function $v^*(s)$ and optimal action-value function $q^*(s, a)$ are represented by the state-value and action-value functions that maximize the expected return (by following the optimal policy π^*) as shown in equations 2.10 and 2.11. The equations below apply for all $s \in \mathcal{S}$ and for all $a \in \mathcal{A}(s)$, i.e. for all possible states and all valid actions in the respective states.

$$\pi^*(A_t = a|S_t = s) = \operatorname*{argmax}_\pi v_\pi(s) = \operatorname*{argmax}_\pi q_\pi(s, a) \tag{2.9}$$

$$v^*(s) = \max_\pi v_\pi(s) = \mathbb{E}_{\pi^*} \left[G_t|S_t = s\right] \tag{2.10}$$

$$q^*(s, a) = \max_\pi q_\pi(s, a) = \mathbb{E}_{\pi^*} \left[G_t|S_t = s, A_t = a\right] \tag{2.11}$$

Note that if the optimal policy π^* was known, we would know exactly which actions to take to maximize the return G_t. Since the optimal policy is generally not known, the purpose of reinforcement learning is to approximate the optimal policy π^* with $\hat{\pi}$. This is is also referred to as the control problem described in section 2.1.9 below.

2.1.8 Prediction problem

The prediction problem is the problem of estimating the state-value function $v_\pi(s)$ or the action-value function $q_\pi(s,a)$ for a given, fixed policy π. The approximations of the state-value and the action-value function are denoted as $\hat{v}_\pi(s)$ and $\hat{q}_\pi(s,a)$. If we solve the prediction problem for different policies, we may come to a conclusion about which policy is preferred, i.e. which policy gives rise to larger expected returns. Consequently, solving the prediction problem for different policies allows to compare these policies and identify the policy that approximates the optimal policy π^* best.

2.1.9 Control problem

The control problem is the problem of finding an optimal policy π^*, or more realistically an approximation of it, namely $\hat{\pi}$. To find an approximation of the optimal policy, different policies may have to be evaluated and compared. This leads back to the prediction problem: in order to solve the control problem, we may have to solve the prediction problem for multiple different policies until we find a policy that cannot be improved further.

2.1.10 Bellman equations

Bellman equations (and adaptations of them) are used in reinforcement learning for solving both, the prediction and the control problem. Essentially, the state-value function $v_\pi(s)$ and the action-value function $q_\pi(s,a)$ are reformulated recursively. The resulting Bellman equations are then turned into assignment statements used to update the approximations of the state-value and the action-value function, $\hat{v}_\pi(s)$ and $\hat{q}_\pi(s,a)$. The main advantage of the recursive formulation is that we can perform such updates after observing a single experience tuple (s,a,r,s') instead of having to wait until all tuples of a particular episode are observed.

As stated before, the return G_t from equation 2.5 can be expressed recursively as $G_t = R_{t+1} + \gamma G_{t+1}$. We may hence reformulate the state-value function by making use of this recursive relationship, which takes us from equation 2.12 to equation 2.13. The corresponding Bellman equation is shown in the subsequent equation 2.14. It's main characteristic is that $v_\pi(s)$ is represented as a function of itself, more precisely of $v_\pi(s')$, where s' is the value of the next state S_{t+1}.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right] && (2.12)\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] && (2.13)\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s') | S_t = s\right] && (2.14)
\end{aligned}
$$

Although the step from equation 2.13 to the Bellman equation 2.14 seems intuitively reasonable (having in mind that $v_\pi(s')$ is an expectation of G_{t+1} since it is defined as $v_\pi(s') = \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s'\right]$), the proof is rather involved and omitted here.

Importantly, we may not only formulate the Bellman equation for the state-value function but also for the action-value function and for the optimal versions of these two functions as shown in section 2.1.7. An overview of all these corresponding Bellman equations be found in the equations below.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s') | S_t = s\right] && (2.15)\\
q_\pi(s,a) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s') | S_t = s, A_t = a\right] && (2.16)\\
v^*(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v^*(s') | S_t = s, A_t = a\right] && (2.17)\\
q^*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q^*(s',a') | S_t = s, A_t = a] && (2.18)
\end{aligned}
$$

2.1.11 Application of the Bellman equations

The expected return represented by the state-value function $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$ and the action-value function $q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$ is generally not known. Therefore, it needs to be estimated by drawing samples of G_t. To be precise, the agent draws samples of experience tuples (s,a,r,s'): in a given state s, the agent takes action a and observes the reward r and the next state s'. Repeated many times, this results in a trajectory $S_0, A_0, R_1, S_1, ..., S_{T-1}, A_{T-1}, R_T, S_T$. Based on this trajectory, the return G_t can then be computed for every t, among others by making use of the Bellman equations.

There are various possibilities for representing the return G_t. Here, four of them are introduced briefly: Monte Carlo, 1-step Q-learning, 1-step TD and n-step TD. TD stands for temporal difference. Q-learning also falls into the category of temporal difference methods, meaning that 1-step Q-learning, 1-step TD and n-step TD are temporal difference methods.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T \qquad \text{MC} \qquad (2.19)$$

$$G_t = R_{t+1} + \gamma \max_{a'} \hat{q}_\pi(S_{t+1}, A_{t+1} = a') \qquad \text{1-step Q} \qquad (2.20)$$

$$G_t = R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}) \qquad \text{1-step TD} \qquad (2.21)$$

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}_\pi(S_{t+n}) \qquad \text{n-step TD} \qquad (2.22)$$

First, we may (in an episodic environment) wait until the end of an episode when all rewards are observed. Then, we could represent G_t with the *Monte Carlo* (MC) approach, based on all observed, discounted rewards as in equation 2.19. The Bellman equations are not applied if the Monte Carlo approach is used because all rewards are observed, and no recursive expression with the state-value or the action-value function is needed.

Alternatively, we may make use of a *1-step* approach. In the 1-step approach, G_t is represented as a combination of R_{t+1} and of the discounted expected return thereafter. In 1-step Q-learning, as in equation 2.20, the discounted expected return is represented with the action-value function by applying the idea of the Bellman equation 2.18. In 1-step TD, as in equation 2.21, the discounted expected return is represented with the state-value function by applying the idea of the Bellman equation 2.17.

Lastly, we may make use of an *n-step* approach. In the n-step approach, G_t is represented as a combination of observed, discounted rewards from R_{t+1} to R_{t+n} and of the discounted expected return $\hat{v}_\pi(S_{t+n})$ thereafter. This approach can be seen as a compromise between the Monte Carlo approach, where G_t is exclusively based on observed rewards, and the 1-step approach, where only one observed reward contributes to G_t.

2.2 Introduction to Neural Networks

Neural networks (NN) are an important component of the RL methods described later in section 2.3. Neural networks represent a type of machine learning model that can approximate complex, non-linear functions $f : x \to y$. A successful form of neural networks is the multilayer perceptron (MLP) described by Bishop in [20], the main reference for this section. In the following, MLPs are introduced very briefly because they were used to represent the RL agents in this thesis. Please refer to [20] for a detailed introduction.

An example of a MLP with two hidden layers is displayed in figure 2.2. Visibly, it resembles a network of neurons connected with synapses, motivating the name of *neural networks*. Given some input vector x, the MLP returns a prediction in form of the output vector $\hat{y}$. The corresponding true output vector is denoted as y and is available for training. Each synapse, i.e. arrow, in the MLP represents a weight w. The objective of training a MLP is to learn all weights so that the prediction error $\frac{1}{2}\sum_k^K (y_k - \hat{y}_k)^2$ is minimized.

To achieve this objective, two processes are repeated iteratively: forward propagation and backward propagation. Forward propagation is equivalent to making a prediction: the input

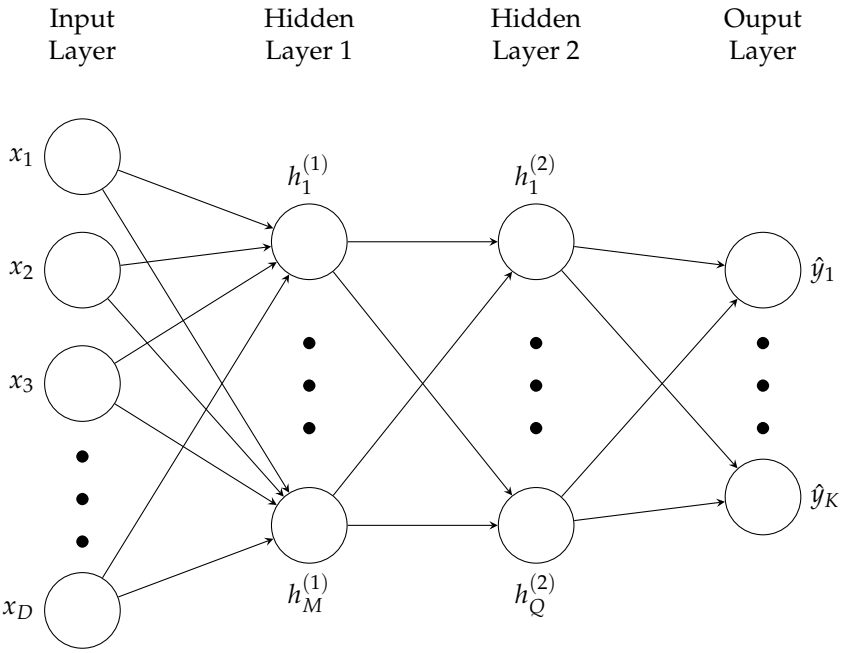


Figure 2.2: Example of a multilayer perceptron with 2 hidden layers. The input layer consists of D nodes, the first hidden layer consists of M nodes, the second hidden layer consists of Q nodes, and the output layer consists of K nodes.

vector x is turned into the corresponding output vector $\hat{y}$. Backward propagation is equivalent to conducting weight updates: the prediction error $\frac{1}{2}\sum_k^K (y_k - \hat{y}_k)^2$ is computed, and each weight is updated in order to decrease the prediction error by using gradient descent.

Forward propagation is conducted by making use of perceptrons. As the name suggests, multilayer perceptrons are networks that consist of multiple layers of perceptrons. Each node in the MLP (except for the input nodes) represents the output of a perceptron. The perceptron consists of three elements: input values, weights and activation function. This is best explained with an example. For instance, the first node of the first hidden layer, denoted as $h_1^{(1)}$, is computed with the following perceptron:

$$h_1^{(1)} = \sigma\left(\sum_{d=1}^{D} x_d w_{1,d}^{(1)}\right), \tag{2.23}$$

Here, input values, weights and activation function are represented by x_d, $w_{1,d}^{(1)}$ and σ respectively. In particular, $w_{1,d}^{(1)}$ represents the weight corresponding to the arrow pointing to node $h_1^{(1)}$ from node x_d. Note that $h_1^{(1)}$ is computed based on all nodes from its previous layer, i.e. $x_1, ..., x_D$, and based on all weights corresponding to arrows pointing from these nodes to $h_1^{(1)}$. Similarly, any of the other nodes in the networks of the hidden layers and the output layers are computed, ultimately resulting in the output vector $\hat{y}$. The purpose of the activation function is, among others, to introduce non-linearity, helping to fit non-linear functions [21]. Please refer to [21] for an introduction about the activation functions used in this thesis, including the ReLU, tanh and softmax activation functions.

Backward propagation is conducted for each weight by performing a gradient step into the direction of decreasing prediction error. For each weight, the partial derivative of the prediction error $\epsilon = \frac{1}{2}\sum_k^K (y_k - \hat{y}_k)^2$ with respect to the weight is computed, denoted as $\frac{\partial \epsilon}{\partial w}$. Subsequently, the weight is updated by taking a step of gradient descent: $w = w - \eta \frac{\partial \epsilon}{\partial w}$, where η denotes some learning rate. If forward propagation and backward propagation are repeated

multiple times iteratively, such weight updates are performed numerously. Ultimately, this results in weights that minimize the prediction error ϵ.

2.3 Methods in Reinforcement Learning

The purpose of this section is to give an overview of the methods from the field of reinforcement learning. In section 2.3.1, the choice of approximate solution methods over tabular solution methods is explained. Subsequently, the three common approaches from the field of approximate solution methods (critic-only, actor-only and actor-critic) are introduced in the sections 2.3.2, 2.3.3 and 2.3.4. Hereby, the focus is on explaining the general concepts rather than the exact methods used for this thesis. The methods used in the thesis, are explained in the subsequent sections: the advantage-actor critic method (A2C), representing the actor-critic approach, is explained in section 2.4, and the deep Q-network (DQN), representing the critic-only approach, is explained in section 2.5.

2.3.1 Tabular solution methods vs. approximate solution methods

Sutton and Barto distinguish between two categories of methods in the field of reinforcement learning: tabular solution methods and approximate solution methods [17]. The main difference between these two is that tabular solution methods require a finite, discrete state space $\mathcal{S}$ and a finite, discrete action space $\mathcal{A}$. This allows for a tabular representation of $\hat{q}(s,a) \approx q^*(s,a)$ with rows for states and columns for actions and of $\hat{v}(s) \approx v^*(s)$ with rows for states and only one column. For instance, given a discrete value of s and a discrete value of a, the action-value $\hat{q}(s,a)$ can be looked up in the action-value table at the row corresponding to s and the column corresponding to a.

However, if the state space (and possibly also the action space) is continuous, then it does not make sense to pick a tabular representation for $\hat{q}(s,a)$ and $\hat{v}(s)$. Instead, $\hat{q}(s,a)$ and $\hat{v}(s)$ may be represented as functions that can take any (continuous) values of s and a as input and return $\hat{q}(s,a)$ and $\hat{v}(s)$ as output. Such functions may for instance be represented with neural networks. Instead of learning values in tables, the parameters (also called weights) of the neural network are to be learned. In other words, $\hat{q}(s,a)$ and $\hat{v}(s)$ are parameterized as some functions $\hat{q}(s,a,\mathbf{w}_q) \approx q^*(s,a)$ and $\hat{v}(s,\mathbf{w}_v) \approx v^*(s)$, where the weight vectors $\mathbf{w}_q$ and $\mathbf{w}_v$ are to be learned during training. In addition, the policy may also be parameterized as $\hat{\pi}(a|s,\boldsymbol{\theta}) \approx \pi^*(a|s)$, where $\boldsymbol{\theta}$ is the weight vector to be learned. Different forms of parameterization are possible but most commonly neural networks have been used since they were shown to be very successful for training RL agents to play Atari games [3].

While tabular solution methods are helpful for understanding the basics of reinforcement learning, they are not covered here because they are not actually applied for this thesis. Instead, emphasis is put on approximate solution methods since all RL methods used in this thesis are approximate solution methods. The choice fell on approximate solution methods because the state space $\mathcal{S}$ is naturally continuous in the context of trading due to continuous variables such as *market price*. This approach is in line with academic research: as the literature review chapter 3 shows, approximate solution methods are predominant in studies about RL for trading.

2.3.2 Critic-Only Methods

In critic-only methods, only the action-value function is used, which is parameterized as $\hat{q}(s,a,\mathbf{w}_q)$. An example of a neural network parameterizing the action-value function is displayed in figure 2.3. The network takes the state s, consisting of D values, as input and gives the expected return $\hat{q}(s,a)$ of this state for each possible discrete action as output. This means that there is one output node per action. In case of three actions, the three output nodes represent the expected returns $\hat{q}(s,a_1)$, $\hat{q}(s,a_2)$ and $\hat{q}(s,a_3)$.

The neural network can be used as follows: for any given state s, the network returns $\hat{q}(s, a_1)$, $\hat{q}(s, a_2)$ and $\hat{q}(s, a_3)$, and subsequently, the action which returns the highest expected return can be taken by the agent in this state. This results in the deterministic policy $\hat{\pi} = \text{argmax}_a \hat{q}(s, a)$. Regarding the network architecture, note that it would also be possible to include the actions into the input layer instead of the output layer. However, since the $\hat{q}(s, a)$ values are needed for all actions, the neural network would have to do several predictions in this case. By including the actions into the output layer, as shown in figure 2.3, all $\hat{q}(s, a)$ values can be obtained with only one prediction. Hence, this architecture is preferred.

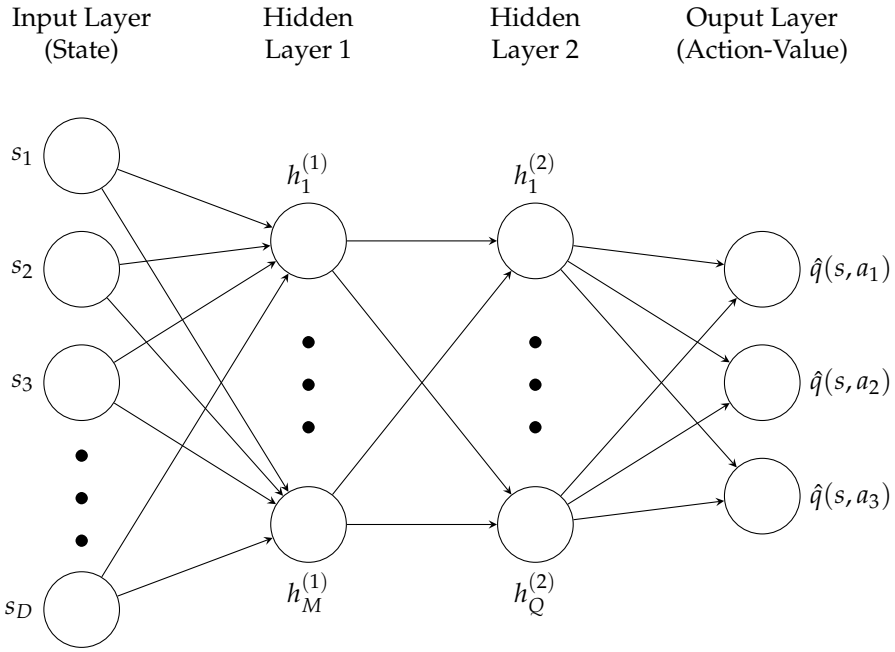


Figure 2.3: Action-value function parameterization $\hat{q}(s, a, \mathbf{w}_q)$ with NN (for critic-only methods). This network is also used for DQN in this thesis.

Intuitively, we may at first say that the objective should be to fit $\hat{q}(s, a, \mathbf{w})$ so that it represents the corresponding *true* expected reward $q(s, a)$ as accurately as possible, measured by the mean squared error. This is because if we know the true expected reward, then we can always pick the truly optimal action a in any given state.

Recall that $q(s, a)$ is defined as an expectation $q(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$. Instead of comparing the corresponding approximation $\hat{q}(s, a, \mathbf{w})$ with $q(s, a)$ directly, we instead compare $\hat{q}(s, a, \mathbf{w})$ with samples of G_t. We do this because $q(s, a)$ is unknown. (If it was known, there would be no need to approximate it with $\hat{q}(s, a, \mathbf{w})$). In conclusion, we aim to find $\hat{q}(s, a, \mathbf{w})$ so that it minimizes the loss in equation 2.24. The corresponding gradient can be derived as in equation 2.25, giving us a problem that may be solved with gradient descent as shown in the algorithm 2. G_t may be represented with the 1-step Q-learning approach, as in equation 2.20: $G_t = R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, A_{t+1} = a', \mathbf{w})$.

$$L(\mathbf{w}) = \frac{1}{2} \left[G_t - \hat{q}(S_t, A_t, \mathbf{w}) \right]^2 \tag{2.24}$$

$$\nabla L(\mathbf{w}) = - \left[G_t - \hat{q}(S_t, A_t, \mathbf{w}) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}) \tag{2.25}$$

2.3.3 Actor-Only Methods

In actor-only methods, only the policy function is used, which is parameterized as $\hat{\pi}(a|s, \boldsymbol{\theta})$. An example of a neural network parameterizing the action-value function is displayed in

figure 2.4. The network takes the state s, consisting of D values, as input and returns the probability of taking each action $\hat{\pi}(a|s)$ as output, here for three actions in total. This means that there is, again, one output node per action. In case of three actions, the three output nodes represent the probabilities $\hat{\pi}(a_1|s)$, $\hat{\pi}(a_2|s)$ and $\hat{\pi}(a_3|s)$. In the neural network, the softmax activation function is used in the output layer to ensure that the probabilities sum up to 1, so that the policy is a valid probability distribution: $\sum_{i\in\{1,2,3\}} \hat{\pi}(a_i|s) = 1$.

The neural network can be used as follows. For any given state s, the network returns the probabilities $\hat{\pi}(a_1|s)$, $\hat{\pi}(a_2|s)$ and $\hat{\pi}(a_3|s)$. Subsequently, an action is sampled from this probability distribution and then taken by the agent, resulting in a stochastic policy. Alternatively, the action with the largest probability can be taken (instead of sampling from the probability distribution), resulting in a deterministic policy. Note that the action-value function, which is used in the critic-only methods to obtain the action, is not needed for actor-only methods because in actor-only methods the action comes directly from the policy function.

Intuitively, every update of the parameters $\boldsymbol{\theta}$ is supposed to result in an updated policy $\hat{\pi}(a|s, \boldsymbol{\theta})$ which gives actions resulting in larger expected return than before the update. In other words, we would like to perform gradient ascent, shifting the policy weights $\boldsymbol{\theta}$ into the direction of maximum increase of the expected return. The expected return can be represented with the state-value function $v_{\hat{\pi}}(s)$.

For conducting gradient ascent, we make use of the *policy gradient theorem*. Essentially, this theorem expresses the gradient of the state-value function $v_{\hat{\pi}}(s)$ with respect to the policy parameters $\boldsymbol{\theta}$. This is possible because as described previously in equation 2.7, the state-value function $v_{\hat{\pi}}(s)$ depends on the policy $\hat{\pi}$: the state-value function gives the expected return assuming that the agent starts in state s and follows the policy $\hat{\pi}$ thereafter. The policy gradient theorem gives us the result from equation 2.26. The proof is omitted here but it can be found in the text book by Sutton and Barto [17].

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[G_t \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta})\right], \tag{2.26}$$

where $J(\boldsymbol{\theta})$ represents the performance function $v_{\hat{\pi}}(s_t)$, which means that $\nabla J(\boldsymbol{\theta}) = \nabla v_{\hat{\pi}}(s_t)$. To illustrate actor-only methods, algorithm 3 shows a particular actor-critic algorithm called REINFORCE. Note that REINFORCE represents G_t with the Monte Carlo approach from equation 2.19, whereas in the critic-only algorithm 2 G_t was represented with the 1-step Q-learning approach from equation 2.20.

2.3.4 Actor-Critic Methods

In actor-critic methods, only the policy function and the state-value function are used, which are parameterized as $\hat{\pi}(a|s, \boldsymbol{\theta})$ and $\hat{v}(s, \mathbf{w}_v)$. The policy function $\hat{\pi}(a|s, \boldsymbol{\theta})$ is the same as the policy function from the actor-only methods in figure 2.4, except that the weights $\boldsymbol{\theta}$ will differ after training. An example of a neural network parameterizing the state-value function is displayed in figure 2.5. The network takes the state s, consisting of D values, as input and gives the expected return $\hat{v}(s)$ as output. This means that there is only one output node. The policy network $\hat{\pi}(a|s, \boldsymbol{\theta})$ is used in the same way as in the actor-only method from section 2.3.3. The state-value network $\hat{v}(s, \mathbf{w}_v)$ is only used during training but not after training.

Actor-critic methods are closely related to actor-only methods. To motivate actor-critic methods, we may focus on the term G_t in the policy gradient theorem $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[G_t \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta})\right]$. G_t may vary strongly depending on the corresponding observed rewards from R_{t+1} to R_T. If the observed rewards are large, G_t will be large and therefore the gradient ascent step will do a strong update to the weights $\boldsymbol{\theta}$. This may result in a very unstable training process, which raises the question whether we may somehow reduce the variance of G_t. We may do so by subtracting a baseline $b(S_t)$ from it, adjusting the policy gradient from equation 2.26 to equation 2.27.

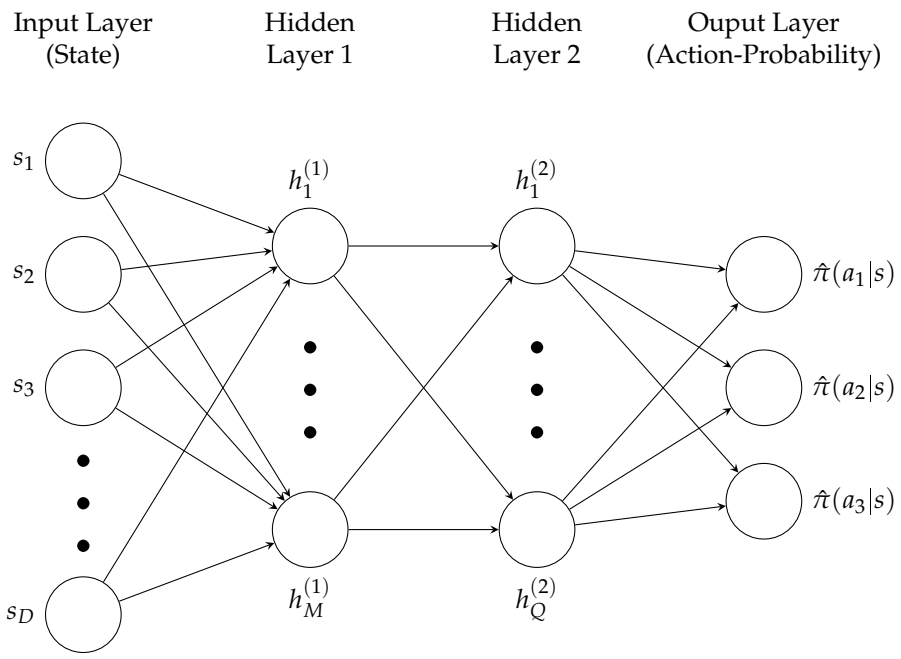


Figure 2.4: Policy function parameterization $\hat{\pi}(a|s, \boldsymbol{\theta})$ with NN (for actor-only and actor-critic methods). This network is also used for A2C in this thesis.

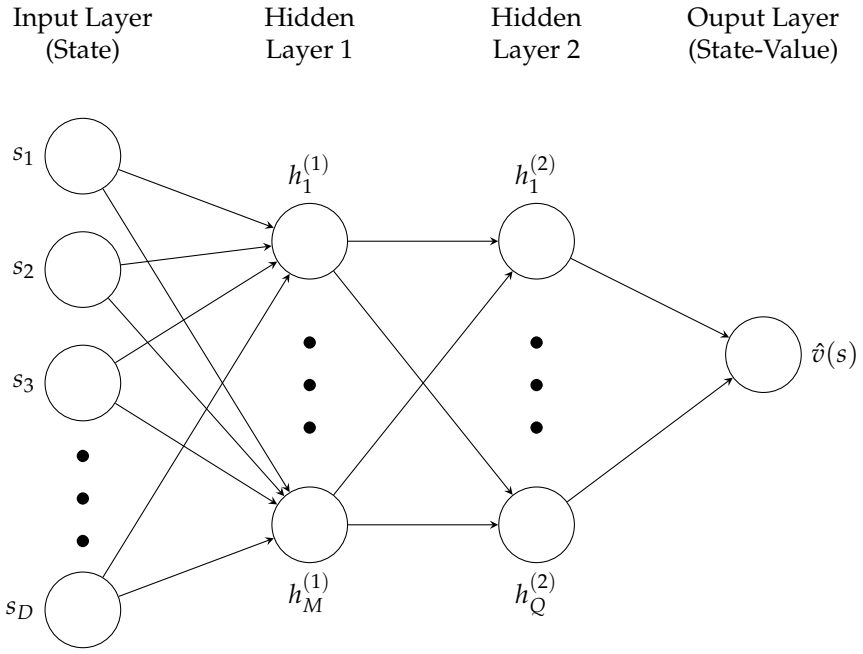


Figure 2.5: State-value function parameterization $\hat{v}(s, \mathbf{w}_v)$ with NN (for actor-critic methods). This network is also used for A2C in this thesis.

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[(G_t - b(S_t)) \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta}) \right] \tag{2.27}$$

This leads to the question which function should be picked as a baseline $b(S_t)$. To reduce variance, it seems reasonable to subtract the expected value of G_t from G_t. We have expressed the expected value of G_t before, namely as state-value function: $v_\pi(s) = \mathbb{E}_\pi [G_t|S_t = s]$. Hence, we may choose an approximation of the state-value function $\hat{v}(S_t, \mathbf{w})$ as baseline, which is shown in equation 2.28. Note that we choose the state-value function and not the action-value function because there is no need to have the action A_t in the condition. This is because the actions are now given by the policy $\hat{\pi}(a|s, \boldsymbol{\theta})$, unlike in the critic-only approach where the action-value function was required to obtain the policy $\hat{\pi} = \text{argmax}_a \, \hat{q}(s, a, \mathbf{w})$.

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[(G_t - \hat{v}(S_t, \mathbf{w})) \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta}) \right] \tag{2.28}$$

Finally, the question is how to represent G_t. In theory, we may apply any of the representations mentioned before in the section 2.1.11. For instance, we may choose a Monte Carlo representation as shown in equation 2.29 below. Alternatively, we might also choose a 1-step TD representation as shown in equation 2.30 below. As explained in section 2.1.11, TD stands for temporal difference. In temporal difference methods, the return G_t is expressed as a sum of observed rewards *and* of discounted expected return after the observed rewards. Hereby, the Bellman equations are applied. Importantly, actor-critic methods were first proposed with a temporal difference approach such as in equation 2.30 but not with a Monte Carlo approach as in equation 2.29 [22]. Sutton and Barto do not even consider the algorithm an actor-critic algorithm if the Monte Carlo approach is used [17]. Instead, they refer to it as a an actor-only method *with baseline*.

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[(R_{t+1} + \gamma R_{t+2} + ... \gamma^{T-t-1} R_T - \hat{v}(S_t, \mathbf{w})) \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta}) \right] \tag{2.29}$$

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta}) \right] \tag{2.30}$$

In summary, if we modify the policy gradient from equation 2.26 to equation 2.30 and learn the required approximation of the state-value function $\hat{v}(s, \mathbf{w})$ in addition to the approximation of the policy $\hat{\pi}(a|s, \boldsymbol{\theta})$, then we move from an actor-only method to an actor-critic method. The actor-critic method is illustrated in algorithm 5. For reference, the algorithm for the actor-only method with baseline is illustrated as well, namely in algorithm 4, as it may be helpful to understand the relation between actor-only and actor-critic methods. Note how the actor-only algorithm 3 differs from the actor-only with baseline algorithm 4 by adding the state-value function as baseline. Then note how the actor-only with baseline algorithm 4 differs from the actor-critic algorithm 5 by changing the representation of the return G_t from $R_{t+1} + \gamma R_{t+2} + ... \gamma^{T-t-1} R_T$ as in equation 2.29 to $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ as in equation 2.30.

Lastly, note that actor-critic methods are sometimes called *advantage* actor-critic methods. The term $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$ from equation 2.30 is referred to as δ in the algorithm 5. This term δ is also referred to as *advantage* [4]. Here, $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ represents the *observed* return G_t after taking a particular action A_t, whereas $\hat{v}(S_t, \mathbf{w})$ represents the *expected* return $\mathbb{E}_{\hat{\pi}} [G_t|S_t = s]$ under policy $\hat{\pi}(A_t|S_t, \boldsymbol{\theta})$. Therefore, if taking action A_t results in $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) > \hat{v}(S_t, \mathbf{w})$, then the action A_t leads to an *advantage* compared to the policy $\hat{\pi}(A_t|S_t, \boldsymbol{\theta})$. Importantly, A_t also comes from the policy itself but since it is a single action, it is not representative of the full policy, a probability distribution, and we may therefore conclude that it can have an advantage over the policy. Consequently, actor-critic methods are sometimes called *advantage* actor-critic methods.

2.4 Synchronous Advantage Actor-Critic (A2C)

A2C is generally referred to as *synchronous advantage actor-critic* method. As such, it belongs to the category of actor-critic methods introduced in section 2.3.4. A2C is originated in A3C, which is an *asynchronous advantage actor-critic* method. The main references for this section are the original paper about A3C from 2016 [4] and a blog post about A2C from 2017 [23]. Note that the A2C authors introduced A2C with said blog post instead of a published paper. This section focuses on A2C, which is the primary method used for this thesis.

In essence, A2C is an advantage actor-critic method as introduced in section 2.3.4. However, instead of a single agent collecting experience in form of tuples (s, a, r, s') in a single environment, there are several agents that collect experience in different copies of the environment. These agents collect experience simulatenously, running on different threads of the processor on the same machine. According to Mnih et al., the benefits of an advantage actor-critic method with multiple agents (e.g. A2C) compared to an advantage actor-critic method with a single agent are *computational efficiency and training stability* [4]. Importantly, training stability refers to the ability to converge across the training period, even if unexpected experience tuples (e.g. with unusually large or small rewards) disrupt the learning progress.

Importantly, the weights learned by the agents, $\boldsymbol{\theta}$ and $\mathbf{w}$ from $\hat{\pi}(a|s, \boldsymbol{\theta})$ and $\hat{v}(s, \mathbf{w})$, are *shared globally*, across the different agents in A2C. Whenever an agent performs an update of the weights, it updates the globally shared weights. Whenever an agent uses the weights, e.g. for sampling an action from the policy $\hat{\pi}(a|s, \boldsymbol{\theta})$, it uses the globally shared weights. The term *synchronous* implies that the agents wait for each other to collect their respective required number of tuples (s, a, r, s') before updating the globally shared weights. The number of tuples to be collected by each individual agent before an update is the hyperparameter t_{max}, which is set by the researcher.

The algorithm can be described as a 2-step process: 1) collection of experience and 2) weight updates. These two steps are repeated many times. In the first step, all agents fetch the up-to-date globally shared weights $\boldsymbol{\theta}$ and $\mathbf{w}$ and start collecting t_{max} tuples (s, a, r, s') each, using the policy and state-value function with said weights. In the second step, once all agents have completed collecting their tuples, the globally shared weights $\boldsymbol{\theta}$ and $\mathbf{w}$ are updated based on the collected tuples of *all* agents. The complete A2C training process is displayed in algorithm 6. To be precise, in the algorithm, not the tuples (s, a, r, s') are collected but instead the gradients. It does not matter for the result if we collect the tuples (s, a, r, s') and compute the gradients later or if we directly compute and collect the gradients.

Note how the A2C algorithm 6 resembles the actor-only with baseline algorithm 4 and the advantage actor-critic algorithm 5. In A2C however, multiple agents are used instead of a single agent. Moreover, G_t is neither represented with the Monte Carlo approach from equation 2.19 nor with the 1-step temporal difference approach from equation 2.21. Instead, G_t is represented with the n-step approach from equation 2.22. Each G_t is computed as the sum of some (at most t_{max}) observed, discounted rewards and of the discounted expected return $\hat{v}(S_{t+t_{max}}, \mathbf{w})$ thereafter.

At this point, it may be interesting to point out the difference between A2C and A3C: in A2C, all agents wait for each other before updating the globally shared weights collectively, whereas in A3C, each agent updates the globally shared weights directly once it has completed collecting sufficient experience. Consequently, in A2C all agents start collecting tuples (s, a, r, s') at the same time and they use the exact same weights doing so. In A3C however, the agents generally start collecting tuples (s, a, r, s') at different time points, and therefore the agents use different weights when collecting their experience.

Based on the current literature, it does not seem as if there is clear academic evidence that either A2C or A3C performs generally better although OpenAI noted in their blog post that A2C performed better in its benchmarks [23]. The main practical difference, according to OpenAI, is that A2C is more efficient when running on a GPU.

2.5 Deep Q-Network (DQN)

Deep Q-networks (DQNs), also called Q-networks for short, belong to the category of critic-only methods from section 2.3.2. Essentially, a DQN in its most basic form is a NN that parameterizes the action-value function as $\hat{q}(s, a, \mathbf{w})$. An example is displayed in figure 2.3.

However, there are a variety of additional ideas that led to the success of DQNs: the ideas of a separate *target network* and *experience replay* were key to the first success of DQNs [24], mainly because they improved *training stability*. As described in section 2.4, training stability refers to the ability of the method to converge across the training period, even if unexpected experience tuples disrupt the learning progress. Later, the ideas of *prioritized experience replay* [25], *dueling networks* [26] and *double Q-learning* [27] allowed for further enhancements. In the following, the concepts used in the final DQN model of this thesis are introduced shortly: target network, prioritized experience replay and dueling networks. These ideas, combined with the general introduction about critic-only methods from section 2.3.2, form the DQN.

2.5.1 Target Network

The *target network* was the first main idea contributing to the success of DQNs in playing Atari games [24]. Note that in the critic-only approach described in section 2.3.2, the weights $\mathbf{w}$ of the Q-network $\hat{q}(s, a, \mathbf{w})$ are updated based on the Q-network $\hat{q}(s, a, \mathbf{w})$ itself. This is because the Q-network is used for computing the losses (and hence also for computing the gradients used for the weight update as shown in equation 2.25). In particular, it is used *twice* in the loss function: first to represent the *true* return G_t and second to represent the predicted expected return $\hat{q}(S_t, A_t, \mathbf{w})$ as shown in equation 2.32 below:

$$L(\mathbf{w}) = \frac{1}{2}\left[G_t - \hat{q}(S_t, A_t, \mathbf{w})\right]^2 \tag{2.31}$$

$$= \frac{1}{2}\left[R_{t+1} + \gamma \max_{a'} \underbrace{\hat{q}(S_{t+1}, A_{t+1} = a', \mathbf{w})}_{1.} - \underbrace{\hat{q}(S_t, A_t, \mathbf{w})}_{2.}\right]^2 \tag{2.32}$$

Note that with the above approach, the weight update at each timestep does not only affect the *predicted* expected return (in 2.) but also the *true* return G_t (through 1.). Mnih et al. found this to be rather unstable and realized that they could improve *training stability* by introducing a separate *target network* [24]. Every C timesteps, the Q-network $\hat{q}(s, a, \mathbf{w})$ is copied to represent the target network $\hat{q}(s, a, \mathbf{w}^-)$. This means that the target network only changes every C timesteps, whereas the Q-network generally changes at *every* timestep when receiving a weight update. The target network is then used for representing the *true* return G_t and the Q-network is (as before) used to represent the predicted expected return as shown below in equation 2.33. Note that the only difference between equations 2.32 and 2.33 is that $\hat{q}(S_{t+1}, A_{t+1} = a', \mathbf{w})$ from 1. was replaced with the target network $\hat{q}(S_{t+1}, A_{t+1} = a', \mathbf{w}^-)$.

$$L(\mathbf{w}) = \frac{1}{2}\left[R_{t+1} + \gamma \max_{a'} \underbrace{\hat{q}(S_{t+1}, A_{t+1} = a', \mathbf{w}^-)}_{1.} - \underbrace{\hat{q}(S_t, A_t, \mathbf{w})}_{2.}\right]^2 \tag{2.33}$$

2.5.2 Prioritized Experience Replay

Experience replay was the second main idea contributing to the success of DQNs in playing Atari games [24]. Later, the improved technique *prioritized experience replay* was introduced by Schaul et al. in [25]. Both, experience replay and prioritized experience replay, improve

training stability. However, *prioritized experience replay* is more efficient than experience replay. This means that shorter training time is needed.

In experience replay, a certain number of experience tuples $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ is stored in a *replay memory* (also called replay buffer). For instance, the experiences for a sliding window of the past 50K timesteps might be stored, where these experiences might come from different episodes. The main idea of experience replay is to conduct weight updates based on experiences *sampled* from the replay buffer. In particular, each weight update of the Q-network is based on a *batch* of *uniformly sampled* experiences from this replay memory. For instance, each update might be based on a batch of 32 experience tuples. Note that weight updates may still be conducted at every single timestep t, each time with a new batch of experiences sampled from the replay buffer.

Prioritized experience replay makes two main adjustments to the experience replay method: first, experiences are not sampled from the replay buffer uniformly but instead with *probabilities corresponding to their importance* and second, the losses corresponding to sampled experiences that are to be used in a batch update are *weighted*.

The importance of an experience e_t is measured by the absolute value of the corresponding loss (see equation 2.31), also called TD-error for temporal difference error. For instance, the TD-error corresponding to experience e_t may be defined as $\epsilon_t = \frac{1}{2}\left[G_t - \hat{q}(S_t, A_t, \mathbf{w})\right]^2$. Schaul et al. suggest to sample the experiences from the replay buffer *not* uniformly but instead with the following probabilities:

$$P(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha}, \tag{2.34}$$

where t is the index of the experience and the sum in the denominator is over all experiences in the replay buffer. $p_t > 0$ is the priority of experience t and Schaul et al. suggest to define it as $p_t = 1/\operatorname{rank}(|\epsilon_t|)$. For example, the experience with the largest absolute TD-error would receive a rank of 1 (and hence a priority of 1), the experience with the second largest TD-error would receive rank 2 (and hence a priority of $1/2$) etc. The hyperparameter $\alpha \geqslant 0$ determines the extent of prioritization. Larger values of α prioritize experiences with large absolute TD-error even more and experiences with small absolute TD-error even less.

After sampling a batch of experiences, the TD-errors corresponding to these experiences are weighted *before* using them to update the Q-network. According to Schaul et al., the weighting is done to reduce bias: by sampling with the probabilities from equation 2.34, the experiences with larger TD-errors are more likely to be sampled and due to their large errors, these experiences would result in larger gradients and hence more extreme updates. This bias is accounted for by assigning weights to the TD-errors: smaller weights are given to TD-errors corresponding to experiences sampled with large probability $P(t)$ and larger weights are given to TD-errors corresponding to experiences sampled with small probability $P(t)$. The weights are defined as follows:

$$w_t = \left(\frac{1}{N} \cdot \frac{1}{P(t)}\right)^\beta, \tag{2.35}$$

where t is again the index of the sampled experience and N is the number of experiences in the replay memory. $\beta \in (0, 1]$ is a hyperparameter that is increased from some value < 1 until it reaches the value 1 at the end of the training period, at which point it fully accounts for the introduced bias. All weights are normalized before conducting the update: $w_t = 1/\max_t w_t$. The Q-network is then updated based on the batch of weighted TD-errors $w_t\epsilon_t$ corresponding to the sampled experiences for this update.

2.5.3 Dueling Networks

Dueling networks were introduced by Wang et al. in 2016 [26]. The idea of dueling networks is to adjust the architecture of the Q-network from figure 2.3, which represents the action-

value function as $\hat{q}(s, a, \mathbf{w}_q)$. Importantly, the Q-network still fulfills the same purpose after it is converted to a *dueling* version of the Q-network: it takes as input a state s and returns as output the estimated expected return $\hat{q}(s, a)$ for each possible discrete action a. The main aspect that changes is the architecture of the *inner* layers as described in the following.

According to Wang et al., the "dueling network should be understood as a single Q-network with two streams that replaces the popular single-stream Q network" [26, p. 2]. The first stream can be thought of as an individual network that predicts the state-value $v(s)$ (just like in figure 2.5). The second stream can be thought of as an individual network that predicts the advantage defined as $A(s, a) = q(s, a) - v(s)$. These two networks each have their own, separate hidden and output layers. However, the two networks are tied together: they share the same input layer (consisting of the state s) and their output layers are combined to a joint output layer consisting of the action-values $\hat{q}(s, a)$. Visibly, the outputs of the two streams, $\hat{A}(s, a)$ and $\hat{v}(s)$ can be combined to the joint output $\hat{q}(s, a)$ with the equation $\hat{q}(s, a) = \hat{v}(s) + \hat{A}(s, a)$. (The actual equation used is slighly more complicated as discussed in the original paper [26].) Because of the shared input layer and a combined output layer, the authors do not speak of two *separate* networks but instead of a *single* Q-network with two streams, the state-value stream and the advantage stream.

The main benefit of dueling networks is nicely explained by Wang et al. in [26]: the state-value stream receives an update whenever *any* action a is taken. For instance, if a particular action a_1 is taken, then the corresponding update does not only improve the predictions $\hat{q}(s, a_1)$ but also the predictions $\hat{q}(s, a_2)$ and $\hat{q}(s, a_3)$. This is because every weight update affects the output of the state-value stream, $\hat{v}(s)$, and the joint output $\hat{q}(s, a)$ is computed based on $\hat{v}(s)$ for all actions a. Ultimately, this may lead to better estimates $\hat{q}(s, a)$.

2.6 Reward Functions for Trading

The choice of the reward function is essential in reinforcement learning because the agent is trained to perform actions that maximize the expected return, where the return G_t is the sum of discounted future rewards as defined in equation 2.3. Consequently, by defining the reward function, we also define what we want the agent to learn. With different reward functions, the agent may learn to take different actions because a different objective needs to be optimized.

2.6.1 Relation of reward and target

Importantly, for a reinforcement learning system, we require that the agent receives a reward $R_{t+1} \in \mathbb{R}$ after every taken action A_t instead of only at the end of the episode. Recall that the foundation of reinforcement learning is the Markov decision process, which gives trajectories based on state, action and reward tuples (s, a, r, s'). Contrary to its name, the *reward* may not only represent positive but also negative or neutral feedback for the agent. The smaller the reward, the more punishing is the feedback signal. The larger the reward, the more rewarding is the feedback signal.

In practice, the reward is generally computed with a function that is chosen by the person designing the RL environment. How the reward function is constructed, depends strongly on the environment and the corresponding target of the agent. All actions A_t need to receive a reward R_{t+1} that is representative of how much they have contributed individually to the target. There are environments with a target metric that is only observed at the end of an episode, such as video games that can either be won or lost. In contrast, there are other environments with a continuous target that is accumulated across the whole episode, such as trading environments where profits are accumulated with trades throughout the trading day.

An example where target outcomes are observed at the end of the episode is the common simplistic reinforcement learning problem based on a *grid world* representing a maze. Essentially, the target of the agent is to find its way out of the maze. In such environments, the

reward for the terminal action, when the agent finds its way out of the maze, may take some positive value. All remaining rewards may take the exact same value, for instance a small negative value like -1 so that the agent learns to take fewer steps to find out of the maze.

An example where target outcomes are accumulated throughout the episode is the case of *trading* as in this thesis. Here, the target of the agent may be to maximize total profit per trading day. The total profit is accumulated by the agent across the trading day, with the agent buying and selling multiple times throughout the trading day. For instance, in a particular trade, the agent may sell some volume of electricity for a *higher* price than the purchase price. This would increase the total profit per day. In another particular trade, the agent may sell some volume of electricity for a *smaller* price than the purchase price. This would decrease the total profit per day. There are many possible functions for computing rewards for such actions, and they are discussed in the subsequent sections.

2.6.2 Target metrics

Before considering possible reward functions for the context of trading, we need to properly define possible target metrics of the agent. For instance, total profit per episode could be used as target metric. However, different target metrics are possible. We may not only want to achieve large profits but we may also want to avoid risky trades that could result in huge losses. Such risk aversion would need to be reflected in the target metric.

Moody et al. distinguish between three different categories of target metrics: a) risk-adjusted return, represented by the Sharpe ratio, b) profit and c) utility of profit [2]. Similar distinctions have been made in a literature review about reinforcement learning for trading in the financial domain where the categories were called Sharpe ratio, profit and other [15].

Interestingly, it has been stated that using the Sharpe ratio as target metric may generally result in larger cumulative profits than using profit itself as a target metric because the Sharpe ratio helps to avoid risky trades [2]. However, it is possible to avoid risk with utility of profit as target metric as well [28]. Consequently, a comparison between the different categories of target metrics and corresponding reward functions is a relevant research problem.

2.6.2.1 Sharpe ratio

The Sharpe ratio was first introduced as *reward-to-variability ratio* in 1966 in order to measure performance of mutual funds [29]. The motivation of this ratio is that large returns tend to be associated with larger variability (and therefore risk) than small returns. Since investors aim for large returns with small variability, some metric is needed that adjusts for the variability of returns, giving lower scores in case of large variability. To understand the Sharpe ratio, we first have to introduce the terms of equity $\mathcal{E}_t$ and return r_t [30].

In the context of the RL system, equity $\mathcal{E}_t \in \mathbb{R}$, defined in equation 2.36, is the total value of the cash and the volume of electricity owned by the agent at time t.

$$\mathcal{E}_t = C_t + Q_t \cdot P_t, \tag{2.36}$$

where C_t is the total cash owned by the agent, Q_t is the total volume of electricity owned by the agent and P_t is the market price per volume of electricity at time t.

Subsequently, the return $r_t \in \mathbb{R}$, defined in equation 2.37, is the absolute change of the equity from time $t-1$ to t. In some cases, the return is also defined as the percentage change instead of the absolute change in equity [30]. However, if specific amounts of volume are invested in each trade, it is common to define return as the absolute change in equity [2]. The absolute change in equity is shown in equation 2.37 below.

$$r_t = \mathcal{E}_t - \mathcal{E}_{t-1} \tag{2.37}$$

Finally, the Sharpe ratio in its simplest form, defined in equation 2.38, is the ratio of the sample mean and the sample standard deviation of returns [2].

$$\mathcal{S}_T = \frac{\bar{r}}{s_r}, \tag{2.38}$$

where the estimates $\bar{r}$ and s_r are computed based on returns r_t with $t \in 1, ..., T$ as described in equations 2.39 and 2.40. Note that equation 2.40 shows the unbiased estimator of the standard deviation but the biased estimator is sufficient for reinforcement learning [2].

$$\bar{r} = \frac{1}{T} \sum_{t=1}^{T} r_t \tag{2.39}$$

$$s_r = \sqrt{\frac{1}{T-1} \sum_{t=1}^{T} (r_t - \bar{r})^2} \tag{2.40}$$

2.6.2.2 Profit

The total profit for an episode from $t = 1, ..., T$, defined in equation 2.41, is the difference between the equity at the end of the episode and the equity at the beginning of the episode.

$$\mathcal{P}_T = \mathcal{E}_T - \mathcal{E}_0 \tag{2.41}$$

Alternatively, one might express the profit in terms of the traded volumes and their respective prices at each step t of the episode, as in equation 2.42.

$$\mathcal{P}_T = \sum_{t=0}^{T-1} \left[V_t^s \cdot P_t - V_t^b \cdot P_t \right], \tag{2.42}$$

where V_t^s and V_t^b represent the volume sold and bought by the agent at time t respectively, and P_t represents the market price per volume at which electricity was sold and bought. Note that the volumes V_t^s and V_t^b are based on the actions taken by the agent, whereas P_t is given by the market. Further note that at a specific time t, the agent can either buy or sell or do none of the two. Hence, at least one of the two volume terms, V_t^s and V_t^b, has to be zero for every t.

2.6.2.3 Utility of profit

In equation 2.41, profit is displayed *as is*. This is equivalent to applying a risk-neutral utility function because no transformation is conducted to account for risk. However, we may also apply different utility functions. For the context of trading, risk-avoiding utility functions are of particular interest. Risk-avoiding utility functions have a concave shape, whereas risk-neutral utility functions have a linearly shape [30]. This is illustrated with an example in figure 2.6. With the risk-neutral utility function, every additional unit of profit results in the same additional unit of utility. With the risk-avoiding utility function, the additional amount of utility resulting from an additional unit of profit depends on the value of the profit: the smaller the profit, the larger is the utility of an additional unit of profit. In other words, the largest gain in utility is achieved by avoiding very large losses.

Convex-shaped utility functions can, for instance, be constructed by using logarithms or powers of < 1. For this thesis, the natural logarithm is used for the utility function, as shown in equation 2.43 below.

$$\mathcal{U}_T = \ln\left(\frac{\mathcal{E}_T}{\mathcal{E}_0}\right), \tag{2.43}$$

where $\mathcal{E}_T$ is the equity at the end of the episode and $\mathcal{E}_0$ is the equity at the beginning of the episode.

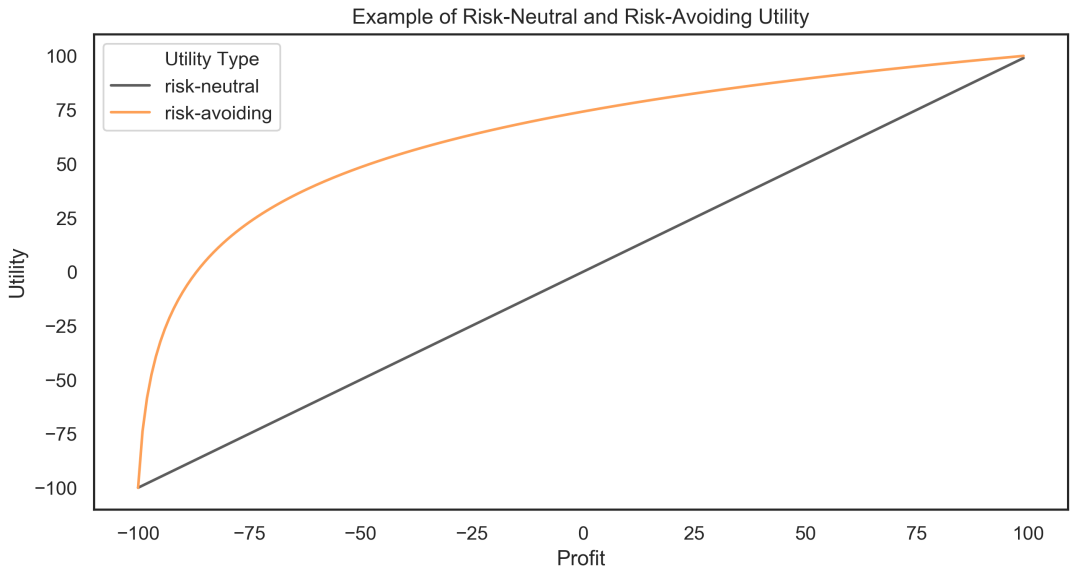


Figure 2.6: Example of risk-neutral and risk-avoiding utility

2.6.3 Reward functions

After defining the target metrics, Sharpe ratio $\mathcal{S}_T$, profit $\mathcal{P}_T$ and utility of profit $\mathcal{U}_T$, this section reviews possible reward functions based on these target metrics. By saying that a reward function is *based* on a respective target metric, what is meant here is that the reward function is similar in nature to the target metric.

This means that the reward function strongly resembles the target metric but that it may not result in policies that maximize the target metric. For the target metric $\mathcal{U}_T = \ln\left(\frac{\mathcal{E}_T}{\mathcal{E}_0}\right)$ for instance, the reward function may be defined as $R_{t+1} = \ln\left(\frac{\mathcal{E}_{t+1}}{\mathcal{E}_t}\right)$. Recall that the RL agent learns to take actions that maximize the return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T$. Visibly, the return G_t differs from the target metric $\mathcal{U}_T$ and hence, the actions taken by the agent may not maximize the target metric $\mathcal{U}_T$. This example also illustrates that designing reward functions in reinforcement learning is not a trivial problem: it may be difficult to find an optimal reward function for a specific target metric because there may be some detachment between return G_t (the sum of discounted rewards) and the target metric.

2.6.3.1 Reward based on Sharpe ratio

Moody et al. suggest to optimize a variant of the Sharpe ratio, namely the differential Sharpe ratio, because it can be differentiated with respect to a single observation of return r_t, allowing for online optimization in their recurrent reinforcement learning algorithm [2], [31]. However, for other RL algorithms, what is used in practice is instead a running average of the Sharpe ratio, as defined in equation 2.44. This is essentially the Sharpe ratio computed with returns from the current and from the last L timesteps. For instance, it was applied in [32], [33].

$$R_{t+1} = \frac{\bar{r}^{(t+1-L):(t+1)}}{s_r^{(t+1-L):(t+1)}}, \tag{2.44}$$

where the estimates $\bar{r}$ and s_r are computed based on returns r_k with $k \in t+1-L, ..., t+1$ as shown below.

$$\bar{r} = \frac{1}{L} \sum_{k=t+1-L}^{t+1} r_k \tag{2.45}$$

$$s_r = \sqrt{\frac{1}{L-1} \sum_{k=t+1-L}^{t+1} (r_k - \bar{r})^2} \tag{2.46}$$

2.6.3.2 Reward based on profit

Absolute return
The maybe most simple profit based reward function, defined in equation 2.47, is the difference in equity between time $t+1$ and t [34], [35]. Note that this is the equivalent of the return in equation 2.37, with the difference that the index has changed since rewards are generally indexed by $t+1$.

$$R_{t+1} = \mathcal{E}_{t+1} - \mathcal{E}_t \tag{2.47}$$

VWAP delta
Another possible metric is the difference between the volume weighted average sell price and the volume weighted average buy price, defined in equation 2.48. This metric does not come from literature. Instead, it is suggested by the author and the commissioner.

$$R_{t+1} = \frac{\sum_{k=0}^{t}[V_k^s \cdot P_k]}{\sum_{k=0}^{t} V_k^s} - \frac{\sum_{k=0}^{t}[V_k^b \cdot P_k]}{\sum_{k=0}^{t} V_k^b}, \tag{2.48}$$

where V_k^s and V_k^b represent the volume sold and bought by the agent at time k respectively, and P_k represents the market price per volume at which electricity was sold and bought.

2.6.3.3 Reward based on utility of profit

The risk-avoiding reward function used for this thesis resembles the corresponding target metric from equation 2.43. The difference is that the ratio $\mathcal{E}_{t+1}/\mathcal{E}_t$ is used instead of $\mathcal{E}_T/\mathcal{E}_0$. This reward function has been suggested because it "penalizes losses stronger than gains" [28, p. 942]. Several studies, for instance [32], [36], have made use of this reward function or a variation of it.

$$R_{t+1} = \ln\left(\frac{\mathcal{E}_{t+1}}{\mathcal{E}_t}\right) \tag{2.49}$$

3 Literature Review

This chapter gives an overview about the electricity market, electricity price forecasts and the use of reinforcement learning for trading in both, the electricity and the financial market. Section 3.1 summarizes relevant characteristics of the electricity market. Section 3.2 gives an overview about the type of models that perform well for forecasting electricity prices. Sections 3.3 and 3.4 give an overview about how reinforcement learning has been applied for trading in the electricity and in the financial domain respectively.

Unfortunately, a prevalent issue in the field of price forecasting and reinforcement learning for trading is the lack of comparability across different studies. In particular in the field of reinforcement learning, different studies use different environments, data sets, evaluation metrics and algorithm implementations. For instance, no paper was found that addresses the problem of trading in the continuous hourly intra-day electricity market in the way it was done in this thesis. Hence, it is difficult to use quantitative results from other studies as reference or performance baseline. For this reason, this literature review rather focuses on giving a broad overview instead of detailed insights.

3.1 Properties of the Electricity Market

The electricity market differs from other markets such as the financial market (where assets such as stocks, indices or funds are traded) or commodity markets (where assets such as gold, steel or corn are traded). The peculiarities of the electricity market have been greatly summarized in other papers, such as [37]. In particular, this paper mentions *two milestones* strongly affecting the electricity market (liberalization of the electricity supply chain and transition to renewable energies) and, among others, *three relevant properties* of the electricity market (balance of supply and demand, dependence of supply on weather conditions and dependence of demand on time). These milestones and properties are explained shortly below.

The *liberalization* of the electricity market in Europe began in the 1990s and lasted until the 2010s [38]. By breaking up monopolies and allowing competing private companies to partake in the electricity supply chain, this milestone shaped the electricity market as it is today and allowed electricity to be traded in the way it is traded for this thesis. The *shift towards renewable energies* such as wind energy, changing the supply of electricity, contributed to more volatile electricity prices due to the dependence of electricity prices on external factors like wind [39].

Importantly, electricity supply always needs to be equal to electricity demand, i.e. the amount of electricity produced needs to correspond to the amount of electricity consumed. This is a requirement of the power grid, through which electricity is delivered from producers to consumers. Differences in supply and demand result in unstable electric frequencies in the power grid, which in turn may result in power plants getting disconnected from the power grid [40]. Essentially, electricity prices help to balance out supply and demand. If demand exceeds supply, prices become expensive, which then reduces demand since consumers do not want to buy electricity at expensive prices. If supply exceeds demand, prices become negative, which then reduces supply since producers want to get paid (instead of having to pay) for feeding electricity into the power grid [41]. Note that it is *not* possible to store large amounts of electricity in order to sell this electricity later when there is more demand than supply [42]. In conclusion, changes in supply (e.g. more produced wind energy) and in demand (e.g. more electricity usage when people are at home) can affect electricity prices.

Both, demand and supply, are volatile and difficult to predict. On the demand side, it is important to note that electricity is a product that is actively used by people, organizations etc. There are certain time periods when people may use more electricity, for instance when they come home in the evenings [43]. Therefore, electricity demand is dependent on the time period. On the supply side, there exist electricity sources which depend on external factors such as wind energy. Wind strength can neither be controlled, nor predicted perfectly [39]. Importantly, the price volatility resulting from variations in supply and demand creates both, risk and opportunity, for trading electricity.

3.2 Electricity Price Forecasting

A review about electricity price forecasting (EPF) from 2014 raises the issue that EPF studies are difficult to compare due to different data sets, error measures and software implementations [44]. As a result, studies may even give conflicting results. Hence, one may not be able to conclude whether a particular prediction model used in one study is more suitable than a different prediction model used in a different study.

A later study from 2018, however, thoroughly benchmarked 27 different models for predicting hourly day-ahead electricity prices. It found that a deep neural network (DNN) significantly outperformed all other models, including other neural networks with long-short term memory (LSTM) and gated recurrent units (GRU) [37]. The DNN applied in this study consisted of two hidden layers with 239 and 162 neurons respectively. The neural networks with LSTM and GRU, however, also significantly outperformed the remaining models, showing that neural networks in general seem to perform well on this task. In addition, the study found that models based on machine learning (e.g. support vector machines, neural networks, gradient boosting trees) significantly outperformed models based on time series models (e.g. ARIMA, DSARIMA, GARCH).

Importantly, this study conducted forecasts for the hourly day-ahead electricity market and not for the hourly intra-day electricity market, which is the subject of this thesis. What both markets have in common is that they are based on 24 instruments, one for each hour of the day. However, in the day-ahead market, the orders for all 24 instruments are submitted at once [45]. This means that the day-ahead market differs noticeably from the intra-day electricity market, where orders are submitted continuously throughout the day. In addition, intra-day prices may be more volatile than day-ahead prices [46]. Unfortunately, there are seemingly no papers that benchmark EPF models on the intra-day electricity market.

Although there exist studies that conducted forecasts for the intra-day electricity market, these studies generally do not aim to achieve good predictions or to compare model performance. Instead, their objective is to study certain effects and characteristics of the electricity market in general [47]–[52]. For this purpose, most of these studies apply time series models such as GARCH.

3.3 Reinforcement Learning in the Electricity Domain

Reinforcement learning has been applied in the electricity domain for various different tasks. For instance, it has been used to train agents to buy or sell electricity as in the optimal trade execution (OTE) problem and related problems [9]–[11], [53], [54], to model overall market behavior [55]–[57], to train electricity broker agents in smart grids [58], [59], to train agents to create electricity consumption plans for charging electric vehicles [60] and to train storage device trading agents [14]. The latter resembles the scenario of this thesis in the sense that both, buy and sell actions, are possible. In this thesis however, the agent cannot physically store electricity. Instead it can only own electricity contracts, which must be sold before the delivery time period. In contrast, storage devices allow to store electricity even until after the delivery time period.

Generally, RL methods from the field of function approximation have been used due to continuous state spaces and possibly continuous action spaces. For instance, *actor-only* methods were applied in [9], [53] and *critic-only* methods were applied in [11], [14], [59], [60]. Also, *actor-critic* methods have been applied in the the electricity domain, mostly for understanding and modeling electricity markets e.g. in [55]–[57]. Very recently in 2020, the actor-critic algorithm A3C has also been successfully applied for electricity trading in [54]. In particular, it was used for wind power producers to sell electricity but also to buy electricity reserves to avoid penalties in case of e.g. too little wind energy production. Actor-critic methods have the main advantage of a more stable training process compared to *actor-only* methods [17]. With scalable implementations, such as A2C and A3C [23] [4], these methods seem generally promising for the problem of trading electricity.

3.4 Reinforcement Learning in the Financial Domain

The problem of trading electricity addressed in this thesis resembles the problem of trading financial products. Generally, the problem is to generate profits by buying *and* selling assets (either electricity or financial products). Since this problem has been covered more extensively in the financial domain than in the electricity domain, the literature about reinforcement learning for trading financial products is reviewed shortly.

In the financial domain, reinforcement learning has received noticeable attention for trading, in particular for trading indices [1], [2], [13], [61]–[65], currencies [16], [31], [36], [66]–[68], stocks [32]–[34], [69], [70] and other products such as derivatives [8], [71], [72]. While most papers address the problem of trading single assets (e.g. one single stock or index), some papers also address the problem of portfolio management, where multiple assets are traded.

Just like in the electricity domain, methods from the field of function approximation are generally used in the financial domain due to continuous state spaces and just like in the electricity domain, the focus has been on critic-only and actor-only methods rather than on actor-critic methods. For instance, *critic-only* methods were applied in [1], [8], [13], [16], [32]–[34], [61], [66]–[69], [73], *actor-only* methods were applied in [1], [2], [8], [13], [31], [36], [61]–[65], [71]–[73] and *actor-critic* methods were applied in [8], [65], [70].

One particular algorithm that is commonly used for trading in the financial domain is called *recurrent reinforcement learning* or *direct reinforcement learning* algorithm. This algorithm has tradition because it was one of the first RL methods applied to the problem of trading in the financial domain by Moody et al. in [1]. It is generally considered to be an actor-only algorithm because it directly learns a policy [15], [62]. However, it does not make use of the policy gradient theorem and it also does not result in a stochastic policy. Instead, there is a neural network with tanh function in the output layer to map predictions into long or short positions of fixed volume, represented by values 1 and -1 respectively. The network is trained with gradient ascent to maximize some utility function. This method was not further considered in this thesis because of its focus on the financial domain where short positions

are possible (unlike in the electricity domain) and because it is designed for placing trades of fixed volume (instead of different volumes).

A literature review about trading with reinforcement learning from 2018 concluded that "there is no comprehensive study benchmarking all three approaches, i.e. actor-only, critic-only and actor-critic" [15, p. 36]. Further, this review concluded that based on existing papers it was not possible to judge whether one of the three method is generally better than other methods. Later in 2019, Zhang et al. applied RL methods from all three categories for trading in the financial domain [8]. They applied these methods on different static data sets, historic market price time series of stocks, indices, currencies and more. Critic-only methods were represented by DQN, actor-only methods by REINFORCE and actor-critic methods by A2C. Unfortunately, this paper does not focus on thoroughly comparing the performance of these methods with each other. It only does so briefly, saying that "DQN obtains the best performance among all models and the second best is the A2C approach" [8, p. 8]. However, the reported results show that the conclusions depend on the data sets and on the evaluation metrics. Importantly, no hypothesis tests were used and no hyperparameter optimization was conducted for the different methods.

4 Data

This chapter introduces the data used in this thesis. Section 4.1 describes how the raw order book data was converted to market data. Section 4.2 describes which features were constructed based on the market data. Section 4.3, describes how the data was split into training, validation and test data.

The price from the market data described in section 4.1 is the price that was traded by the RL agent. Therefore, it is essential for computing the rewards and the evaluation metrics (introduced later on in the methods section). The features described in section 4.2 were used to represent the state space, meaning that the RL agent used these features for decision making.

4.1 Data Preparation

4.1.1 Order Book Data

The raw data consists of order book data. Essentially, the order book data represents the overall market in the past. In this thesis, the RL agent learns to trade on this historic market. Before training the RL agent, the raw order book data is, however, transformed as described in section 4.1.2. Note that none of the agent's actions are in the order book data. Instead, the order book data contains orders that were actually placed by market participants in the past, e.g. electricity producers and consumers.

In particular, there are two order books: one with sell orders and one with buy orders. One order basically says that a market participant wants to either buy or sell a certain volume of electricity for a certain price. Not all orders get executed. When an order gets executed, this means that an actual trade takes place. Simply speaking, a buy order with a certain price gets executed if there is a sell order with the same or a smaller price and the same or a larger volume. If a buy order is executed, there is always a corresponding sell order that is executed as well. However, an executed buy order can also correspond to several executed sell orders and vice versa. This is illustrated in the schematic figure 4.1, which shows an excerpt of the buy and sell order book. In the figure and throughout the thesis, volume is measured in megawatt hours (MWh) and price is measured in EUR. The arrows in the diagram illustrate which buy and sell orders were executed together and thereby converted to a trade. Hereby, buy and sell orders come from various different market participants. Note that the data

displayed in the diagram is simplified and does not correspond to the actual data used in the thesis.

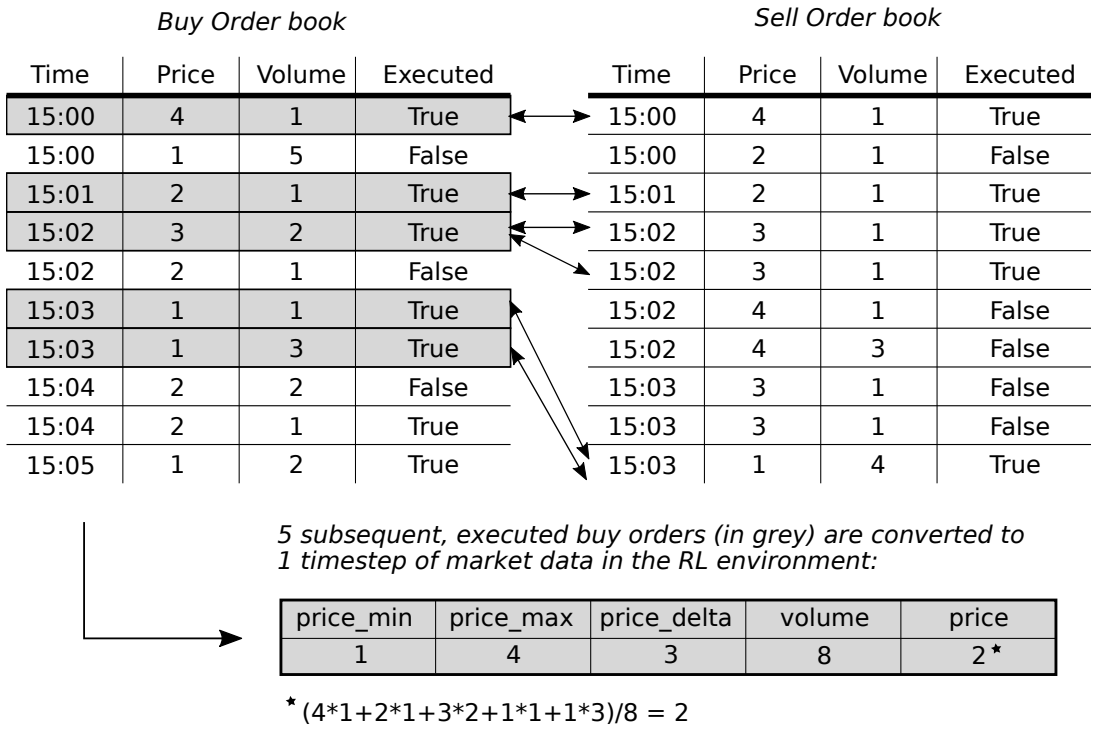


Figure 4.1: Diagram of order book data. Volume is measured in megawatt hours (MWh) and price is measured in EUR. This schematic diagram illustrates that buy and sell orders get executed if they match in their price and volume. One buy order can get executed with several sell orders and vice versa. The 5 subsequent executed buy orders highlighted in grey would be aggregated to 1 time step in the RL environment.

4.1.2 Market Data

In this thesis, the agent can buy or sell some volume of electricity at a certain market price. Whenever the agent chooses to buy or sell a certain volume of electricity, the corresponding volume is bought or sold at market price. In other words, all orders of the agent get executed, i.e. converted to a trade. This is a simplification of the real-world scenario, where the agent does not only need to specify the volume to buy or sell but also the desired price at which to buy or sell when placing an order. In the real-world scenario, a buy order only gets executed if there exist corresponding sell orders that match the price and the volume of the buy order. Reversely, a sell order only gets executed if there exist corresponding buy orders that match the volume and the price of the sell order.

Since orders in this thesis are executed at market price, this price is relevant for the computation of the rewards and evaluation metrics such as profit. The market price, along with other market data, was obtained by summarizing every 5 subsequent executed buy orders from the raw order book data. In other words, every timestep t in the RL environment contains information about a bin of 5 trades, representing the market in their corresponding time interval. For instance, the 5 buy orders highlighted in grey in figure 4.1 would be converted to 1 timestep of market data in the RL environment. Note that only the buy order book was considered (and not the sell order book) since executed buy orders are also represented as

executed sell orders in the sell order book. Instead of using the buy order book, the sell order book could have been used alternatively.

Each bin of 5 executed buy orders was summarized by computing the following features: the minimum price, the maximum price, the delta between the minimum and the maximum price, the total volume and the volume weighted average price. These features are also described in table 4.1 and an example computation is shown in figure 4.1. Note that the volume weighted average price for a bin of 5 orders is from here on only called *price* or *market price*. It is the price that is traded by the RL agent. For reference, the formula can be found below.

$$\text{price}_\text{t} = \frac{\sum_{j \in \{t_{first},...,t_{last}\}} V_j \cdot P_j}{\sum_{j \in \{t_{first},...,t_{last}\}} V_j}, \tag{4.1}$$

where V_j and P_j are volume and price of order j respectively. Each sum is computed with 5 orders, indexed by j, that fall into the bin represented by timestep t in the RL environment.

4.2 Feature Engineering

Various features were created for the state space. These features can be categorized into three types: market features (e.g. price and volume), indicator features (e.g. simple moving average and exponential moving average) and time features (e.g. hour of the day and time until delivery). An overview of the features can be found in table 4.1.

4.2.1 Feature Overview

The market features, which were mentioned before in section 4.1.2, basically summarize the 5 executed orders at each timestep t. The indicator features are computed based on several timesteps of market feature data and hence contain information about the market across time. In the financial domain, such features are generally referred to as technical indicators, which are commonly used for decision making in trading [80]. They have also been applied in the context of reinforcement learning for trading [8], [15]. Lastly, the time features are valuable because of differences in price volatility across the time of the day (e.g. larger volatility in the evenings when electricity consumption is high).

All features were scaled in some way. Time features were scaled differently than market and indicator features. Time features were scaled individually prior to training into the range [0, 1] by dividing them with their respective maximum values (e.g. hour of the day was divided by 24). To understand how the market and indicator features were scaled, the state space of the agent needs to be described shortly: all features were provided to the agent in form of a sliding window. For research question 1 for instance, the state space contained information about all 21 features for 10 timesteps (9 past timesteps $t-9, t-8, ..., t-1$ and the current timestep t). This means that the state space was a matrix $\mathbf{S}_{10\times 21}$, where 10 represents the number of timesteps and 21 the the number of features. In other words, the state space consisted of 210 values, implying an input dimension $D = 210$ in figures 2.4 and 2.5. During training, the market and indicator features were converted to percentage changes relative to their respective values at the current timestep t. For each of the market and indicator features, the percentage changes from timesteps $t-9, t-8, ..., t-1, t$ to timestep t were computed respectively. This was done at every timestep t. The resulting percentage changes were restricted to the range $[-10, 10]$. For instance, values larger than 10, representing percentage increases of more than 1000%, were set to 10. Subsequently, all values were divided by 10 to scale the values into the range [-1, 1].

A correlation plot of the scaled features is displayed in figure 4.2. Visibly, the features that are computed based on the price feature (most indicators) show relatively strong correlations. However, also the time features show some small correlation with the other features. The distributions of market and indicator features *before* scaling are visualized in figures A.2 and A.3.

Table 4.1: Feature overview

Index	Feature	Type	Description
1	price	market	Volume weighted average price of the executed orders in the bin of timestep t
2	price_min	market	Minimum price of the executed orders in the bin of timestep t
3	price_max	market	Maximum price of the executed orders in the bin of timestep t
4	price_delta	market	Difference btw. maximum and minimum price (based on 2, 3)
5	volume	market	Total volume of the executed orders in the bin of timestep t
6	VWAP	indicator	Volume weighted average price of executed orders from time 1 to t (based on 1, 5) [74]
7	SMA5	indicator	Simple moving average of the prices from the last 5 timesteps (based on 1) [75]
8	SMA10	indicator	Simple moving average of the prices from the last 10 timesteps (based on 1) [75]
9	EMA5	indicator	Exponential moving average of the prices from the last 5 timesteps (based on 1) [76]
10	EMA10	indicator	Exponential moving average of the prices from the last 10 timesteps (based on 1) [76]
11	bollinger10_hband	indicator	Upper bollinger bands based on 10 timesteps using factor of 2 (based on 1, 8) [77]
12	bollinger10_lband	indicator	Lower bollinger bands based on 10 timesteps using factor of 2 (based on 1, 8) [77]
13	MACD	indicator	Moving average convergence/divergence, i.e. EMA12 - EMA 26 (based on 1) [78]
14	RSI	indicator	Relative strength index based on the prices from the last 10 timesteps (based on 1) [79]
15	time_e_month	time	Month of the year / 12
16	time_e_weekday	time	Weekday / 7
17	time_e_hour	time	Hour of the day / 24 (when the orders in bin of timestep t are executed)
18	time_instrument	time	Hour of the day / 24 (when the traded electricity is to be delivered)
19	time_delta_delivery_s	time	Mean time delta (in unix time) btw. order filing and delivery (divided by max. value)
20	time_delta_delivery_e	time	Mean time delta (in unix time) btw. order execution and delivery (divided by max. value)
21	time_delta_e_s	time	Mean time delta (in unix time) btw. order filing and execution (divided by max. value)

Furthermore, the distributions of market and indicator features *after* scaling are visualized in figures A.4 and A.5 respectively.

4.2.2 Encoding Instruments

4.2.2.1 Instruments and Contracts

An important aspect to consider is that on the hourly intra-day electricity market, instead of trading one product, 24 different products are traded: one product for each hour of the day.

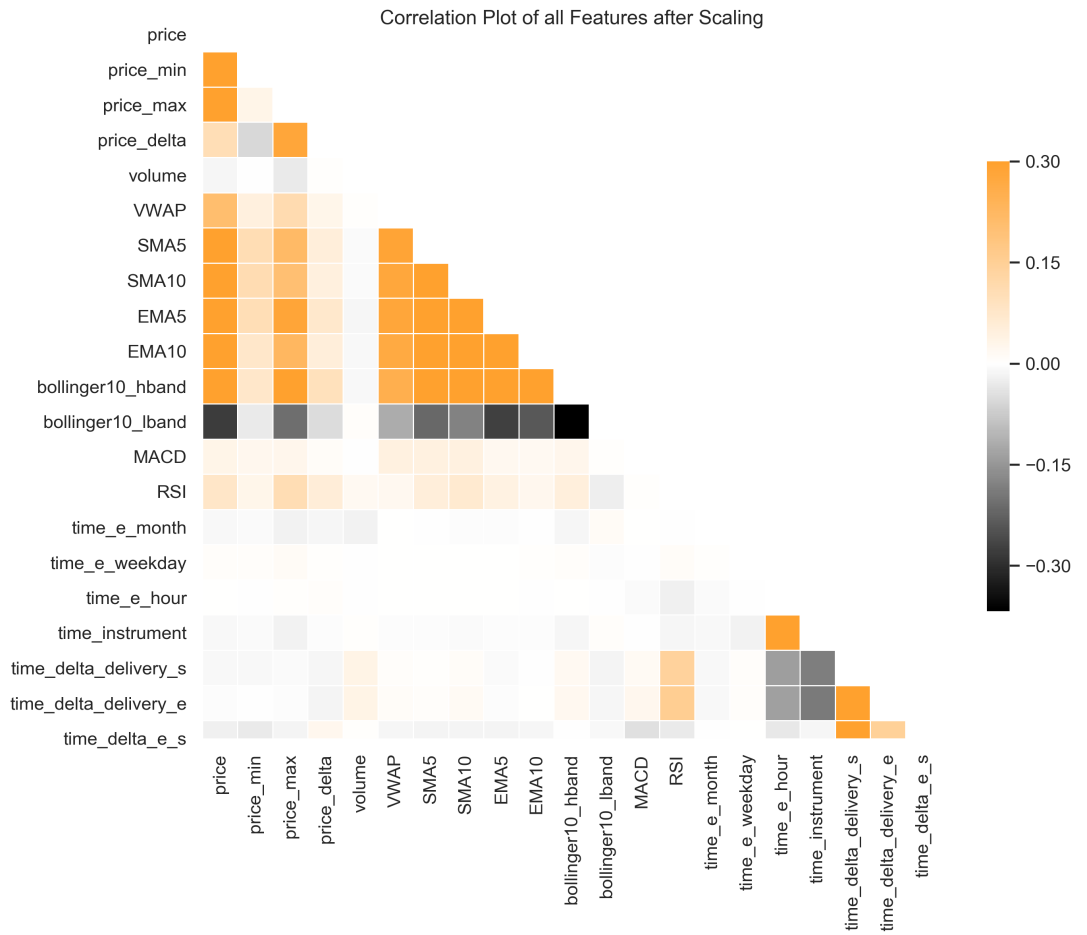


Figure 4.2: Correlation plot of all features after scaling

In other words, electricity producers and consumers sell and buy electricity for a specific hour in the future. The hourly products are also called *instruments*, meaning that an instrument is equivalent to a specific hour of the day.

Instruments are traded for particular dates since electricity is used at a particular hour on a particular date. The pair of an instrument and a date is called *contract*. For instance, one contract could represent electricity to be delivered at 20:00 on June 30, 2016. In other words, this contract corresponds to the 20:00 *instrument* to be delivered on June 30, 2016. In the context of RL, every contract corresponds to one episode. Essentially, the features from table 4.1 were computed for every contract *separately*, based on orders for this contract.

Each contract is traded within a single, fixed time period. When this time period is over, the electricity corresponding to the contract is delivered to the consumer. Afterwards, the same contract can never be traded again since a particular combination of hour and date can only occur once. The intra-day market for the a certain date opens at 15:00 on the *previous* date. This means that all 24 instruments of a certain date can be traded from 15:00 on the previous date until an hour before delivery of the respective instrument. Therefore, every instrument can be traded for a different number of hours. The instrument corresponding to the first hour of the day can be traded for the least amount of hours and the instrument corresponding to the last hour of the day can be traded for the largest amount of hours.

4.2.2.2 Differences between Instruments

Two main differences between instruments can be found in the amount of traded volume and in the volatility of prices. This is largely based on the nature of electricity: it is a product that is consumed by people, organizations etc. As described in section 3.1, unexpected changes in demand and supply affect electricity prices. Importantly, there are time periods when electricity demand tends to be larger and more volatile. For instance, demand is much larger and more volatile in the evenings (e.g. at 20:00) when many people use electricity than at night (e.g. at 04:00) when most people sleep.

This is reflected by the corresponding instruments: electricity to be delivered in the evenings (corresponding to e.g. the 20:00 instrument) is generally traded at a much larger volume and with more volatile prices than electricity to be delivered at night (corresponding to e.g. the 04:00 instrument). Figure 4.3 illustrates this. It shows the distribution of unscaled prices by instrument. For example, one can see that the prices for the 04:00 instrument have smaller extreme values and smaller variation than the prices for the 20:00 instrument.

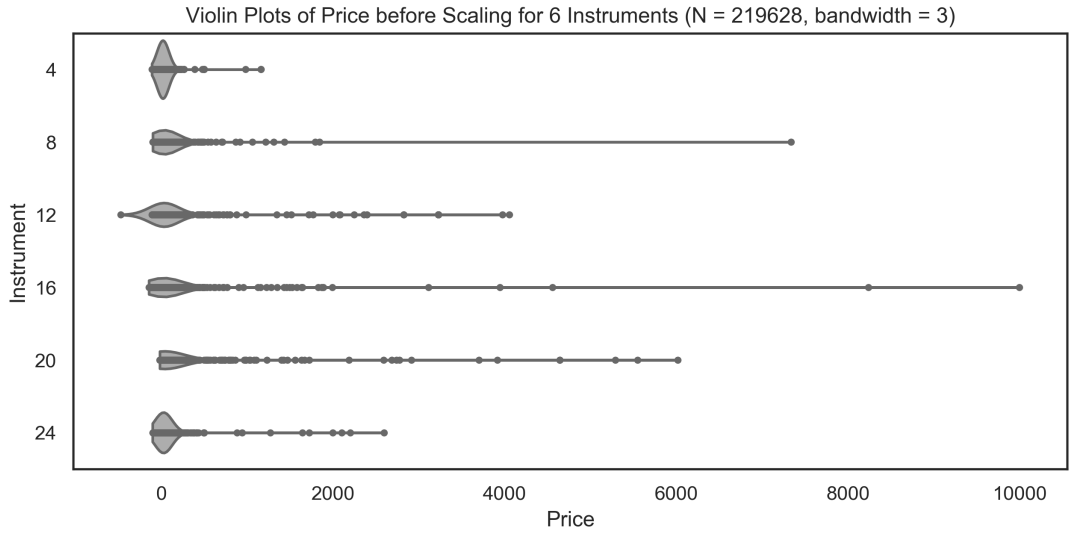


Figure 4.3: Violin plots of price before scaling for 6 instruments

4.2.2.3 Encoding different Instruments

The question is whether the RL agent should be able to choose which instrument to trade, or whether it should be given which instrument to trade. While it is possible to trade different contracts simultaneously, letting the agent choose between trading e.g. the 04:00 and the 20:00 instrument, this was not done for this thesis. Instead, the purpose of this thesis was to create an agent that can potentially trade any of the 24 products, but only one at a time. For instance, the agent should be profitable when instructed to trade the 20:00 instrument in one episode and it should also be profitable when instructed to trade the 04:00 instrument in another episode. There are two possible approaches for this.

Either 24 individual agents are trained on the different instruments respectively, or one agent is trained on all 24 instruments and the instrument is encoded in the state space. The second approach is chosen in this thesis because of its advantage that a) only *one* agent needs to be trained (instead of 24 agents) and that b) this agent can learn from a large amount of data, namely from data of all 24 instruments (instead of just one instrument). While the electricity market may differ across instruments, for instance in price volatility and amount of traded volume, the agent can learn to adjust to these differences because the instrument

is encoded in the state space and therefore known by the agent. As shown in table 4.1, the instrument is encoded as the time_instrument feature in the state space.

4.3 Data Split

The data set was split into data sets for training, validation and testing. The split was conducted by time instead of randomly in order to obtain realistic estimates about how the agent would generalize to future data. Also in real-world applications, future data may differ from training and validation data due to changes in the electricity market. The time period from January 2014 to February 2016 was used for training, the time period from March 2016 to June 2016 was used for validation and the time period from July 2016 to December 2016 was used for testing. In terms of timesteps that can be taken by an agent, this corresponds to a 60%, 21% and 19% split for training, validation and test data respectively as displayed in table 4.2. This table contains information across all contracts (and hence also across all instruments). As mentioned in section 4.2.2, one episode in the RL environment corresponds to one contract. Episodes with fewer than 10 possible timesteps were removed. More information about how the data sets were used can be found in the method chapter 5.

Table 4.2: Overview of training, testing and validation data

Type	Time Period	Total Steps	% of Steps	Total Episodes	% of Episodes
Training	Jan 2014 - Feb 2016	503,371	59.8	12,557	66.4
Validation	Mar 2016 - Jun 2016	161,173	19.2	2,887	15.3
Testing	Jul 2016 - Dec 2016	176,600	21.0	3,481	18.4
Total	Jan 2014 - Dec 2016	841,144	100.0	18,925	100.0

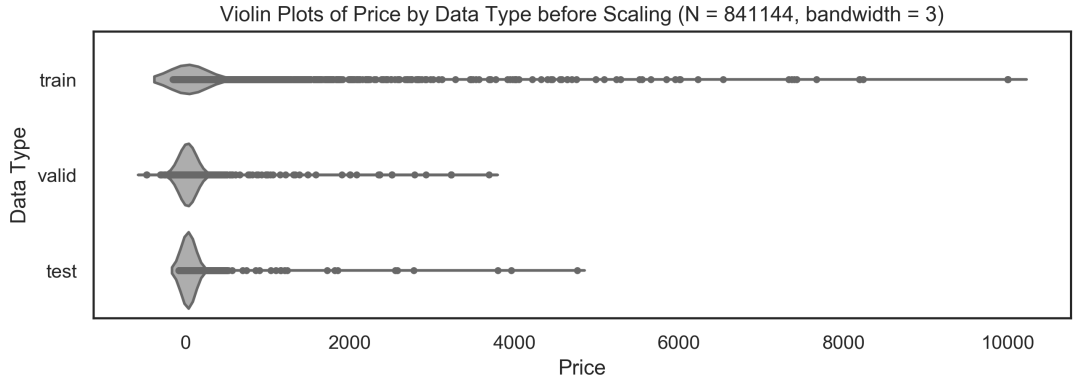


Figure 4.4: Violin plots of price before scaling by data type

As stated previously, the electricity market does change over time. The reader may therefore wonder how the three data sets differ with respect to their distributions of the market price (before scaling). The distribution of the market price is of relevance because the RL agent trades the market price and therefore the performance of the RL agent also depends on the market price distribution. For illustration, the price distributions are illustrated visually in figure 4.4 across all instruments. Visibly, there are noticeable differences in terms of mean, standard deviation and extreme values. These differences put a limitation on the results to be achieved by the RL agent in this thesis. However, to keep the evaluation realistic, the choice was made not to adjust for differences in prices since this could also not be done in a real-life application. Splitting the data by time, as in this thesis, is also how it was done by a study that thoroughly benchmarked models for EPF [81].

5 Method

This chapter describes the methodology applied in this thesis. To begin with, section 5.1 summarizes the RL environment, section 5.2 introduces the evaluation metrics (for evaluating the conducted experiments), section 5.3 describes the approach for hyperparameter tuning and feature selection, section 5.4 introduces the baseline strategies (for benchmarking the RL agent) and section 5.5 describes how hypothesis tests were conducted.

The subsequent sections 5.6, 5.8 and 5.7 particularly summarize how training and evaluation were conducted for the research questions: 1) reward function comparison, 2) method comparison and 3) baseline strategy comparison. An overview of RL method, reward function, features and hyperparameters used for these research questions is shown in table 5.1.

Table 5.1: Overview of methods by research question

Question	Topic	Method	Reward	Features	Hyperparameters
1.	Reward	A2C	All	All	Default
2.	Method	A2C and DQN	Optimized	Optimized	Optimized
3.	Baseline	A2C	Optimized	Optimized	Optimized

Moreover, section A.5 in the appendix summarizes the training metrics (for evaluating the training progress) and section A.6 in the appendix describes how episodes were categorized according to certain price patterns (spikes, uptrend and downtrend).

5.1 Environment

The first step in reinforcement learning is to define the environment with which the agent is supposed to interact. Among others, state space, action space and reward function need to be defined. The state space represents the information available to the agent for decision making. The action space contains the possible actions that the agent can take at each time step. The reward function, which is *not* known to the agent but only to the environment, determines which feedback the agent gets for its actions. The purpose of this section is to give a short summary of the RL environment used in this thesis. In particular, episodes, action space, state space and reward functions are described.

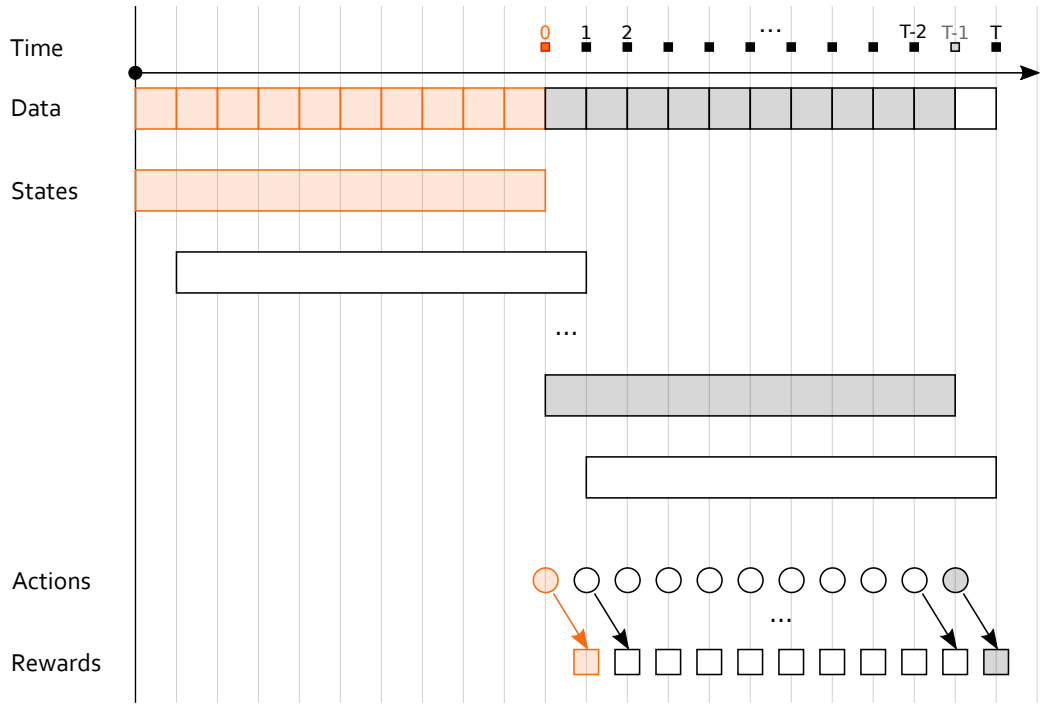


Figure 5.1: Diagram of RL environment for one episode, where states are based on 10 time steps. In orange, the first state S_0, action A_0 and reward R_1 are highlighted. In grey, the state S_{T-1} and the last action A_{T-1} and last reward R_T of the episode are highlighted.

5.1.1 Episodes

In this thesis, the interaction of the agent with the environment is structured into separate logical units, called episodes. Each episode corresponds to a particular contract that is traded by the agent. A contract, as described in section 4.2.2, represents electricity to be delivered at a specific date and hour (also called *instrument*). The states of an episode are computed based on the order book data for the corresponding contract as described in chapter 4.

The length T of an episode, i.e. the number of steps that can be taken by the agent during the episode, depends on the number of executed buy orders for the corresponding contract (in the order book data). Since the number of executed buy orders per contract differs, the episodes also differ in their lengths. The market data of every timestep t is based on 5 executed orders as described in section 4.1.2. Therefore, if there were for instance 105 executed buy orders for a particular contract, then there would be 21 potential timesteps for the corresponding episode. However, because the state space is based on a window of past timesteps as mentioned before in section 4.2.1, the agent cannot start trading when the market opens. Instead, it has to wait until sufficient timesteps have passed so that the state space can be properly represented. If the state space in the above example was based on 10 timesteps, then the agent could take 11 steps, i.e. actions, during the episode, resulting in an episode length of $T = 11$. This is illustrated in figure 5.1.

5.1.2 State Space

Each state S_t is represented by a matrix of dimensions $F \times$ seq_len, where F is the number of features and seq_len is the number of timesteps used. By default, all $F = 21$ features from the overview in table 4.1 were used with seq_len $= 10$ timesteps corresponding to $t-9, t-8, ..., t-1, t$. This results in a total of $D = 21 \cdot 10 = 210$ input values in figures 2.3, 2.4 and 2.5, which illustrate the neural networks for the action-value function, policy function

and state-value function respectively. However, different values of seq_len and different sets of features were tested during hyperparameter tuning and feature selection for the second research question. All features were scaled: market and indicator features into the range $[-1, 1]$ and time features into the range $[0, 1]$ as described in section 4.2.1.

5.1.3 Action Space

At the each time t, the agent can trade the market price (in EUR) described in section 4.1.2. The agent returns a single, discrete action A_t with value a, where $a \in \{0, 1, 2\}$. This action specifies the amount of volume of electricity (in MWh) that the agent chooses to *own* at the end of the discrete timestep t. This action is then translated into the amount of volume bought V_t^b and the amount of volume sold V_t^s by the agent. At least one of these two terms V_t^b and V_t^s is 0 for each t since the agent either buys or sells (or does none of the two but never both). If A_{t-1} is the volume owned by the agent at the end of the previous timestep and A_t is the volume to be owned at the end of the current timestep, then the corresponding volumes to be bought and sold by the agent at the end of time t may be summarized as in equation 5.1.

$$f(A_t, A_{t-1}) = \begin{cases} \text{buy: } V_t^b = A_t - A_{t-1}, & \text{if } A_t > A_{t-1} \\ \text{sell: } V_t^s = A_{t-1} - A_t, & \text{if } A_t < A_{t-1} \\ \text{hold: } V_t^b = V_t^s = 0, & \text{if } A_t = A_{t-1} \end{cases} \tag{5.1}$$

In summary, the action A_t represents the amount of volume that the agent wants to *own* at the end of the timestep t. This action is translated into a buy, sell or hold position specifying the amount of volume that is to be bought or sold, V_t^b and V_t^s, according to equation 5.1. The specified volume is then bought or sold at the respective market price P_t at time t.

In addition, the agent was subject to further limitations. For instance, it was only allowed to buy if there was enough cash C_t available, where the initial cash C_0 was 1000 EUR at the beginning of each episode. Furthermore, in the last possible action during an episode, any volume owned by the agent was forcefully sold because the agent in this thesis was not meant to get any electricity delivered physically.

5.1.4 Reward Function

Four reward functions were applied during the thesis: Sharpe ratio, absolute return, vwap delta and log equity ratio. They are described in the equations from the theory section 2.44, 2.47, 2.48 and 2.49 respectively. For the Sharpe ratio, a running average of 5 time steps was used ($L = 4$ for 4 past timesteps and 1 current timestep).

5.2 Evaluation Metrics

For this thesis, five metrics were chosen for evaluating and interpreting the performance of the trained agents. The first metric, profit per volume traded, was used for answering the research questions and the other four metrics were included for further discussion. Metrics 1-3 were included based on discussions with the commissioner and metrics 4-5 were included based on previous studies [82] [8].

Essentially, metrics 1 and 2 (profit and profit per volume traded) measure the *financial result*, metric 3 (number of trades) measures the *activity* of the agent, metric 4 (percentage of positive returns) measures the *quality of trades* as investments and metric 5 (maximum draw down) measures *worst-case risk*. All five evaluation metrics are defined below. The same notation as in the theory section 2.6.3 about the rewards is used. Note that all metrics are computed per episode. For instance, if an agent is evaluated on 3481 test episodes, then for this agent, 3481 values are computed for each of the 5 metrics.

Importantly, the evaluation metrics are *not* fully aligned with the target metrics from section 2.6.2. For instance, the Sharpe ratio and the log equity ratio were not included as evaluation metrics. This can be explained as follows. First, the evaluation metrics were chosen for *all research questions* (and not just the first research question where different reward functions corresponding to the respective target metrics were tested). Second, the Sharpe ratio and the log equity ratio were *not the main objective* (which instead was profit per volume traded). Third, evaluation metrics beyond the target metrics may provide *valuable additional insights*, e.g. about risk and activity level. It is important to stress that there is always some *detachment* between target metric and what the agent optimizes for, namely the return G_t (as described in section 2.6.3). With that in mind, purely focussing on the target metric may give a limited understanding of the agents' performance. Hence, looking at other evaluation metrics can help to get a better understanding of the performance.

Metric 1: Profit per volume traded
The main metric used for answering all research questions is profit per volume traded, as defined in equation 5.2. This metric describes the *financial result* obtained on average *with each traded volume* during an episode. Large values are preferred. For instance, a large profit per volume traded for a certain episode may indicate that almost every trade resulted in some noticeable incremental profit during the episode.

$$\mathcal{M}_T^{(1)} = \frac{\sum_{t=0}^{T-1} \left[V_t^s \cdot P_t - V_t^b \cdot P_t \right]}{\frac{1}{2} \sum_{t=0}^{T-1} V_t^s + V_t^b} \tag{5.2}$$

Note that at the end of each episode, the total volume bought is equal to the total volume sold since the agent starts the episode with 0 MWh of electricity and needs too end the episode with 0 MWh of electricity. Therefore, the volume traded in the denominator can alternatively be expressed as the total amount of volume bought or as the total amount of volume sold: $\frac{1}{2} \sum_{t=0}^{T-1} V_t^s + V_t^b = \sum_{t=0}^{T-1} V_t^b = \sum_{t=0}^{T-1} V_t^s$

Metric 2: Profit
Profit is defined in equation 5.3 below. This metric describes the *financial result* obtained *with all trades* in a certain episode. Large values are preferred. For instance, a profit of 100 for a certain episode means that the agent owned 100 EUR more in cash at the end than at the beginning of the episode, thanks to all its actions taken during the episode.

$$\mathcal{M}_T^{(2)} = \sum_{t=0}^{T-1} \left[V_t^s \cdot P_t - V_t^b \cdot P_t \right] \tag{5.3}$$

Metric 3: Number of trades
The number of trades is defined in equation 5.4. This metric describes the level of *activity* of the agent. It represents the number of timesteps in an episode where the agent either buys or sells some volume of electricity, which is the case when $A_t > A_{t-1}$ or $A_t < A_{t-1}$ respectively. For instance, if the agent had *two* timesteps where it bought 2 and 1 volumes of electricity and *three* timesteps where it sold 1 volume of electricity each, the number of trades would be *five*. A small number of trades (in combination with large profit) is preferred.

$$\mathcal{M}_T^{(3)} = \sum_{t=0}^{T-1} \mathbb{1}_{\{A_t > A_{t-1} \text{ or } A_t < A_{t-1}\}} \tag{5.4}$$

Metric 4: Percentage of positive returns
The percentage of positive returns is defined in equation 5.5. The returns r_t were previously defined in equation 2.37. To avoid confusion, note that the monetary return r_t is meant here and not the RL return G_t. This metric describes the percentage of timesteps during an episode,

in which the agent observed an increase in its equity $\mathcal{E}_t$. Large values are preferred. For instance, a value of 0.6 for a certain episode means that during this episode, the equity $\mathcal{E}_t$ increased in 60% of the timesteps during the episode. The metric can be seen as a measure of the *quality of trades* as investments (with large quality corresponding to episodes where the agent invests, i.e. *owns* electricity, whenever the price *goes up*).

$$\mathcal{M}_T^{(4)} = \frac{1}{T} \sum_{t=1}^{T} \mathbb{1}_{\{r_t > 0\}} \tag{5.5}$$

Metric 5: Maximum draw down
The maximum draw down (MDD) is defined in equation 5.6. MDD describes the largest percentage decrease in equity $\mathcal{E}_t$ from one timestep to the subsequent timestep during the episode. All MDD values are negative, except for episodes in which equity increases across all timesteps. Large values are preferred, e.g. -0.1 is preferred over -0.5. For instance, a value of -0.5 for a certain episode means that during the worst timestep of this episode, the equity of the agent decreased in value by 50%. This metric can be seen as a measure of *worst-case risk* since it describes how much equity was lost in the worst timesteps of an episode.

$$\mathcal{M}_T^{(5)} = \min_{\forall t \in \{1,\dots,T\}} \frac{\mathcal{E}_t - \mathcal{E}_{t-1}}{\mathcal{E}_{t-1}} \tag{5.6}$$

5.3 Bayesian Optimization

5.3.1 Motivation

Hyperparameters and features have in common that they are generally chosen by the researcher before training a machine learning model. Choosing the optimal hyperparameters and features can noticeably improve model performance and possibly speed up training by reducing model complexity [83], [84]. Common approaches to feature selection, originated in supervised learning, are algorithms such as forward and backward stepwise selection as well as best subset selection [85]. Common approaches to hyperparameter optimization are for instance grid search, random search and bayesian optimization [86], [87].

With regards to hyperparameter optimization, Bayesian optimization has become popular due to its *efficiency* and its ability to find *global optima* [84]. It has also been applied in reinforcement learning. For instance, Bayesian optimization noticeably contributed to the success of AlphaGo [88]. With regards to feature selection, certain approaches have been proposed for the context of reinforcement learning [89]–[91] but these methods are generally designed for specific use cases, e.g. for certain RL models. Even in academia, successful approaches for feature selection in reinforcement learning have generally been rather straight-forward:

First, it seems common to use features *as they are*, e.g. video game images for *Atari* games [3] or positions of objects on board games for *AlphaGo Zero* [88]. Second, it seems common to use manually handcrafted features and test if they improve performance. For instance, this was done when applying reinforcement learning for *Dota 2* [6] and also when training *AlphaGo* [88]. While these approaches obviously worked, they may not be the best methodological approaches in scenarios where feature engineering makes sense: in such scenarios it seems appropriate to optimize features just like hyperparameters are optimized. This motivates the approach taken in this thesis: for the second research question, hyperparameter optimization *and* feature selection *and* in addition reward function selection, were approached with Bayesian optimization *together*.

Detailed introductions to Bayesian optimization can be found in [92] and [93], the main references for the following short introduction. In Bayesian optimization, we essentially model an objective function $f(x)$ that is to be optimized. In this thesis for instance, the objective function takes as *input* the vector of optimization parameter values x and returns as *output* the median of the main evaluation metric, profit per volume traded, based on the

validation data on which the RL agent is evaluated. In Bayesian optimization, the objective function is modelled sequentially: in every trial, a vector x is generated, the function $f(x)$ is evaluated at this x and then the model of the objective function $f(x)$ is updated with the Bayes theorem based on the observed values of x and $f(x)$. Importantly, the vector of x is generated in each trial in such a way that the model of $f(x)$ can be improved in regions of high *uncertainty* about the values of $f(x)$ and in regions where large values of $f(x)$ are expected.

5.3.2 Implementation

For this thesis, the Optuna framework, introduced in [94], was used for Bayesian optimization. Using this framework, the objective function $f(x)$ was approximated with a graphical model, the Tree Parzen Estimator, explained in [84] and [95]. Essentially, Optuna is a Python package that allows to apply Bayesian optimization in a very flexible way. With Optuna, it is not only possible to specify custom *optimization parameters* whose optimal values are to be found but also to define a custom *objective function* that is to be optimized.

Two functions are to be defined by the researcher: a sampling function, which generates a new set of optimization parameters to be tested in the next trial, and the objective function, which takes this set of parameters, conducts a trial by evaluating the set of parameters and returns the metric to be optimized. The complete objective function is implemented by the researcher. For this thesis, it was defined to do the following: a) prepare the training and validation data sets, b) prepare the RL environment, c) train the RL agent on training data, d) evaluate the RL agent on validation data, e) return the median of the main evaluation metric, profit per volume traded, based on the validation episodes.

Three different categories of optimization parameters were defined: *features* for the state space, *hyperparameters* for the training phase and the *reward function* for the RL environment. For the features, 21 binary optimization parameters were defined (one for each feature in table 4.1): each feature was either used for training and evaluating or *not* used for training and evaluating. For the hyperparameters, an overview for A2C is given in tables A.14 and A.15, and an overview for DQN is given in tables A.12 and A.13 in the appendix. For the *reward function*, one categorical variable was defined, which could take four values (one for each of the four reward functions from section 5.1.4). Note that inside the objective function, the *feature* parameters were used in step a) to adjust the training and validation data sets (subsetting them to the specified features to be used in the respective trial), the *reward function* parameter was used in step b) to adjust the RL environment (selecting the specified reward function for giving feedback to the agent) and the *hyperparameters* were used in step c) to train the RL agent (with the specified hyperparameters for A2C or DQN).

The main advantage of the above approach is that it theoretically allows to *efficiently* find the *globally optimal combination* of features, hyperparameters and reward function (for the validation data). This would not be possible if reward function selection, feature selection and hyperparameter tuning were conducted in separate stages, consisting of first finding the best reward function, then the best features and then the best hyperparameters while always keeping the parameters of the remaining two categories fixed. Lastly, note that applying Bayesian optimization for feature selection as described above is better than e.g. applying general approaches such as forward or backward stepwise selection or best subset selection, which are described in [85]. Forward and backward stepwise selection do not allow for *all* feature combinations to be tested but the approach taken here does since *every* feature can either be used or not in *any* trial. Best subset selection is very inefficient because all feature combinations *are* tested but with the approach taken here, only *promising* combinations as suggested by the Bayesian optimization algorithm are tested.

5.4 Baseline Strategies

Baseline strategies were implemented in order to compare the performance of the RL agent with simple methods that do not make use of RL. Both, the RL agent and the baseline strategies, represent agents that trade electricity. The action space, described in section 5.1.3, remains the same for the RL agent and the baseline strategies. At every timestep t, the action $A_t \in \{0, 1, 2\}$ determines the volume of electricity that the agent will *own* at the end of this timestep. The action A_t is then converted into buy or sell orders depending on A_{t-1}, the amount of electricity owned by the agent at the previous time step.

The four implemented baseline strategies (random trader, forecast trader, moving average trader and spike trader) are described in the sections below. In short, the *random trader* randomly samples an action at every timestep, the *forecast trader* takes an action based on its price forecast for the next timestep, the *moving average trader* owns some electricity when the price is above the moving average price and owns no electricity otherwise and the *spike trader* always owns some electricity but sells when there is a sudden price increase. Importantly, the spike trader was implemented retrospectively in order to mirror the strategy of the RL agent.

The only baseline strategy that requires a training phase is the *forecast trader* since its forecast model needs to be trained. The other three baseline strategies are simple algorithms that can be directly applied to the test data without any prior training phase. Furthermore, note that the random trader does not use any information from the state space that is available to the RL agent, the *forecast trader* uses the exact same state space as the RL agent and both, *moving average trader* and *spike trader*, only use the price from the state space. For simplicity, a constant volume of 2 MWh (and never 1 MWh) was traded by the forecast, moving average, and spike trader. However, a volume of 1 MWh would have been possible alternatively.

5.4.1 Random Trader

At every timestep t, the random trader samples an action $A_t \in \{0, 1, 2\}$. Each possible action is sampled with equal probability. Sampling random actions is a widely used baseline strategy in reinforcement learning, but it generally performs particularly well in trading applications (from the financial domain). For instance, a comparison of a random trader with four other trading strategies based on technical indicators has shown that "standard trading strategies and their algorithms, [...] perform on average not better than the purely random strategy, which, on the other hand, is also much less volatile." [96, p.11]

5.4.2 Forecast Trader

A forecast trader is a more advanced benchmark than a random trader. Forecast traders have also been used as baselines for trading RL agents in the financial domain [2], [16]. For instance, Gao and Chan implemented a forecast trader that would buy if the prediction for the next timestep corresponded to a price percentage increase $> 3\%$, sell if the prediction for the next timestep corresponded to a price percentage decrease $< -3\%$ and keep the current investment otherwise [16].

A very similar approach is taken for this thesis. A prediction model is trained to perform a prediction at every timestep t for the next timestep $t + 1$. In particular, it performs a classification of three classes, corresponding to a price percentage change of $> 3\%$, $< -3\%$ or inbetween. If the model classifies a price percentage increase $> 3\%$, the action is $A_t = 2$. In other words, the forecast trader buys 2 MWh of electricity if it does not own any electricity and does nothing if it already owns 2 MWh of electricity. If the model classifies a price percentage decrease $< -3\%$, the action is $A_t = 0$. In other words, the forecast trader sells all

electricity that it owns. If the model classifies a price percentage change between $< -3\%$ and $> 3\%$, then the trader keeps the current investment.

$$A_t = \begin{cases} 0, & \text{if forecast is < -3\%} \\ 2, & \text{if forecast is > 3\%} \\ A_{t-1}, & \text{otherwise} \end{cases} \tag{5.7}$$

The question is which model and which features should be used for the forecast model. To allow for a fair comparison, forecasts are conducted based on the state space that was used for the RL agent as described in table 4.1. In other words, each action A_t is determined with a forecast that is based on state S_t. Note that all 21 features were used for the forecast model, whereas fewer features were used for the RL agent due to Bayesian optimization. Furthermore, the network architecture used for the forecast model resembled the network architecture of the RL agent: a neural network with two hidden layers of 64 nodes each, as shown in figure A.21 in the appendix.

5.4.3 Moving Average Trader

Trading strategies based on moving averages are common in the financial domain [97]. These trading strategies are based on the "widespread belief that the most recent stock prices contain more relevant information on the future direction of the stock price than earlier stock prices" [97, p. 5]. The main concept is to compare the price P_t with a moving average of this price, e.g. the simple moving average of 10 timesteps $SMA_t = \frac{1}{10}\sum_{j=0}^{9} P_{t-j}$. If the price P_t is *larger* than the corresponding simple moving average SMA_t, a *buy* signal is given (assuming that the price will further increase because the most recent price is above the moving average computed with older prices). Reversely, if the price P_t is *smaller* than the corresponding simple moving average SMA_t, a *sell* signal is given (assuming that the price will further decrease because the most recent price is below the moving average computed with older prices). For the context of this thesis, the rule can be summarized as in equation 5.8, where $SMA_t = \frac{1}{10}\sum_{j=0}^{9} P_{t-j}$.

$$A_t = \begin{cases} 0, & \text{if } P_t \leqslant SMA_t \\ 2, & \text{if } P_t > SMA_t \end{cases} \tag{5.8}$$

5.4.4 Spike Trader

This baseline strategy was included *retrospectively*, i.e. *after* understanding the strategy of the final RL agent. When analyzing the actions of the RL agent, the author found that it performed so well because it had learned to sell some volume of electricity at sudden price increases. This motivates the *spike trader* baseline strategy in equation 5.9, a strategy that was *reverse* engineered to mirror the main strategy of the RL agent. Essentially, the action of the spike trader is always 2, i.e. it always owns 2 MWh of electricity. However, when the price increases by $> 25\%$ from one timestep to another (starting to form a "spike") the action is 0, i.e. the spike trader sells 2 MWh of electricity.

$$A_t = \begin{cases} 0, & \text{if } P_t / P_{t-1} > 1.25 \text{ and } \text{abs}(P_{t-1}) > 1 \\ 2, & \text{otherwise} \end{cases} \tag{5.9}$$

5.5 Statistical Evaluation

The purpose of the statistical evaluation was to test the hypothesis that the agent with the best performance with respect to the main evaluation metric $\mathcal{M}_T^{(1)}$, profit per volume traded

as defined in equation 5.2, achieves significantly better results than the other agents. From here on, profit per volume traded is referenced as $\mathcal{M}$ to simplify the notation.

For each of the three research questions, the best agent was compared against the remaining agents: in the first research question, the RL agent trained with the best reward function was tested against the three remaining agents trained with different reward functions, resulting in three hypothesis tests. In the second research question, the RL agent trained with the best RL method was tested against another RL agent trained with a different RL method, resulting in one hypothesis test. In the third research question, the RL agent was tested against four baseline strategies, resulting in four hypothesis tests.

5.5.1 Statistical Method

Various statistical tests were considered before finally choosing bootstrap confidence intervals and bootstrap hypothesis tests because of their interpretability, customizable test statistic and flexibility with respect to the data distribution [98], [99].

First, it was noted that the samples were not independent since all agents were evaluated on the exact same 3481 test episodes. Therefore, tests that assume independent samples, such as the independent two-samples t-test, were excluded [100]. Instead, the paired two-samples t-test and the paired samples Wilcoxon test were considered [101], [102]. Among others, the former assumes that the differences of the paired values follow a normal distribution and the latter assumes that they are symmetrically distributed. (With 3481 test episodes, there were 3481 pairs of values $\mathcal{M}$ for the pair of agents to be compared in a particular test.)

Neither of these assumptions seemed to be fulfilled based on Q-Q plots and boxplots of the differences in paired values (see example plots for the first research question in figure A.6). Further, the Shapiro-Wilk normality test confirmed that the assumption of normality was not fulfilled, rejecting the null hypothesis of normal distribution with $p<0.0001$. Essentially, the distributions are strongly skewed to the right. Transformations for reducing skewness, such as log transformations, were not considered in favor of interpretability. Lastly, methods based on resampling were considered and the final choice fell on bootstrap hypothesis tests and confidence intervals.

5.5.2 Test Statistic

The test statistic of interest, defined in equation 5.10, is the median of the paired differences in profit per volume traded $\mathcal{M}$. Each pair corresponds to a specific test episode i on which both agents, indexed by 1 and 2 below, were evaluated.

$$\hat{\theta} = \text{median}\left(\mathcal{M}_{i,1} - \mathcal{M}_{i,2}\right), \tag{5.10}$$

where $i \in \{1, 2, ..., 3481\}$ is the index representing the test episode. The median was chosen due to the large skewness of the paired differences. For the context of trading, it is preferred that extreme values (of paired differences in profit per volume traded) do not affect the judgement. Note that the median of the paired difference, i.e. median $(\mathcal{M}_{i,1} - \mathcal{M}_{i,2})$, was chosen instead of the difference of medians, i.e. median $(\mathcal{M}_{i,1})$ − median $(\mathcal{M}_{i,2})$, in order to undoubtedly preserve the paired structure of the data.

5.5.3 Bootstrap Hypothesis Test

In each hypothesis test, two agents were compared: agent 1, the supposedly better performing agent, and agent 2, the supposedly worse performing agent. The null hypothesis that

both agents perform equally well was tested against the alternative hypothesis that agent 1 performs better than agent 2:

$$\begin{aligned} H0 &: \theta = 0 \\ H1 &: \theta > 0 \end{aligned} \tag{5.11}$$

The main idea of *bootstrap* is to resample with replacement from the *original* data sample. Here, the original data sample consists of 3481 paired differences $d_i = (\mathcal{M}_{i,1} - \mathcal{M}_{i,2})$, where $i \in \{1, 2, ..., 3481\}$ is again the index representing the test episode. If we resample from this data with replacement once, we obtain another 3481 values. However, some d_i will be duplicated and some will be missing due to sampling with *replacement*. Now, multiple such bootstrap samples, here $B = 4999$ bootstrap samples, are drawn and for each of them, the test statistic from equation 5.10 is computed. This gives the bootstrap test statistics $\hat{\theta}^*_j$, where $j = 1, 2, ..., 4999$ is the index for the bootstrap sample.

The main idea of *bootstrap hypothesis testing* is to compare the test statistic $\hat{\theta}$, computed based on the *original* data, with the bootstrap sampling distribution of $\hat{\theta}^*$ under the H0, i.e. the sampling distribution of $\hat{\theta}^*$ if the null hypothesis was true [103]. If $\hat{\theta}$ is large (or small) enough in comparison with the sampling distribution of $\hat{\theta}^*$ under the H0, then the H0 is rejected. For this purpose, either a p-value can be determined and compared to a significance level α (rejecting the H0 if $p < \alpha$) or critical values $\hat{\theta}_{\text{crit}}$, corresponding to the significance level α, can be compared with the observed value of $\hat{\theta}$ (rejecting the H0 if $\hat{\theta} > \hat{\theta}_{\text{crit}}$).

Two guidelines have been suggested for bootstrap hypothesis testing in [104]. The first guideline is to compare $\hat{\theta}$ with $\hat{\theta}^* - \hat{\theta}$ and the second guideline is to compare $\hat{\theta}/\hat{\sigma}$ with $(\hat{\theta}^* - \hat{\theta})/\hat{\sigma}^*$. According to the authors, the first guideline ensures sufficient *power* and the second guideline reduces *error in the level of significance*. Recall that power is the probability of rejecting the H0 if the H1 is true and error in the level of significance is the difference in the significance level that the test actually has and the level that it is supposed to have, namely the level α set by the researcher.

The first guideline essentially ensures that $\hat{\theta}$ is compared with the sampling distribution of $\hat{\theta}$ *under the H0*, represented by $\hat{\theta}^* - \hat{\theta}$ as mentioned before. Note that this is equivalent to transforming the paired differences *before* resampling as follows: $d'_i = d_i - \hat{\theta} = (\mathcal{M}_{i,1} - \mathcal{M}_{i,2}) - \text{median}\,(\mathcal{M}_{i,1} - \mathcal{M}_{i,2})$. By subtracting the median of the paired differences from the paired differences, we ensure that the median of the transformed paired differences (based on which bootstrap sampling is conducted) takes the value 0. Hence, $\hat{\theta}^*_j - \hat{\theta}$ represents the sampling distribution of $\hat{\theta}$ if the H0 from equation 5.11 is true.

The second guideline accounts for the unknown scale of $\hat{\theta}$, denoted as $\hat{\sigma}$, and for the unknown scale of $\hat{\theta}^*$, denoted as $\hat{\sigma}^*$. In particular if the two scales differ noticeably, the second guideline should be adhered to. However, this guideline requires estimates of $\hat{\sigma}$ and $\hat{\sigma}^*$. For clarity, $\hat{\sigma}$ and $\hat{\sigma}^*$ denote the standard deviation of $\hat{\theta}$ and $\hat{\theta}^*$ respectively. They can also be referred to as standard errors. Unfortunately, there is no appropriate closed-form estimator of the standard error of *medians* (if the sampled values, here the differences d_i, are not normally distributed). Instead, the standard errors can be estimated with bootstrap, resulting in the dual bootstrap hypothesis test described in [103]. The dual bootstrap hypothesis test was used for this thesis in order to adhere to not only the first but also the second guideline from [104]. The dual bootstrap hypothesis test is summarized in algorithm 1 below.

Finally, Bonferroni adjustment was used to account for multiple hypothesis tests comparing the same agent with several other agents. The significance level was set to $\alpha = 0.05$. For the first research question (with three hypothesis tests), α was adjusted to $\alpha/3 \approx 0.017$. For the second research question, there was only one hypothesis and hence no adjustment was needed. For the third research question (with four hypothesis tests), the α was adjusted to $\alpha/4 \approx 0.013$. Note that the significance levels were rounded to the third digit in order to fulfill the requirement that $\alpha(B + 1)$ is an integer, where $B = 4999$ [103].

Algorithm 1: Dual bootstrap hypothesis test [103]

input : Parameters B_1, B_2 for the number of bootstraps to be conducted, original data sample of paired differences $d_i = \mathcal{M}_{i,1} - \mathcal{M}_{i,2}$ for $i \in \{1, ..., N\}$

output: Bootstrap p-value

1 Obtain estimate $\hat{\theta} = \text{median}\,(d_i)$ for $i \in \{1, ..., N\}$ based on the original data sample

2 Generate B_2 bootstrap samples. Compute $\hat{\theta}_j^*$ for each bootstrap sample where $j \in 1, ..., B_2$. Estimate $\hat{\sigma}$, the estimate for the standard deviation of $\hat{\theta}$, as the sample standard deviation of all bootstrap estimates $\hat{\theta}_j^*$.

3 Calculate the test statistic $\hat{\tau} = \hat{\theta}/\hat{\sigma}$.

4 Generate B_1 bootstrap samples. Compute $\hat{\theta}_k^*$ for each bootstrap sample where $k \in 1, ..., B_1$.

5 For each of the B_1 bootstrap samples, perform step 2 exactly as before, obtaining $\hat{\sigma}_k^*$ for $k \in 1, ..., B_1$. Compute the bootstrap test statistics as $\tau_k^* = (\hat{\theta}_k^* - \hat{\theta})/\hat{\sigma}_k^*$.

6 Calculate the bootstrap p-value as $p = \frac{1}{B_1} \sum_{k=1}^{B_1} \mathbb{1}_{\{\tau_k^* > \hat{\tau}\}}$

5.5.4 Bootstrap Confidence Intervals

In addition to the p-values for the bootstrap hypothesis tests, 95% bootstrap confidence intervals were reported for reference. In particular, bias-corrected and accelerated bootstrap confidence (BCa) intervals were computed based on $B = 4999$ bootstrap samples. BCa intervals were introduced under the title "Better bootstrap confidence intervals" in [105] and are widely used, among others because they also work well when the sampling distribution is *skewed* [99]. Since the paired differences $d_i = (\mathcal{M}_{i,1} - \mathcal{M}_{i,2})$ are skewed, as illustrated in the Q-Q plots in figure A.6, BCa intervals are a suitable choice for this thesis. Importantly, these intervals were not used for hypothesis testing. Instead, they were only provided as further reference. Therefore, no Bonferroni adjustment was conducted.

5.6 Reward (Question 1)

For the first research question, the four reward functions described in 5.1.4 were compared. The underlying purpose was to understand which of the four reward functions results in the best performance with respect to the main evaluation metric, profit per volume traded.

5.6.1 Training

Four different agents were trained: one agent for each reward function. All agents were trained using A2C with the same default hyperparameters (as described in table A.15) and with all 21 features (as described in table 4.1). Both, training and validation data sets, were used for training and the test data set was used for evaluation (as described in section 4.3).

The policy $\hat{\pi}(a|s, \boldsymbol{\theta})$ was represented by a neural network as displayed in figure 2.4. The state-value function $\hat{v}(s, \mathbf{w}_v)$ was represented by a neural network as displayed in figure 2.5. Both neural networks consisted of an input layer with 210 nodes (based on 21 features and 10 timesteps for each feature) and two hidden layers with 64 nodes each. For the policy network, there were 3 output nodes due to 3 possible discrete actions $a \in \{0, 1, 2\}$. For the state-value network, there was 1 output node by definition. The tanh activation function was used within both networks. In the output layer, the softmax activation function was used for the policy network, and the linear activation function was used for the state-value network.

5.6.2 Evaluation

The evaluation, conducted on the test data, can be structured into three parts: a statistical evaluation, a descriptive evaluation and a training evaluation.

In the *statistical* evaluation, three hypothesis tests were conducted as described before in section 5.5 to test whether the supposedly best agent performs significantly better than the remaining three agents. The corresponding hypotheses are listed below in equation 5.12. Index 1 corresponds to the best performing agent (with the supposedly best reward function), and indices 2-4 correspond to the remaining agents (with the remaining reward functions). For instance, test 1 below compares the agent with index 1 against the agent with index 2.

$$\begin{aligned} &\text{Test 1}: \quad H0 : \theta_{1,2} = 0, \quad H1 : \theta_{1,2} > 0 \\ &\text{Test 2}: \quad H0 : \theta_{1,3} = 0, \quad H1 : \theta_{1,3} > 0 \\ &\text{Test 3}: \quad H0 : \theta_{1,4} = 0, \quad H1 : \theta_{1,4} > 0 \end{aligned} \tag{5.12}$$

The purpose of the *descriptive* evaluation was to further compare the performance and the actions of the different agents on the test episodes. First, the distributions of the evaluation metrics described in section 5.2 were visualized with boxplots for each reward function. Second, three episodes with different price patterns (uptrend, downtrend, spikes) were visualized for each of the four agents to illustrate the different decision policies of the agents. Episodes were categorized according to these price patterns by using the rules from section A.6 in the appendix.

The purpose of the *training* evaluation was to check whether the agents were able to improve their performance across the training period. Hence, the training metrics from section A.5.1 were plotted across the steps of the training period for each of the four agents. The metrics were z-scaled for each agent to allow for comparisons across agents. Because of strong variation in the z-scaled metric values across timesteps, the moving averages of 1000 steps were plotted (instead of the z-scaled metrics for each individual timestep).

5.7 Methods (Question 2)

Two methods were compared: A2C and DQN. The purpose of this was to understand potential differences in performance between RL methods. Hereby, A2C was chosen as representative of actor-critic methods and DQN was chosen as representative of critic-only methods. For the method comparison, Bayesian optimization was applied to find an optimal set of features, hyperparameters and reward function for A2C and DQN. Note that A2C was also used in the first research question, however without Bayesian optimization.

5.7.1 Training

Two different agents were trained: one agent for each method. For A2C, the same network architecture as for the first research question was used (see section 5.6.1). For DQN, the action-value function $\hat{q}(s, a, \mathbf{w}_q)$ was represented by a dueling neural network (see section 2.5.3). Each of the two streams of this network, the state-value stream and the advantage stream, consisted of two hidden layers with 64 neurons each. The ReLU activation function was used inside the streams and the linear activation function was used for their output layers. The output layer of the dueling network, computed based on the outputs of the two streams, consisted of three output nodes representing $\hat{q}(s, a_1), \hat{q}(s, a_2), \hat{q}(s, a_3)$, just like the in the Q-network displayed in figure 2.3. Note that the number of input nodes of the networks was smaller than 210 for both final models, DQN and A2C, since fewer than 21 features were selected during feature selection with Bayesian optimization.

For both models, the features, hyperparameters and reward function were selected using Bayesian optimization as described in section 5.3. In particular, 600 trials were conducted for each model, A2C and DQN. In each trial, an RL agent was trained with a set of optimization

parameters (features, hyperparameters and a reward function) as suggested by the Bayesian optimization algorithm. The training data was used for training and the validation data set was used for evaluating each set of optimization parameters. The best set of parameters was selected as the set corresponding to the trial where the RL agent performed best on the validation data, as measured by median profit per volume traded computed on the validation episodes. For clarity, the best set was determined for DQN and A2C separately (from their respective 600 trials). The *final* models for DQN and A2C were each trained on training and validation data *together* using their respective best set of optimization parameters. Subsequently, these final models were evaluated on the test data to answer the research question.

5.7.2 Evaluation

The evaluation, conducted on the test data, can be structured into three parts: a statistical evaluation, a descriptive evaluation and a training evaluation.

In the *statistical* evaluation, one hypothesis test was conducted as described before in section 5.5. The purpose was to test whether the supposedly better agent, indexed by 1, performs significantly better than the remaining agent, indexed by 2:

$$\text{Test 1}: \quad H0 : \theta_{1,2} = 0, \quad H1 : \theta_{1,2} > 0 \tag{5.13}$$

The purpose of the *descriptive* evaluation was to further compare the performance and the actions of the A2C and DQN agents on the test episodes. Just like for the first research question, the distributions of the evaluation metrics described in section 5.2 were visualized with boxplots for DQN and A2C. Furthermore, the exact same three episodes from the first and second research question were visualized for the RL agent and for the baseline strategies.

The purpose of the *training* evaluation was, just like in research question 1, to check whether the agents were able to improve their performance across the training period. Hence, the training metrics described in section A.5 were plotted across the steps of the training period for both, A2C and DQN. In particular, the moving averages (of 1000 timesteps) based on the *un*scaled training metric values were visualized.

5.8 Baseline (Question 3)

The optimized A2C agent from the second research question was compared with four baseline strategies: the random, forecast, moving average and spike trader (as described in section 5.4). The purpose of this was to investigate whether the optimized RL agent can perform better than simple alternative solutions, which do not make use of reinforcement learning.

5.8.1 Training

No new RL agent was trained. Instead, the A2C agent from the second research question was re-used. Only one of the four baseline strategies required training, namely the *forecast* trader. The other three trading stratgies, i.e. the *random*, *moving average* and *spike* traders as well as the already trained RL agent could directly be evaluated on the test data (without training).

The forecast trader was already introduced in section 5.4.2. It is based on a neural network as shown in figure A.21, which predicts the percentage change of the market price (from t to $t+1$) based on the state S_t, consisting of all 21 features given for 10 timesteps each. In particular, this was framed as a classification problem with three classes represented by percentage changes: $> 3\%$, $< -3\%$ and inbetween. The network consisted of 210 input nodes, two hidden layers with 64 nodes each and 3 output nodes. The ReLU activation function was used within the network and the softmax activation function was used for the output layer.

The forecast model was trained with the Python package Keras, using a batch size of 128 for the weight updates and using cross-entropy as loss function (called

categorical_crossentropy in Keras; see [106] for reference). Dropout was used with $p = 0.5$ for both hidden layers to prevent overfitting as recommended in [107].

First, the appropriate number of epochs was determined. For this purpose, a model was trained on the training data and evaluated on the validation (and training) data after every epoch. In particular, this model was trained until the loss on the validation data did not further improve by at least 0.0001 during a window of 25 epochs. This happened after 162 epochs. Plots of accuracy and loss by training epoch can be found in figure A.22.

Subsequently, the final model was trained on training and validation data *together* for the previously determined number of 162 epochs. The model's *classification* performance on the test data was analyzed before subsequently using the model for trading by means of the forecast baseline (as described in section 5.4.2) along with the other baseline strategies. The classification performance on the test data is described in table A.7 and the subsequent tables on the same page in the appendix.

5.8.2 Evaluation

The evaluation, conducted on the test data, can be structured into two parts: a statistical evaluation and a descriptive evaluation. The purpose of the evaluation was to compare the RL agent with the four baseline strategies.

In the *statistical* evaluation, four hypothesis tests were conducted as described before in section 5.5 to test whether the RL agent performs significantly better (or worse) than the four baseline strategies. Since the agent did perform worse than one baseline strategy, the following hypothesis tests were conducted, where index 1 corresponds to the RL agent and indices 2-5 correspond to the four baseline strategies. In test 1, the agent is compared with the baseline strategy that seemed to perform better than the agent based on descriptive statistics, in particular median profit per volume traded.

$$\begin{aligned} \text{Test 1}: &\quad H0 : \theta_{1,2} = 0, \quad H1 : \theta_{1,2} < 0 \\ \text{Test 2}: &\quad H0 : \theta_{1,3} = 0, \quad H1 : \theta_{1,3} > 0 \\ \text{Test 3}: &\quad H0 : \theta_{1,4} = 0, \quad H1 : \theta_{1,4} > 0 \\ \text{Test 4}: &\quad H0 : \theta_{1,5} = 0, \quad H1 : \theta_{1,5} > 0 \end{aligned} \tag{5.14}$$

The purpose of the *descriptive* evaluation was to further compare the performance and the actions of the agent and the baseline strategies on the test episodes. Just like for the previous research questions, the distributions of the evaluation metrics described in section 5.2 were visualized with boxplots for the RL agent and the four baseline strategies. Furthermore, the exact same three example episodes from the previous research questions were visualized for the RL agent and the baseline strategies. For reference, the rewards received by the RL agent for its actions and the rewards that the baseline strategies *would* have received (if they were RL agents) were plotted as well.

5.9 Software

For this thesis, a custom environment was written in Python, following the environment requirements the *gym* library by OpenAI [23]. Subsequently, this environment was used for training with implementations of the algorithms A2C and DQN by *stable-baselines* [108]. For hyperparameter tuning, the *Optuna* framework was used, which approaches hyperparameter tuning as an Bayesian optimization problem [94].

6 Results

Overall, the results are promising in regards to the underlying purpose of this thesis to create a RL agent that can profitably trade electricity. The final RL agent learned to buy and sell electricity in a way that resulted in profits. In particular, the agent learned to sell at sudden price increases and to buy again after subsequent price decreases, making use of the volatility in electricity prices. The agent achieved a median profit per volume traded of 4.02 EUR across all test episodes. Importantly, these results are optimistic because of the simplified environment. In this thesis, all orders of the RL agent are executed at market price and the market price is known to the agent. In a real-world application, these conditions are not given. The final RL agent, compared with baseline strategies in section 6.3, was trained with A2C and the vwap delta reward function. Features, hyperparameters and the reward function of this agent were optimized with Bayesian optimization on the validation episodes.

Detailed descriptions of the results for each of the three research questions follow below. Section 6.1 addresses the first research question about the reward function comparison, section 6.2 is dedicated to the second research question about the method comparison and section 6.3 focuses on the third research question about the baseline strategy comparison.

6.1 Reward (Question 1)

6.1.1 Statistical Evaluation

The distributions of profit per volume traded in EUR, based on 3841 *test* episodes, are described for each agent and its respective reward function in table 6.1. The agents trained with profit based reward functions, vwap delta and absolute return, achieved larger mean and median profits per volume traded than the agents trained with reward functions that adjust for risk, Sharpe ratio and log equity. However, the results achieved by the profit based agents also have a larger standard deviation. The agent trained with the vwap delta reward function achieves the best results in terms of median and mean profit per volume traded.

The question is whether the median differences of profit per volume traded θ between the best reward function, vwap delta, and the remaining three reward functions are significantly larger than 0. The bootstrap p-values for the corresponding three hypothesis tests are shown in table 6.2. The $H0 : \theta = 0$ was rejected in favor of the $H1 : \theta > 0$ when comparing the vwap delta with the Sharpe ratio and the log equity ratio reward functions since $p < 0.017$.

Table 6.1: Distribution of profit per volume traded by reward function (test data, n = 3481)

Reward Type	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
vwap_delta	3481	10.09	52.82	-57.66	-0.40	2.83	9.77	1957.21
abs_return	3481	6.94	47.20	-50.54	-0.16	1.91	6.48	1957.21
sharpe_ratio	3481	1.77	15.13	-48.29	-0.41	0.34	1.77	712.21
log_equity	3481	0.51	8.22	-88.71	-0.57	0.05	0.91	325.17

However, the test failed to reject the $H0$ when comparing the vwap delta with the absolute return reward function since $p > 0.017$. Hence, it can be concluded that the profit based reward function *vwap delta* significantly outperforms both risk-adjusted reward functions, *Sharpe ratio* and *log equity*. However, it cannot be concluded that it significantly outperforms the other profit based reward function *absolute return*.

Table 6.2: Bootstrap hypothesis tests and confidence intervals for the median differences of profit per volume traded (test data, n = 3481, R = 4999)

Agent_1	Agent_2	median($d_{1,2}$)	Boot. 95% CI	Boot. SE	p-value
vwap_delta	abs_return	0.10	(0.00, 0.20)	0.05	0.0314
vwap_delta	sharpe_ratio	2.00	(1.80, 2.30)	0.13	0.0000
vwap_delta	log_equity	2.48	(2.22, 2.78)	0.14	0.0000

6.1.2 Descriptive Evaluation

6.1.2.1 Example Episodes

Among all test episodes, there were 40% spike episodes, 11% uptrend episodes, 14% downtrend episodes and 35% no trend episodes, where the test episodes had been categorized as described in section A.6. To provide an insight into the behavior of the agents, the actions of all four agents are visualized for exemplary episodes, namely for an episode with *spikes*, an episode with *uptrend* and an episode with *downtrend*. The visualization of the episode with spikes can be found below in figure 6.1 and the visualizations of episodes with uptrend and downtrend can be found in figures A.12 and A.13 in the appendix.

Figure 6.1 shows an episode with *spikes*. The figure illustrates that the agents trained with reward functions based on profit learn to sell when the electricity price spikes up and to buy right after spike is over, resulting in large profits. In contrast, the agents trained with reward functions that account for risk do not primarily tend to sell at the spikes.

Figure A.12 shows an episode with *uptrend*. The figure illustrates that all agents tend to generate profits in case of an uptrend. However, it seems as if the agents trained with profit based reward functions do particularly well because they buy a large volume early. For instance, they buy 2 MWh of electricity at the beginning of the episode instead of 1 MWh.

Figure A.13 shows an episode with *downtrend*. The figure illustrates that, in case of a downtrend, the agents trained with profit based reward functions do *not* perform as well as the agents trained with reward functions that account for risk. This is because they tend to buy a large amount of volume at the beginning of the episode when the price is large and to sell it at the end of the episode when the price is small, resulting in a large loss.

In addition, tables A.1 and A.2 in the appendix show the distributions of profit and of per volume traded by *reward function* and *episode type*, where the test episodes were categorized as described in section A.6. These tables back up the main conclusions drawn from the visualizations of example episodes above: the profit based reward functions (vwap delta and absolute return) *outperform* the risk-adjusted reward functions (Sharpe ratio and log equity) in episodes with spikes and uptrend but they *are outperformed* in episodes with downtrend.

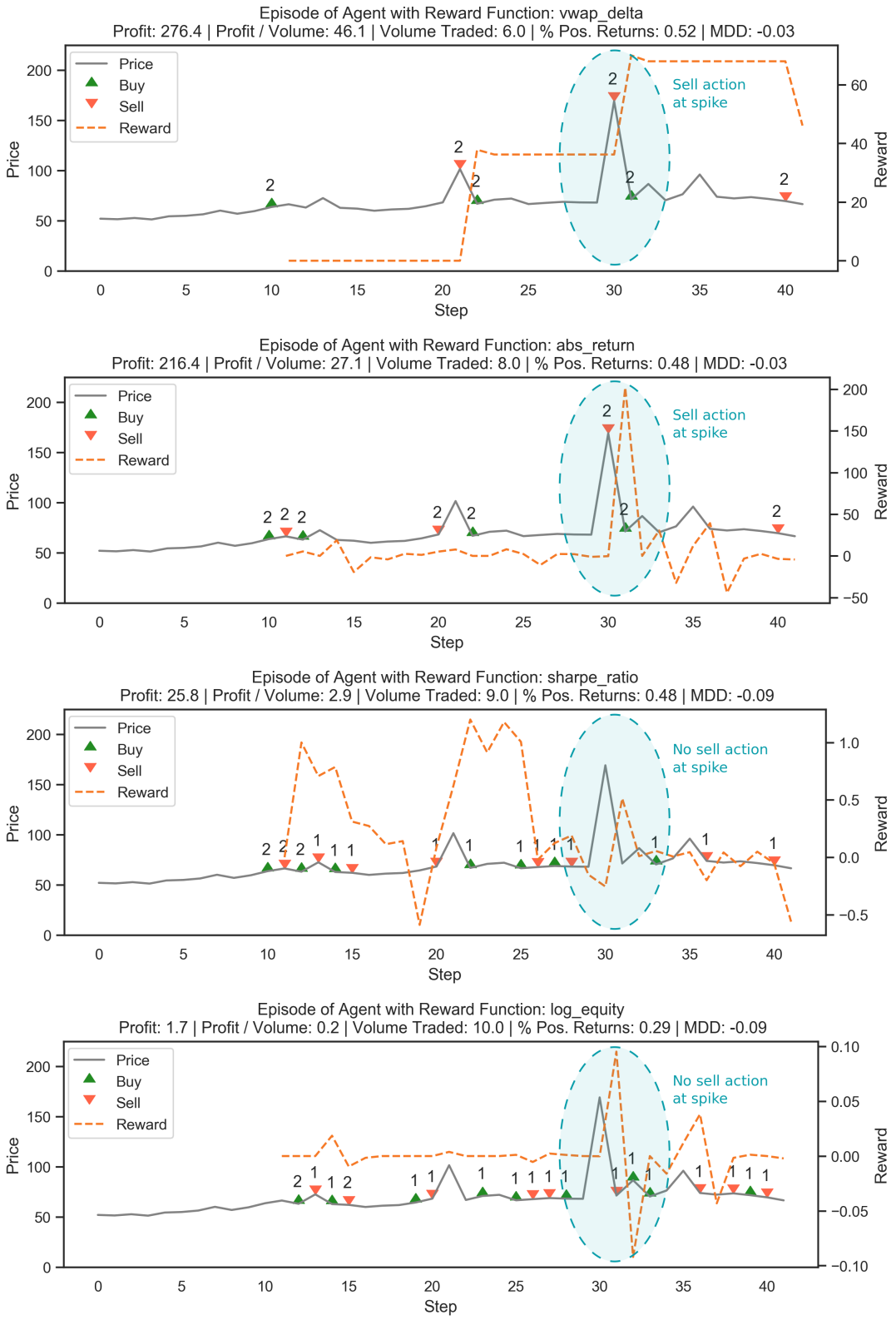


Figure 6.1: Question 1 - Example of episode with spikes for all 4 reward functions (test data). The figure illustrates that the agents trained with profit based reward functions (vwap delta and absolute return) tend to sell at the spikes, whereas the agents trained with risk-adjusted reward functions (Sharpe ratio and log equity) do not primarily tend to sell at the spikes.

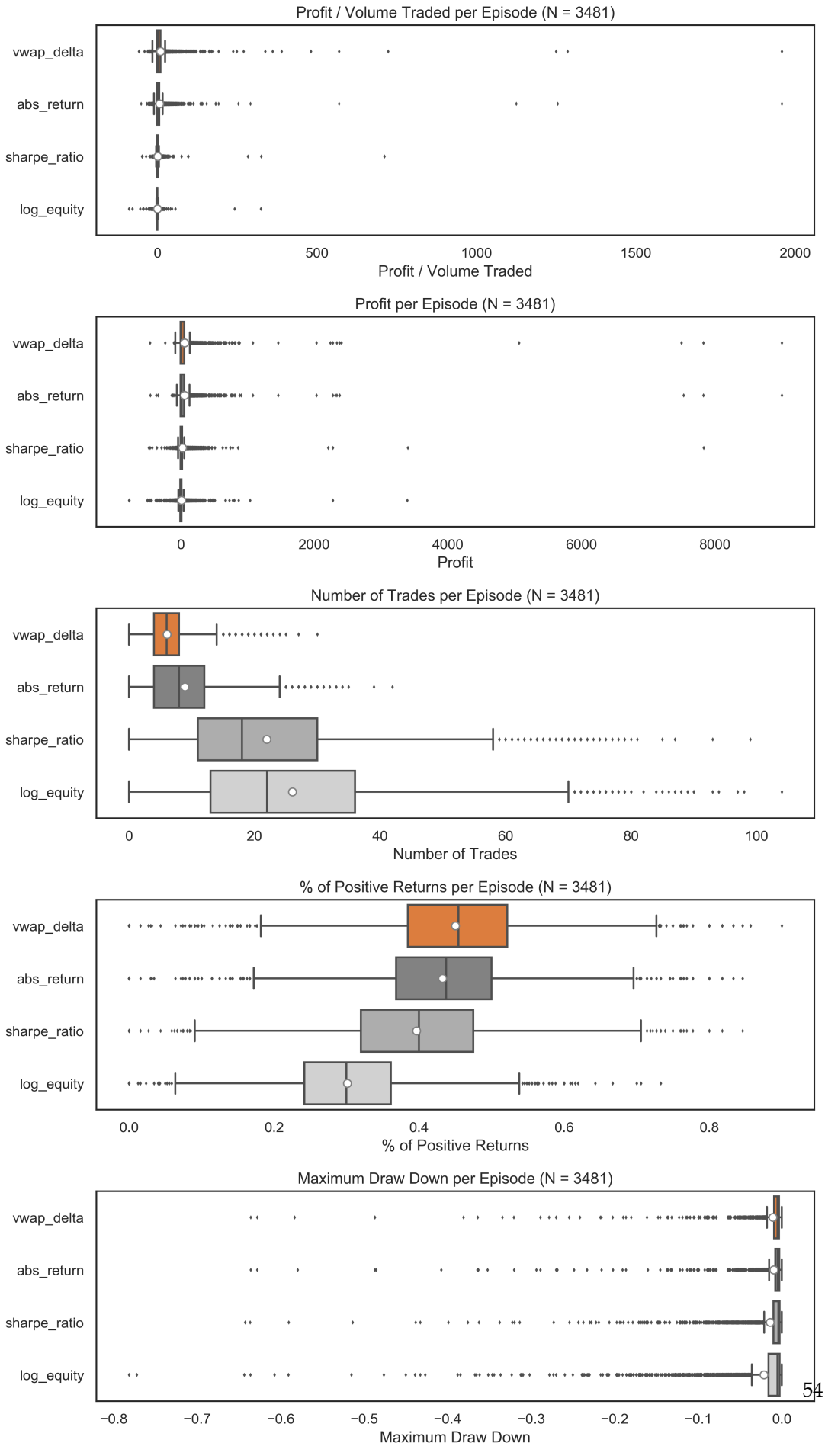


Figure 6.2: Question 1 - Distribution of evaluation metrics by reward type (test data)

6.1.2.2 Evaluation Metrics

Boxplots of the five evaluation metrics for all four agents are shown in figure 6.2. The white dots and black lines inside the boxes indicate means and medians respectively. The lower and upper ends of each box indicate the 25th and 75th percentiles respectively. The whiskers outside the boxes are based on a 1.5 interquartile range as described in [109].

Representing the *financial result*, the distributions of profit and profit per volume traded are more skewed to the right for the profit based reward functions than for the risk-adjusted reward functions. This illustrates that, compared to the risk-adjusted reward functions, the profit based reward functions had noticeably more episodes where they achieved a large financial result. Representing the level of *activity*, the distributions of the number of trades illustrate that the profit based reward functions result in fewer trades than the risk-adjusted reward functions. Representing *quality of trades* as investment, the distributions of the percentage of positive returns show that the profit based reward functions generally have a larger percentage of timesteps where their equity value increases than the risk-adjusted reward functions. Representing *worst-case risk*, the distributions of MDD show that the worst timesteps per episode (in terms of the percentage of equity value lost) are generally *not* worse for the profit based reward functions than for the risk-adjusted reward functions. Rather, the worst-case risks seems to be slightly smaller for the profit based reward functions.

6.1.3 Training Evaluation

The training metrics to evaluate the learning progress during training are defined in section A.5 in the appendix. Figure A.7 shows the training metrics during the 500K steps of training for the first research question. Overall, the training metrics imply that the agents for all reward functions show similar convergence: they learn action preferences that increase their respective rewards. This implies that differences in performance across these agents exist because the agents' reward functions incentivize different actions and not because of differences in the training progress of the agents.

Most importantly, the first plot in figure A.7 illustrates that the *discounted rewards* increase across the training period for all four reward functions. This implies that all agents manage to learn a policy that increases their respective discounted rewards. Recall that each agent has a different reward function and hence each agent learns to increase discounted rewards from a different reward function.

The second plot in figure A.7 illustrates that the *entropy* decreases across the training period for all four reward functions. This implies that the policies of all agents change over time so that the agents prefer to take certain actions in certain states. In other words, the policies change over time so that the agents are not indifferent about their actions anymore but instead have a preference for certain actions.

The third and the fourth plots in figure A.7, displaying the *policy gradient loss* and the *state-value loss*, do not show a clean decrease as under simplified circumstances in figure A.8. The *policy loss* does seem to decrease over time for all reward functions. However, the decreasing trend is disrupted in some steps, where the loss goes up again. The *state-value* loss seems particularly noisy. For the absolute return and the log equity reward functions, the state-value loss decreases quite steadily over time. For the vwap delta and the Sharpe ratio reward functions, however, there are periods where the state-value loss increases noticeably, followed by periods where the loss decreases noticeably again.

6.2 Methods (Question 2)

6.2.1 Bayesian Optimization

As described in section 5.7, 600 optimization trials were conducted for both, A2C and DQN, in order to find an optimal combination of optimization parameters (reward function, hy-

perparameters and features) for each of the two methods. The trained RL agent of each trial was evaluated by computing the median profit per volume traded on the validation data set, which is visualized in figure 6.3, for both methods and all their respective trials. The best set of optimization parameters was selected as the set corresponding to the trial where the RL agent performed best on the validation data, as measured by median profit per volume traded computed on the validation episodes. In particular, the best set was found at trial 178 for A2C and at trial 599 for DQN. The corresponding median profit per volume traded on the validation data was 5.16 for A2C and 5.67 for DQN.

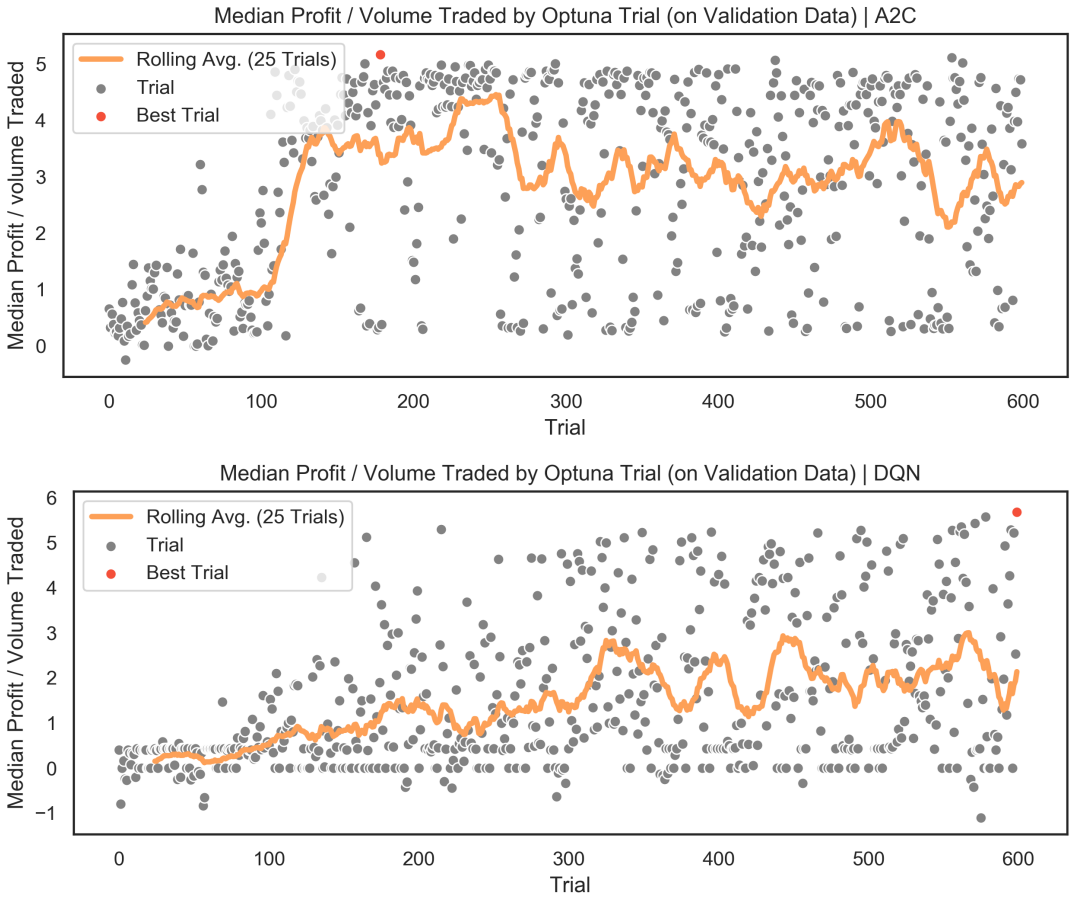


Figure 6.3: Median profit per volume traded by trial for A2C and DQN (validation data)

For both, A2C and DQN, the *reward function* vwap delta was selected in the best trial. This supports the conclusion from the first research question that the vwap delta reward function leads to better results with respect to the main evaluation metric, profit per volume traded. The *hyperparameters* for A2C and DQN that were selected in the best trial can be found in the appendix in tables A.15 and A.13 respectively. Lastly, the *features* selected for A2C and DQN in the best trial can be found in the table A.16. In total, 8 out of 21 features were selected for A2C, and 14 out of 21 features were selected for DQN. For both methods, features from all three categories (market, indicator and time features) were selected.

6.2.2 Statistical Evaluation

The distributions of profit per volume traded in EUR, based on 3841 *test* episodes, are described for the A2C agent and for the DQN agent in table 6.3. On the one hand, the A2C agent achieved a slightly larger median of profits per volume traded than the DQN agent,

along with a larger minimum and 25th percentile of profit per volume traded. On the other hand, the DQN agent achieved a larger mean profit per volume traded than the A2C agent, along with a larger maximum and 75th percentile of profit per volume traded.

Table 6.3: Distribution of profit per volume traded by model type (test data, n = 3481)

Model Type	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
A2C	3481.00	12.20	55.90	-36.69	-0.22	4.02	12.52	1957.21
DQN	3481.00	15.71	81.55	-92.36	-2.09	3.89	16.42	2521.85

The question is whether the median difference of profit per volume traded θ between the agent trained with A2C and the agent trained with DQN is significantly larger than 0. In fact, the two agents achieved the *exact same* profit per volume traded in 44% of the test episodes (1529 of 3481), resulting in differences of 0 for these episodes. With 1529 differences of 0, all boostrapped test statistics $\hat{\theta}_j^*$ took the value 0. Consequently, there was no variation in the bootstrapped test statistics and hence, neither the bootstrap confidence interval, nor the p-value for the boostrap hypothesis test could be computed as intended. Nevertheless, the finding that $\hat{\theta}_j^* = 0$ for all $j \in \{1, ..., 4999\}$ implies that we fail to reject the $H0 : \theta = 0$. Hence, there is not sufficient evidence to conclude that the median differences of profit per volume traded between A2C and DQN is larger than 0. In other words, it is not possible to conclude that A2C outperforms DQN with respect to the test statistic.

Table 6.4: Bootstrap hypothesis tests and confidence intervals for the difference in medians of profit per volume traded (test data, n = 3481, B = 4999)

Strategy_1	Strategy_2	$\text{median}(d_{1,2})$	Boot. 95% CI	Boot. SE	p-value
a2c	dqn	0.00	-	0.00	-

6.2.3 Descriptive Evaluation

6.2.3.1 Example Episodes

To provide an insight into the behavior of the agents trained with A2C and DQN, their actions are visualized for exemplary episodes, namely for an episode with *spikes*, an episode with an *uptrend* and an episode with a *downtrend*. The visualization of the episodes with spikes, uptrend and downtrend can be found in figures A.14 A.16 and A.17 in the appendix. In addition, tables A.3 and A.4 show the distributions of profit and of per volume traded by *model type* and *episode type*. The test episodes were categorized as described in section A.6.

Overall, the actions of the A2C and DQN agents trained with Bayesian optimization resemble the actions of the best A2C agent trained trained without Bayesian optimization from the first research question. In other words, the agents learn to sell at spikes. If there are no spikes, then the agents keep their volume until the end of the episode, selling it at the last timestep. The latter tactic is also called *buy-and-hold* strategy [110].

With Bayesian optimization, the performance of the A2C agent from the first research question could be improved in regards to mean and median profit per volume traded. While the A2C agent *without* Bayesian optimization sometimes fails to sell at spikes, the A2C agent *with* Bayesian optimization only rarely fails to sell at spikes, which explains the improvement in profit per volume traded.

Comparing the A2C agent with the DQN agent, both after Bayesian optimization, the A2C agent seems to sell at more spikes than the DQN agent. For instance, in the example episode A.14, the DQN agent does not sell at any of the spikes whereas A2C does. This explains the slightly larger median profit per volume traded for A2C in table 6.3. In the vast majority of

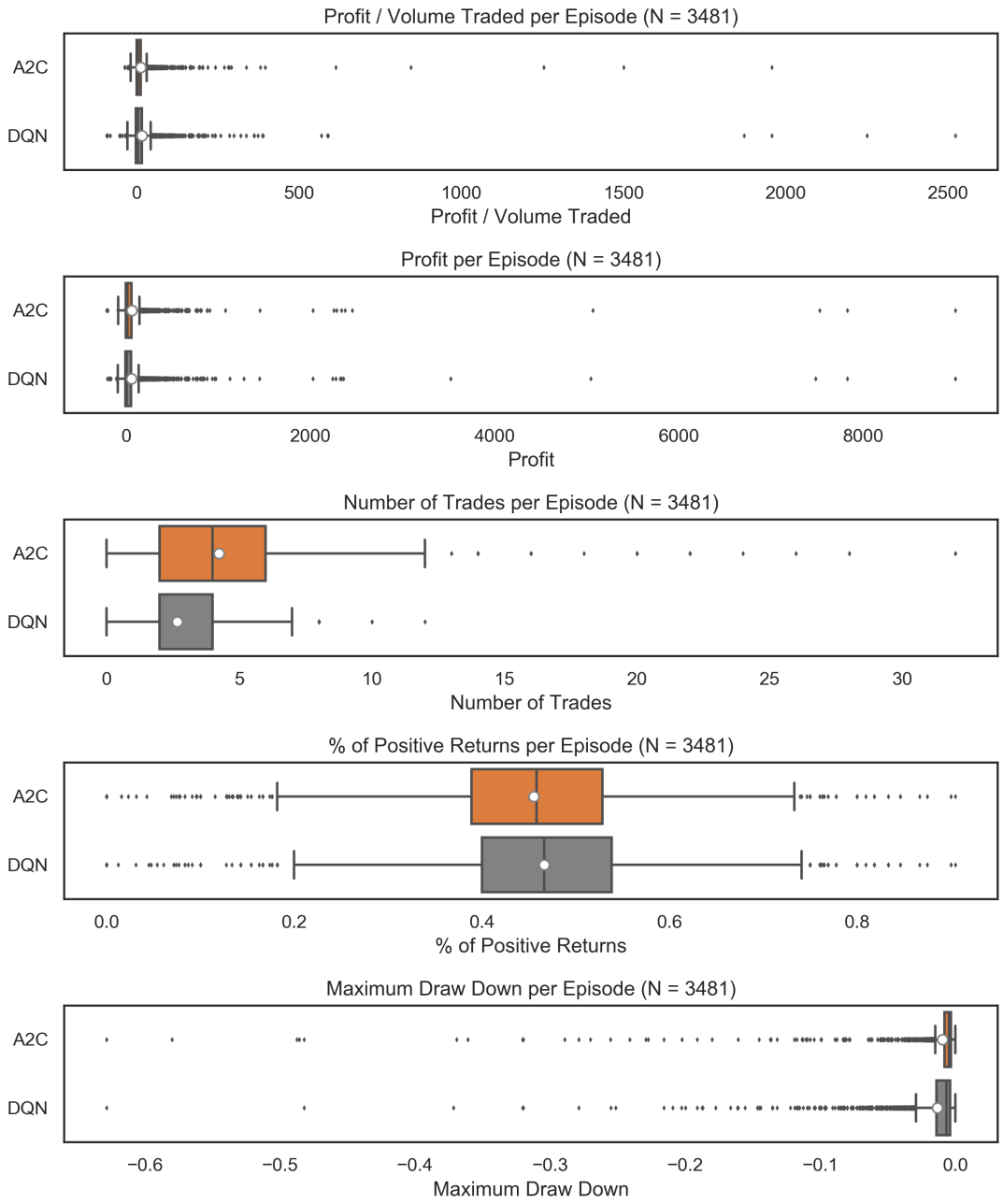


Figure 6.4: Question 2 - Distribution of evaluation metrics by model type (test data)

episodes however, both agents sell at all spikes and may even take the exact same actions like in the additional example episode A.15.

6.2.3.2 Evaluation Metrics

Overall, the performance of the A2C and DQN agents is very similar with respect to the evaluation metrics, as visualized in figure 6.4. The two main differences are the following: first, the A2C agent generally places more trades per episode than the DQN agent. For instance, it sells more often at the spikes as mentioned previously. Second, the A2C agent seems to

have slightly better MDD values than the DQN agent, which means that it generally looses a smaller percentage of equity value in the worst timesteps of the episodes.

6.2.4 Training Evaluation

The training metrics for the training period of the A2C agent and the DQN agent are displayed in figures A.9 and A.10 respectively. For reference, all training metrics are described in section A.5.1 in the appendix. The *training metrics for A2C* look as expected. Just like for the first research question, the A2C agent develops a preference for certain actions in certain states and thereby manages to increase its discounted rewards over time, which is shown by decreasing entropy and increasing discounted rewards. In addition, both policy loss and state-value loss decrease across the training period after increasing at first.

The *training metrics for DQN* show that the DQN agent manages to increase its rewards during the training period as well. However, along with increasing rewards also the action-value loss increases until the end of the training period. This suggests that further improvements may be possible with respect to the hyperparameters, e.g. the number of training steps. However, no further adjustments were made in this regard because the hyperparameters had been systematically determined with Bayesian optimization. Due to time consuming Bayesian optimization trials, no more than 600 trials were conducted.

6.3 Baseline (Question 3)

6.3.1 Statistical Evaluation

The distributions of profit per volume traded in EUR, based on 3841 *test* episodes, are described for each trading strategy in table 6.5. The RL agent is the optimized agent trained with A2C from section 6.2. This final RL agent outperforms the random, forecast and moving average trader baseline strategies in terms of median and mean profit per volume traded. However, it is outperformed by the spike trader baseline strategy. Nevertheless, the spike trader has a noticeably smaller minimum profit per volume traded than the RL agent.

Table 6.5: Distribution of profit per volume traded by trading strategy (test data, n = 3481)

Strategy	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
RL agent	3481.00	12.20	55.90	-36.69	-0.22	4.02	12.52	1957.21
Spike	3481.00	14.18	66.10	-161.46	-1.26	4.60	15.45	2250.00
Random	3481.00	0.17	9.14	-316.68	-0.66	0.00	0.76	250.00
Forecast	3481.00	0.70	11.73	-77.20	-0.87	0.00	0.99	344.51
Moving avg.	3481.00	-2.15	12.44	-171.55	-2.76	-0.61	0.69	216.64

The question is whether the median differences of profit per volume traded θ between the RL agent and the four baseline strategies are significantly larger or smaller than 0. The bootstrap p-values for the corresponding four hypothesis tests are shown in table 6.6. All p-values are smaller than the significance level (with $p < 0.013$) for the tests comparing the agent with the forecast, random and moving average trader. Hence all null hypotheses are rejected in favor of the alternative hypotheses for these tests. This means that the RL agent significantly outperforms the random, forecast and moving average trader baseline strategies.

Recall that the spike trader was implemented retrospectively in order to mirror the main strategy of the RL agent. When comparing the RL agent with the spike trader, neither the bootstrap confidence interval, nor the p-value for the boostrap hypothesis test could be computed as intended. This was because the agent and the spike trader achieved the *exact same* profit per volume traded in 45% of the test episodes (1579 of 3481), resulting in differences of 0 for these episodes. With 1579 differences of 0, all boostrapped test statistics $\hat{\theta}_j^*$ took the value

0. The finding that $\hat{\theta}_j^* = 0$ for all $j \in \{1, ..., 4999\}$ implies that we fail to reject the $H0 : \theta = 0$. Hence, there is not sufficient evidence to conclude that the median differences of profit per volume traded between the spike trader and the RL agent is larger than 0. In other words, it is not possible to conclude that the spike trader outperforms the RL agent with respect to the test statistic.

Table 6.6: Bootstrap hypothesis tests and confidence intervals for the difference in medians of profit per volume traded (test data, n = 3481, R = 4999)

Strategy_1	Strategy_2	median$(d_{1,2})$	Boot. 95% CI	Boot. SE	p-value
RL agent	Spike	0.00	-	0.00	-
RL agent	Forecast	3.55	(3.23, 3.84)	0.14	0.0014
RL agent	Random	3.75	(3.47, 4.10)	0.17	0.0000
RL agent	Moving avg.	3.94	(3.60, 4.29)	0.19	0.0000

6.3.2 Descriptive Evaluation

6.3.2.1 Example Episodes

To compare the actions of the RL agent with the actions of the four baseline strategies, three exemplary episodes are visualized: an episode with *spikes*, an episode with an *uptrend* and an episode with a *downtrend*. The visualization of the episodes with spikes, uptrend and downtrend can be found in figures A.18 A.19 and A.20 in the appendix.

Figure A.18 shows an episode with *spikes*. The figure illustrates that both, the RL agent and the spike trader, sell at the spikes and buy after the spikes. The random trader sometimes sells at the spikes by chance. The forecast trader tends *not* sell at the spikes because of inaccurate price forecasts. The moving average trader tends to *buy* at the spikes, assuming that the price will further increase, which however almost never happens.

Figure A.19 shows an episode with *uptrend*. The figure illustrates that both, the RL agent and the spike trader, buy some volume at the beginning and hold it until the end of the episode, using a *buy-and-hold* strategy [110]. The random trader again places random trades throughout the episode. In this episode, the forecast trader does not trade because all forecasts are in the range $[-3\%, 3\%]$. In other episodes with uptrend however, it does trade occasionally if forecasts are $> 3\%$ or $< -3\%$. In this episode, the moving average trader buys some volume at the beginning of the episode and holds it until the end of the episode, similar to the RL agent and spike trader. However, the moving average trader would sell in the middle of an uptrend if there were timesteps with decreasing price *during* the uptrend.

Figure A.20 shows an episode with *downtrend*. The figure illustrates that both, the RL agent and the spike trader, again use the buy-and-hold strategy. While this strategy leads to profits in episodes with uptrend, it results in a loss for episodes with downtrend. The random trader places random trades throughout the episode. The forecast trader occasionally places trades if forecasts are $> 3\%$ or $< -3\%$. The moving average trader does not trade during downtrends unless there are timesteps with increasing price *during* the downtrend.

In addition, tables A.5 and A.6 in the appendix show the distributions of profit and of per volume traded by *trading strategy* and *episode type*, where the test episodes were categorized as described in section A.6. These tables back up the main conclusions drawn from the visualizations of example episodes above: the RL agent and spike trader outperform the baseline strategies in episodes with spikes and uptrend because they sell at spikes and hold their investments during uptrends, which results in profits. However, they are outperformed by the baseline strategies during downtrends because they again hold their investments, which results in losses.

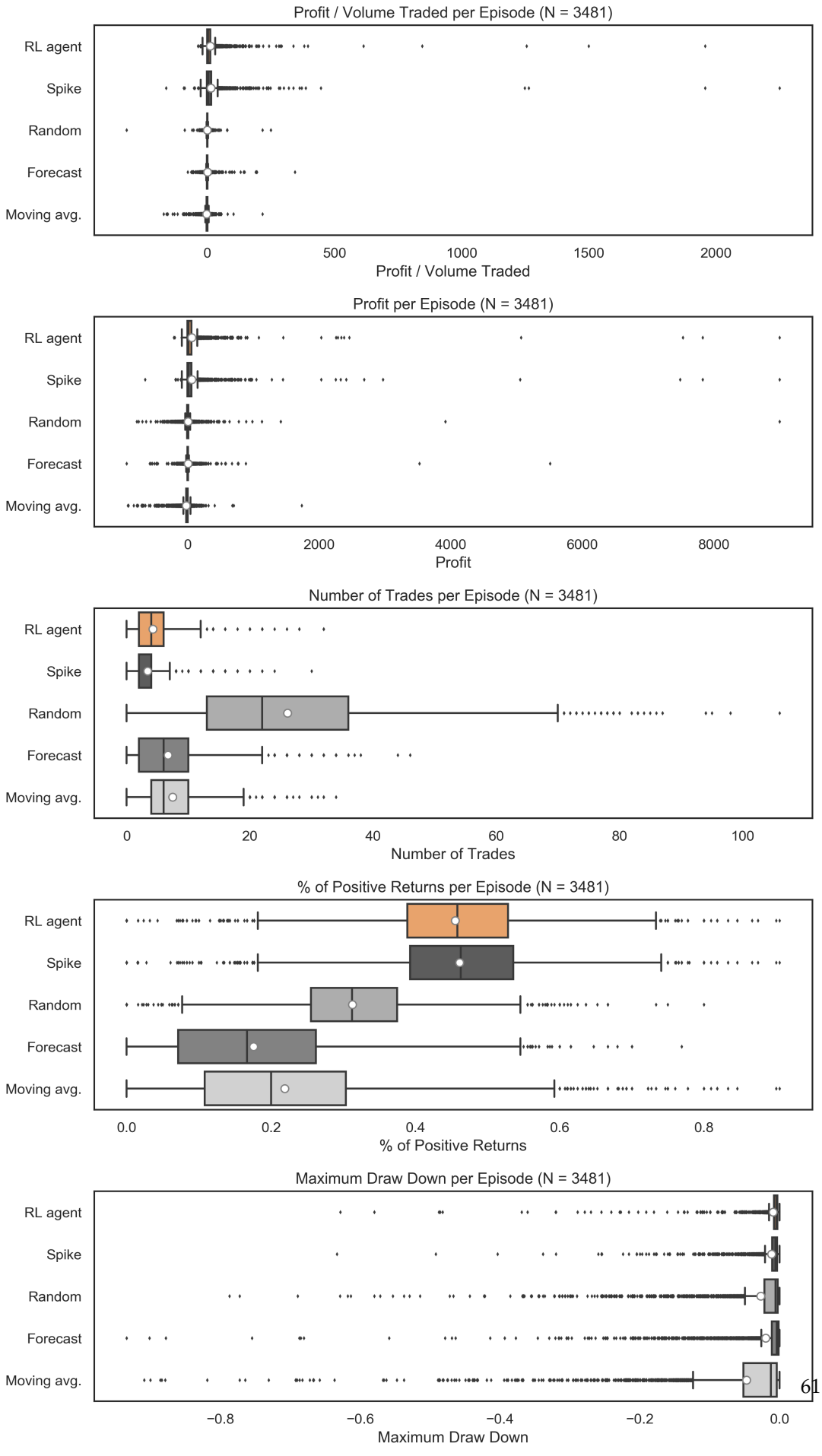


Figure 6.5: Distribution of evaluation metrics by baseline strategy (test data)

6.3.2.2 Evaluation Metrics

Boxplots of the five evaluation metrics for all four agents are shown in figure 6.5. Representing the *financial result*, the distributions of profit and profit per volume traded are more skewed to the right for the RL agent and the spike trader. This illustrates that, compared to the other baseline strategies, the RL agent and the spike trader had noticeably more episodes where they achieved a large financial result.

Representing the level of *activity*, the distributions of the number of trades illustrate that the spike trader generally takes the smallest number of trades per episode. Hereby, the spike trader is followed closely by the RL agent, the forecast and the moving average trader.

Representing *quality of trades* as investment, the distributions of the percentage of positive returns show that the RL agent and the spike trader generally have a larger percentage of timesteps where their equity value increases than the other baseline strategies.

Representing *worst-case risk*, the distributions of MDD show that the worst timesteps per episode (in terms of the percentage of equity value lost) are generally *not* worse for the RL agent and the spike trader than for the other baseline strategies. Rather, the opposite seems to be the case since the distributions seem slightly less skewed to the left for the RL agent and the spike trader than for the other baseline strategies.

7 Discussion

In this chapter, the results and methods are discussed for each of the three research questions. Above all, the three research questions are answered. Section 7.1 addresses the first research question about the reward function comparison, section 7.2 is dedicated to the second research question about the method comparison and section 7.3 focuses on the third research question about the baseline strategy comparison. Importantly, the conclusions are limited to the problem of intra-day trading in the electricity domain, which was addressed in this thesis.

7.1 Reward (Question 1)

***Reward**: Is the Sharpe ratio, a prevalent reward function from the financial domain, the best reward function for intra-day trading in the electricity domain?*

To answer this research question, four agents were trained with different reward functions. In particular, two risk-adjusted reward functions (Sharpe ratio and log equity ratio) and two profit based reward functions (vwap delta and absolute return) were implemented. The corresponding four agents were all trained under the same circumstances (e.g. default hyperparameters and features), however with different reward functions. Each agent was evaluated on the same 3481 test episodes. The performance of the four agents was compared based on the main evaluation metric (profit per volume traded), computed for each test episode.

7.1.1 Results

Based on the results, the above research question can be answered with *no*. The best results in terms of median profit per volume traded were obtained with the profit based reward function *vwap delta*, followed by the absolute return, Sharpe ratio and log equity ratio reward functions (in this order). Moreover, the vwap delta reward function was found to significantly outperform the Sharpe ratio and the log equity ratio. Since the Sharpe ratio is clearly outperformed by the vwap delta reward function, it does not seem to be the best reward function for intra-day trading in the electricity domain. Overall, the profit based reward functions (vwap delta and absolute return) led to better results in regards to the main evaluation metric, profit per volume traded, than the risk-adjusted reward functions (Sharpe ratio and log equity ratio).

This raises the question *why* reward functions that adjust for risk, such as the Sharpe ratio, seem less appropriate for intra-day trading in the electricity domain. After all, the Sharpe ratio has been a popular reward function in the financial domain [2]. The answer may lie in the properties of electricity. While financial assets such as stocks, indices and funds are investments that are expected to increase in value over a rather long time period, electricity is a product with a *very volatile price* that heavily depends on the supply and demand for a given time period such as the hour of the day. If the supply exceeds the demand, e.g. due to unexpectedly large production of wind energy, the prices may become negative. If the demand exceeds the supply, e.g. due to people and companies using more electricity, prices may spike up massively. In summary, unexpected variations in supply and demand result in price volatility. Trading agents that learn to make use of this volatility can be very successful. As illustrated in figure 6.1, the agents trained with the profit based reward functions, vwap delta and absolute return, learned to sell electricity when the price spikes up. Thereby, they learned to make use of the price volatility. This was possible because their reward functions gave large positive rewards for selling when the price spikes up, reinforcing such behavior.

The reward functions that adjust for risk, in contrast, did not reward such behavior in the same way. Recall that the *Sharpe ratio* reward function is defined as the ratio of the running sample mean of returns over the running sample standard deviation of returns as defined in equation 2.44. If the agent sells when the price spikes up, the mean in the numerator increases. However, also the standard deviation in the denominator increases. Therefore, selling at spikes generally does not result in large rewards, which means that the Sharpe ratio does not encourage the behavior of selling at spikes. The *log equity ratio* reward function, defined in equation 2.49, is considered a risk-adjusted reward function because of its convex shape corresponding to a risk-avoiding utility function (illustrated in figure 2.6). Due to its shape, this reward function strongly punishes actions followed by a large percentage *decrease* in equity. Importantly, it gives very similar rewards to actions followed by some small percentage *increase* and some small percentage *decrease* in equity. In other words, this reward function does not give a strong enough incentive to avoid small percentage decreases in equity. It seems like this causes not only a smaller percentage of timesteps with positive returns but also a larger number of trades as shown in figure 6.2. This has a negative impact on the main evaluation metric, which is high if *large* profits are obtained with a *small* traded volume.

7.1.2 Method

Three methodological aspects need to be discussed: the lack of hyperparameter optimization and feature selection, the specific implementation of the reward functions and the choice of the main evaluation metric.

The four agents were trained with their respective reward functions using all features from table 4.2.1 and default hyperparameters from table A.15. Ideally, each agent should have been trained with features and hyperparameters *optimized* for the respective reward function to ensure that each agent achieves its full potential. However, the training metrics for the agents help to alleviate concerns. They do not seem to indicate that the agents trained with risk-adjusted reward functions (Sharpe ratio and log equity) were put at a disadvantage due to the default features and hyperparameters. The learning curves in figure A.7, in particular the curves showing the entropy loss and the discounted rewards, show that all agents developed a preference for certain actions and thereby increased their respective discounted rewards over the training period. Hence, the fact that all agents were trained with default features and hyperparameters does not seem to diminish the credibility of the results.

Furthermore, small adjustments in the reward functions might have possibly led to different results. For instance, the Sharpe ratio reward function was computed based on running averages of mean and standard deviation across 5 timesteps. Using a different number of timesteps could have resulted in a different performance of the Sharpe ratio reward function. However, different studies also made similar choices. For instance [32] and [33] used 5 and 21

(or 22) timesteps for the number of days in a week and month. Similarly, note that it would have been possible to compute the log equity ratio reward function and the profit based reward functions with moving averages. Understanding how different modifications of each of the reward functions affect the performance (and possibly training stability) would make for an interesting research question in the future. In the scope of this thesis, however, only one version of each reward function was implemented.

Lastly, note that the main evaluation metric of this thesis, profit per volume traded, was based on profit. The reward functions that performed particularly well on this metric were profit based as well. Whether a certain reward function is considered suitable or not depends on the chosen evaluation metric. For instance, profit based reward functions might have shown a worse performance compared to risk-adjusted reward functions if different evaluation metrics had been used for the comparison. Hence, it is important to stress that the experiments found profit based reward functions to perform better than risk-adjusted reward functions, not universally but *with respect to the evaluation metric* profit per volume traded.

7.1.3 Outlook

Future research could investigate specific characteristics of the reward functions, in particular the implications of using *moving averages* in reward functions, of using *sparse rewards* in form of the main evaluation metric at the end of an episode and of applying methods for improving performance if rewards are sparse. These three suggestions are discussed below.

First, one should understand whether reward functions based on *moving averages* can generally increase training stability and improve results. Note that the best reward function in this thesis, *vwap delta*, is based on the difference between two cumulative moving averages: the volume weighted average selling price and the volume weighted average buying price. Thereby, the reward at a given timestep is dependent on past timesteps as well since the moving average is also computed with past metrics. Hence, if an outstandingly good (or bad) action is taken in one timestep, this will result in large (or small) rewards for not only this action but also for subsequent actions. This is illustrated in figure 6.1, where the orange dotted lines show the rewards. Note that the vwap delta rewards resemble a staircase: after successful actions, the rewards stay large for several timesteps. This is different for the rewards of the remaining reward functions, in particular absolute return and log equity, which do not use any moving averages.

Second, one should consider giving non-informative *constant rewards* during the episode and only a *single non-constant reward* at the end of the episode. This resembles the simplistic maze problem, where the agent receives constant negative rewards at each step, except for the last timestep at which it receives a single positive reward for finding its way out of the maze. In this thesis for instance, one could give zero valued rewards during the episode, except for the last timestep where the main evaluation metric, profit per volume traded, would be given as reward. On the one side, this will make it more difficult to learn which actions affected the main evaluation metric positively and to reinforce these actions accordingly. In other words, the *credit assignment* problem will be harder to solve. On the other side, this might help to actually learn actions that maximize the *main evaluation metric* as desired. Recall that RL agents learn to optimize return, i.e. the sum of discounted future rewards shown in equation 2.4, which can be very different than the main evaluation metric depending on the chosen reward function. If the main evaluation metric is only given at a single timestep at the end of the episode (possibly even with a discount factor of 1), then the true sum of future expected rewards is equal to the main evaluation metric. Consequently, the agent will directly optimize for the main evaluation metric.

Lastly, if only single non-constant rewards are given as described above, one may apply and further develop methods that improve training if rewards are sparse. For instance, *hindsight experience replay* was introduced for exactly this issue [111]. However, hindsight experience replay cannot be applied *as is* for the context of this thesis. It was mainly developed

for environments, where there is a particular *goal state* (e.g. a location on the map that the agent needs to reach). Unfortunately, there is generally no particular goal state in the context of trading. Instead, the goal is to maximize a certain metric such as profit per volume traded. Nevertheless, it may be possible to adjust the idea of hindsight experience replay for environments *without* a particular goal state.

7.2 Method (Question 2)

***Method**: Does the RL agent achieve a better performance when trained with an actor-critic method than when trained with a critic-only method?*

To answer this research question, two agents were trained with different RL methods. In particular, one agent was trained with the actor-critic method *A2C* and another agent was trained with the critic-only method *DQN*. Both agents were trained with features, hyperparameters and reward function selected by Bayesian optimization. The agents were each evaluated on the same 3481 test episodes. The performance of the two agents was compared based on the main evaluation metric (profit per volume traded), computed for each test episode.

7.2.1 Results

Based on the results, the above research question has to be answered with *no*. In this thesis, A2C was used as representative for actor-critic methods and DQN was used as representative for critic-only methods. The agents trained with A2C and DQN both learned to sell at the spikes and to use a buy-and-hold strategy if there are no spikes. Both agents achieved the exact same profit per volume traded in 44% of the test episodes. As for the statistical evaluation, there was not sufficient evidence to conclude that the A2C agent outperforms the DQN agent with respect to profit per volume traded. Importantly, these findings are limited to the RL environment of this thesis. Possibly, different conclusions might be reached with different, for instance more complex environments.

Beyond performance, other factors may be considered for the choice of the RL method. For instance, A2C may be preferred because of its stochastic policy and its support for both, discrete and continuous, action spaces. In contrast, DQN uses a deterministic policy and is limited to discrete action spaces. Moreover, finding an optimal set of reward function, features and hyperparameters was faster for A2C than for DQN (as shown in figure 6.3).

7.2.2 Method

Three methodological aspects need to be discussed: the issue of overfitting the validation data during Bayesian optimization, the possibility of better DQN performance after even more Bayesian optimization trials and the interpretation of the results with respect to actor-critic versus critic-only methods.

First, there might be some concern about the use of validation data for Bayesian optimization. Recall that each set of optimization parameters (features, hyperparameters and reward function) was assessed by evaluating the corresponding agent on the validation data. Moreover, the best set of optimization parameters was selected as the set that performed best on the validation data. This approach entails the risk that the selected parameters do not generalize well to unseen test data. This risk might have been reduced by using cross validation instead. Although no cross validation was applied in this thesis, it seems like the selected optimization parameters generalized quite well to the test data. For A2C, the median profit per volume traded was 5.16 on the validation and 4.02 on the test data. For DQN it was 5.67 on the validation data and 3.89 on the test data. Figure 4.4 shows that the price distributions of the validation and test data are similar, which may explain the decent generalization to the

test data. Nevertheless, it would be better to conduct optimization with cross validation, in particular for RL agents that will be used in real-world applications.

Second, one should point out that 600 trials of Bayesian optimization may not have been sufficient for DQN. The following observations support this concern: first, the best trial was the second to last trial (trial 599 of 600 trials in total). Second, figure 6.3 shows that for DQN the validation performance was improving throughout the whole training period, whereas for A2C it was only improving until trials 150-200 and remaining stable from thereon. Third, the action-value loss was increasing for DQN throughout the whole training period as shown in figure A.10. Therefore, it is possible that a better set of optimization parameters, resulting in better performance on validation and test data, could have been identified with more Bayesian optimization trials. For this thesis, no further trials were conducted because of time reasons. One trial took approximately between 5 and 75 minutes depending on the optimization parameters selected for the trial.

Lastly, the relevance of the three method classes (critic-only, actor-only and actor-critic) needs to be discussed. While it is possible to categorize RL methods as critic-only, actor-only and actor-critic, it is important to note that certain methods became successful and popular because of *additional* ideas that go beyond this categorization. For instance, the success of the critic-only method DQN did not come because of its nature of being critic-only. Instead, its success in playing Atari Games was due to experience replay and a separate target network [24]. Later, DQN was further improved by additional ideas such as prioritized experience replay [25], dueling networks [26] and double Q-learning [27]. Similarly, the success of A2C did not come because of its nature of being actor-critic. Instead, its success came with the idea of letting multiple agents gain experience on different copies of the environment simultaneously [4]. As a result, it may not be appropriate to attribute differences in performance between certain RL methods to their nature of being critic-only, actor-only or actor-critic. Instead, it may be more appropriate to look at RL methods as individual methods, not as representatives of these method categories. For instance, it may be more suitable to compare A2C with DQN instead of comparing actor-critic methods *represented* by A2C with critic-only methods *represented* by DQN.

7.2.3 Outlook

In this thesis, there was not sufficient evidence to show that A2C outperforms DQN. Interestingly, a survey about reinforcement learning for trading in the financial domain also came to the conclusion that there is "no clear winner" [15, p. 36] among any of the three RL method categories (critic-only, actor-only, actor-critic). This conclusion was, among others, based on the fact that different reviewed papers reported different method categories to be superior.

Among others, differences in performance may be explained with the following reasons: first, it is possible that differences in performance were found only in *descriptive* statistics but not with *inferential* methods such as hypothesis tests, which could have shown differences to be insignificant. Second, it is possible that performance differences between RL methods were due to insufficient optimization of features, hyperparameters and rewards, putting certain RL methods at a disadvantage. Third, it is possible that certain RL methods systematically work better for certain RL environments. Clearly, future research is needed to better understand differences in performance of certain RL methods for trading.

First, research is needed to establish a standard for hypothesis testing in reinforcement learning for trading. In particular, with actions coming from a stochastic policy in some RL methods, it is not sufficient to plainly compare RL methods based on descriptive statistics. Nevertheless, most studies do exactly that. Clear guidelines for robust hypothesis testing in reinforcement learning would help to improve this status quo. Hereby, resampling methods such as bootstrap hypothesis tests might be a good starting point.

Second, studies are needed to exhaustively benchmark RL methods *with Bayesian optimization* on the exact same RL environments. Bayesian optimization of features, hyperparameters

and reward functions will ensure that each RL method can realize its full potential, allowing for a fair comparison of different RL methods. The findings of this thesis suggest that differences in performance between RL methods may be insignificant: the two applied methods A2C and DQN learned very similar strategies, resulting in an insignificant performance difference. Moreover, this thesis has shown that successful strategies in the context of trading may be simple, as reflected by the *spike trader*, which always holds some volume of electricity and sells at spikes. Such *simple* strategies may be discovered by various different RL methods after sufficient optimization of their hyperparameters, features and reward functions. If different RL methods are able to learn these strategies, there may not be any significant difference in performance between these RL methods.

Third, studies are needed that systematically compare RL methods on multiple environments, which are similar in nature but differ in their level of complexity. Such a comparison will help to better understand if there are certain characteristics of an RL environment for which specific RL methods are more or less suitable. For instance, *level of complexity* might be adjusted by varying episode length, size of the action space, size of the state space and randomness in the environment. Different characteristics may be compared as well, e.g. whether the environment is dynamic or static (with next states s' depending on action a or not).

7.3 Baseline (Question 3)

***Baseline**: Does the RL agent achieve a better performance in trading electricity than baseline strategies that are commonly used for trading?*

To answer this research question, the optimized A2C agent from the second research question was compared with four trading strategies of different nature: a *random* trader, a *forecast* trader, a *moving average* trader and a *spike* trader. The spike trader was implemented retrospectively to match the A2C agent's strategy of selling at price spikes. All trading strategies were each evaluated on the same 3481 test episodes. Their performance was compared based on the main evaluation metric (profit per volume traded), computed for each test episode.

7.3.1 Results

Based on the results, this question can be answered with *yes*. The RL agent was able to significantly outperform the random, forecast and moving average baseline strategies in terms of profit per volume traded. There was not sufficient evidence to conclude that the RL agent is better or worse than the spike trader. However, the spike trader was designed *retrospectively* to match the behavioral strategy of the RL agent, *after* analyzing its actions. Therefore, the spike trader itself does not actually represent a common baseline strategy. Although the spike trader resembles the general idea of "Buy low, sell high" [112], it differs from this idea because the spike trader only tries to sell high and is indifferent about the buying price.

Nonetheless, the great performance of the spike trader illustrates that successful strategies may be quite *simple* in the context of trading. Furthermore, this thesis illustrates that reinforcement learning may be used to identify such trading strategies: after understanding the strategy of a successful RL agent, this strategy may be converted to a rule-based algorithm just like the spike trader. Converting the RL agent to a rule-based algorithm reduces complexity. This may result in better generalization to unseen data, better interpretability, and easier deployment to real-world trading applications. In the context of neural networks, it has been shown that the knowledge of complex and ensemble models can be transferred to simpler models [113]. Converting the strategy of an RL agent to a rule-based algorithm resembles this idea. While in this thesis, the rules were simply identified based on visual inspection, it would be better to make use of a systematic approach for converting the RL agent's trading strategy into rules. A particularly promising approach might be to train a decision tree that predicts the actions taken by the RL agent [114].

Although the spike trader attempts to mirror the RL agent's strategy of selling at spikes, both trading strategies still differ in certain aspects. This is because the spike trader was not designed to *exactly* match the policy of the RL agent but only to match the *particular* strategy of selling at large price increases. It is of particular interest to compare the RL agent with the spike trader since this may help to understand advantages and disadvantages of the RL agent. For this purpose, all test episodes for RL agent and spike trader were visualized. This was done in an exploratory manner, without quantifying the results. The main objective was to understand why the RL agent performs better or worse in certain cases. As a result, two aspects in favor of the RL agent and two aspects in favor of the spike trader were identified.

7.3.1.1 Advantages of RL agent

As shown in tables A.5 and A.4 in the appendix, the RL agent outperforms the spike trader in *no trend* episodes. As described in section A.6, these episodes correspond to episodes that were *not* categorized as spike, uptrend or downtrend episodes. Importantly, these episodes may still contain some form of spikes, uptrend or downtrend. For instance, *no trend* episodes may contain *small* spikes with a percentage increase $\leqslant 25\%$ or spikes with a pique that lasts over several timesteps. This is because such spikes would not meet the categorization rules for *spike* episodes from section A.6 and may therefore be categorized as *no trend* episodes instead. Note that according to the categorization rules, an episode is categorized as *spike* episode if it contains at least 1 sequence of 3 timesteps $t-1, t, t+1$, where the price increases by $\geqslant 25\%$ from $t-1$ to t and decreases by $\leqslant -25\%$ from t to $t+1$. There are two reasons why the RL agent is generally superior in the *no trend* episodes.

First, the RL agent also sells at spikes that correspond to a percentage increase $< 25\%$, whereas the spike trader only sells at percentage increases $\geqslant 25\%$. Although percentage increases $\geqslant 25\%$ also exist in no trend episodes, they are rather rare. In particular, such percentage increases can only occur in *no trend* episodes if they are not directly followed by a percentage decrease of $- \leqslant 25\%$ in the subsequent timestep. Otherwise, the corresponding episodes would be categorized as spike episodes. In conclusion, the RL agent learned to sell not only at *large* price increases but also at *small* price increases. This puts the RL agent at an advantage in episodes with *only* small price increases because the RL agent sells at these small price increases, whereas the spike trader does not.

Second, after selling at a price increase, the RL agent generally waits until the price *decreases* before buying again, whereas the spike trader immediately buys after selling. There exist episodes, where the price further increases or stays approximately the same after a $\geqslant 25\%$ price increase. Few timesteps later however, the price generally decreases strongly. Visually, one may think of this as an extended spike or a spike that stays at its pique for several timesteps. Since the spike trader buys *directly* after it sells, it buys when the price is high in such scenarios. This generally gives poor results because the price tends to drop shortly after. In conclusion, the spike trader sometimes buys electricity at a high price, whereas the RL agent makes an effort to buy electricity at a low price: the RL agent waits until the price decreases and then it buys some volume of electricity.

7.3.1.2 Advantages of spike trader

As shown in table A.5, the spike trader slightly outperforms the RL agent in regards to the main evaluation metric on episodes with *spike* and with *uptrend* . There are two reasons why the spike trader is generally superior in these episodes.

First and most importantly, the spike trader places very few trades, fewer trades than the RL agent. This is illustrated by the third plot in figure 6.5, showing the distribution of the *number of trades per episode* across all test episodes. The spike trader places fewer trades because it *only* sells when the price increases by $\geqslant 25\%$, whereas the RL agent also sells when the price increases by a smaller percentage. Due to fewer trades, the denominator in the main

evaluation metric, profit per volume traded, displayed in equation 5.2 is generally smaller for the spike trader than for the RL agent. If the RL agent places the exact same trades as the spike trader but *in addition* sells at a *small* spike, then the profit per volume traded will be smaller for the RL agent than for the spike trader. This is because the additional trade will result in only a small profit relative to its volume, decreasing the overall profit per volume traded for the episode. In other words, the RL agent achieves *worse* results because of taking additional *appropriate* actions (selling at small spikes). Mainly, this issue illustrates a limitation of the main evaluation metric: it puts strong emphasis on the amount of volume traded, which in some cases punishes actions even though they result in incremental profit. Perhaps, it might be good to revise the main evaluation metric *profit per volume traded* in future research so that additional trades are rewarded, not punished, if they result in some small but noticeable incremental profit.

Secondly, the spike trader performs better in some episodes with *spikes* because the spike trader *always* sells at the spikes whereas the RL agent sometimes misses out on selling at spikes. This happens rather rarely however.

7.3.2 Method

The issue of the main evaluation metric has already been addressed when discussing the advantages of the spike trader in section 7.3.1.2. From a methods view, two further aspects might be discussed. First, the choice of the baseline strategies and second the implementation of the baseline strategies.

First, there may be some concern that these baseline strategies are not appropriate for the given problem of trading volatile electricity prices. In particular the performance of the random, forecast and moving average trader was rather poor. However, the amount of possible baseline strategies from the financial domain is large and not all baseline strategies could be considered. The forecast trader was selected because it had been applied as baseline strategies in previous studies about reinforcement learning for trading, for instance in [16]. The random trader was selected because in the financial domain, it was shown to give a similar performance as several other baseline strategies based on indicators [96]. The moving average trader was selected as a particular representative of indicator based trading strategies.

Second, there may be some concern about the implementation of the baseline strategies. In regards to the forecast trader from section 5.4.2, one may question why it buys when the price is predicted to increase by $> 3\%$ and not by for example $> 5\%$ or any other percentage. In regards to the moving average trader from section 5.4.3, one may question why the moving average was computed based on 10 timesteps and not any other number of timesteps. Also in regards to the spike trader, on may question why this trader only sells when the price increases by $\geqslant 25\%$. Ultimately, the answer is that alternative implementations would have well been possible. However, it would have been out of scope to optimize the baseline strategies with respect to their implementational details. At last, the main limitation of baseline strategies for trading *is* that they are based on simple heuristics and therefore generally not optimal for the problem at hand.

7.3.3 Outlook

Future research about converting policies of successful RL agents into rule-based algorithms would be very valuable. In particular in the context of trading, such an approach would make sense. This is because of the following reasons. First, it seems *realistic* that successful trading strategies can be expressed as rule-based algorithms, mainly because they may be relatively simple (unlike the strategies of RL agents that play Go, Dota 2 and StarCraft II [5]–[7]). Second, *speed* of execution matters in real-world trading applications and rule-based algorithms may be faster than complex RL agents based on neural networks. Third, *interpretability* is important because it may help to manage risk of a real-world trading application. For instance,

if a black-box RL agent suddenly starts taking losses several episodes in a row, it may be difficult to understand why that is. If a rule-based algorithm starts taking losses, one may immediately understand the reasons for the losses and make a judgement whether the agent should keep trading or not. In future research, different approaches for converting successful RL agents into rule-based algorithms should be considered. A promising approach may be to train decision trees that predict the actions of the RL agents. Complexity might, for instance, be regulated by pruning the decision trees and by using a reduced state space.

Furthermore, different baseline strategies may be applied. In particular, baseline strategies in accordance to the idea "Buy low, sell high" [112] may be suitable because such strategies would make use of the volatility in the electricity market. These strategies are based on the assumption that the price is *mean-reverting*, which means that the price will always come back to its mean after an increase or a decrease [115]. Modeling electricity prices as mean-reverting processes was done before, for instance in [116]. Such models might helpful for designing baseline trading strategies that buy when the price is low and sell when it is high.

7.4 Transfer into practice

Overall, the results of this thesis seem promising because the final RL agent was able to learn a successful, profitable trading strategy. This implies that reinforcement learning is particularly suitable for identifying successful trading strategies. In contrast, supervised learning is not predestined for identifying new trading strategies. Supervised methods perform predictions, without addressing the question how these predictions should be turned into actions.

While the performance of the final agent, achieving a median profit per volume traded of 4.02 EUR on test data, seems impressive, it is not representative of the performance that can be achieved in a real-world application. It is important to consider overall limitations of this thesis when attempting to apply RL for a real-world trading application. Therefore, this section addresses these limitations. In particular, limitations of the environment, the evaluation and the model are discussed.

The *environment* of this thesis was simplified in several aspects. First and foremost, every order of the agent was guaranteed to be executed in this thesis. The agent only had to specify the volume that it wanted to buy or sell and this volume was then bought or sold at the current market price, known to the agent. In a real world application, the agent would need to specify the volume that it intends to buy or sell *and* the price at which it intends to buy or sell. Moreover, the resulting order may or may *not* get executed, depending on the other market participants. Second, the environment of this thesis assumed that the agent's actions do not have any effect on the market. Although only small volumes of electricity were traded by the agent, it is still possible that the agent's actions have an effect on the market price in a real-world application. Third, the representation of the market price in this thesis assumed very fast response times of the agent. Every subsequent 5 executed buy orders were aggregated to 1 timestep in the RL environment. In time periods with high order density, multiple timesteps in the RL environment therefore occurred within a second of real time. Such fast reaction times may not be possible in a real world application. Lastly, it is noteworthy that the data of this thesis was from 2014 to 2016. Today, the market may behave differently. Therefore, more up-to-date order data should be used for training and evaluating a real-world RL application.

The *evaluation* of this thesis, although exhaustive, can be improved to account better for randomness. There are several sources of randomness in the context of reinforcement learning: randomness in the actions taken by the agent (during training and evaluation) and randomness of the environment (since the market is not fully predictable). During training, RL methods generally make use of random sampling to allow for exploration. Furthermore, RL methods with stochastic policies, such as A2C, make use of sampling during evaluation as well. To really account for randomness in training and evaluation, multiple agents should be trained and evaluated with different random seeds although this is very time consuming. In

this thesis, only one agent was trained instead and a bootstrap hypothesis test was used for inference. For a real-world application however, it may be better to take the time consuming approach in order to mitigate risk. In this context, stochastic weight averaging (SWA) might be an efficient alternative to account for volatility during training [117]. For this thesis, first experiments indicated that offline SWA can be used to reduce variation in performance across different seeds without compromising training time. With offline SWA, the final model weights are computed as an average of the weights across the training period. During training, the RL agent makes use of weights *without* averaging, as usual. Offline SWA does not affect the training phase as the weight averaging only occurs retrospectively, after training.

Lastly, limitations of the *model* should be discussed. In this thesis, the RL agent was based on neural networks. Neural networks are the most common model type in reinforcement learning, among others because they allow to fit complex, non-linear functions. However, neural networks have been called *black box* models before for a reason [118]. Interpreting and explaining the outputs of neural networks can be difficult. In the context of trading, however, risk management is very important. Understanding *why* an automatic trading system takes certain actions can be invaluable to manage risk. For example, a thorough understanding may shine light on the weaknesses of the model. This would help to answer questions such as: in which market conditions would the RL agent perform poorly? Knowing about the weaknesses of the RL agent, one can take precautions accordingly. This thesis has shown that successful trading strategies may be simple, as illustrated by the spike trader. Converting successful trading strategies of RL agents to simple rule-based algorithms therefore seems like a realistically possible approach. Therefore, the ideal steps for a real-world application might be the following: first, RL agents should be trained to identify successful trading strategies and second, these RL agents should be converted to interpretable rule-based algorithms.

7.5 Ethical considerations

It is important to consider the work of this thesis from an ethical and societal viewpoint. Hereby, emphasis is put on reinforcement learning for *trading* in particular. Beyond trading, it is noteworthy that reinforcement learning has the potential to be abused for autonomous weapon systems. Although leading research institutions in the field of reinforcement learning, such as OpenAI and Google DeepMind, have pledged against such work [119], this risk remains and it may grow with further advancements in this field.

In regards to trading in general, one may argue that RL agents may be converted into rule-based algorithms. For instance, in this thesis the spike trader was used to represent the strategy of the RL agent to sell at large price spikes. This leads to the question whether *algorithmic trading* is ethical. This question has been discussed by Wellman et al. in [120] for the financial domain. The authors state that algorithmic trading may actually contribute to more efficient markets. In the context of trading electricity for instance, if multiple RL agents attempted to sell at price spikes, this would decrease the size of the spikes due to increased supply provided by the agents. Recall that prices spike up when the demand exceeds the supply and if there is plenty of supply (with several RL agents trying to sell at a spike), the price may not actually spike up. In other words, RL agents may help to reduce volatility. However, Wellman et al. highlight that the effects of algorithmic trading systems on the overall market are still poorly understood and that concerns remain.

While the effects of algorithmic trading in the financial domains seem poorly understood, the effects of algorithmic trading in the electricity domain seem even more poorly understood. Nevertheless, one may argue that trading electricity *does not work without a certain degree of automation*. The interplay of supply and demand are central in the domain of electricity because electricity is a good that is actually consumed (unlike financial products such as stocks). Producers have to adjust to the demand and consumers have to adjust to the supply. Hereby, forecasts play an essential role for both producers and consumers. They use

forecasts to plan ahead in order to be able to meet the actual demand and supply. Recall that the electricity contracts traded in this thesis are contracts for electricity at a certain hour *in the future*. Producers and consumers reduce their financial risk by making good forecasts and by trading accordingly. Hence, a certain degree of automation seems required and appropriate in the domain of electricity trading.

However, while automation seems to make sense for electricity producers and consumers, one may be concerned about agents in the market that neither produce nor consume electricity but only trade electricity for profits. It may also be possible that producers and consumers trade electricity for profits in addition to their main activity of fulfilling their supply and demand. Generally only companies that fulfill certain conditions are allowed to trade electricity on European Energy Exchange markets [121]. It yet remains to be seen how future applications of reinforcement learning used by these companies will affect the market. For instance, it might be possible that RL agents learn to manipulate markets: in the context of trading electricity, agents may try to manipulate the market by *creating* spikes. Clear is that the impact of such applications has to be carefully studied and monitored and that limitations will be required if these applications affect the market negatively.

8 Conclusion

The purpose of the thesis was to create a RL agent that can profitably trade electricity on the continuous hourly intra-day market in Germany. In particular, the RL agent was supposed to maximize *profit per volume traded*. This means that each volume of electricity traded by the agent should result in large incremental profits. To approach this problem, a custom RL environment was implemented, in which the RL agent had to make trading decisions not only about whether to *buy or sell* but also about the *volume* to buy or sell.

Emphasis was put on a) finding a suitable reward function, b) finding a suitable RL method and c) comparing the final RL agent to baseline strategies. In regards to reward functions, a new reward function *vwap delta* was introduced. This reward function was the best performing reward function in terms of the main evaluation metric, profit per volume traded. It significantly outperformed risk-adjusted reward functions such as the Sharpe ratio and the log equity ratio. In regards to RL methods, the actor-critic method A2C was compared with the critic-only method DQN. For both methods, Bayesian optimization was applied to select an optimal combination of features, hyperparameters and reward function. Both agents learned very similar behavioral strategies and there was not sufficient evidence to conclude that they differed significantly in their performance. In regards to baseline strategies, the RL agent was able to significantly outperform a random trader, a forecast trader and a moving average trader. However, its performance was very similar to a custom rule-based algorithm, which was reverse engineered to match the behavioral strategy of the RL agent. This algorithm was called *spike trader* due to its approach to always hold some volume of electricity and to sell when there is sudden price increase. There was not sufficient evidence to conclude that the RL agent and the spike trader differed significantly in their performance.

Importantly, the RL agent in this thesis was trained in a simplified RL environment, which means that the agent cannot be used for real-world trading at this point. Two major simplifications were made. First, the agent was trained on a *static* data set of historic market orders. Therefore, the actions of the agent did not have any effect on the market price since the historic market price time series were given as fixed input to the RL environment. Second, the historic market orders were converted to market prices, which the agent then learned to trade. Thereby, the action space was simplified because instead of placing *bids* for electricity, the agent could simply *buy or sell* electricity at the given market price of each timestep. While bids may or may not be converted to actual trades in the real world, all buy and sell orders by the RL agent in this thesis were guaranteed to be executed.

In spite of these simplifications, this thesis has shown that reinforcement learning does not only have the potential to create profitable trading agents but also that it can be used to identify optimal trading strategies by learning from these agents. In the context of trading, these optimal strategies may then be converted to rule-based algorithms that are interpretable, computationally efficient and simple to deploy in real-world trading applications.

A Appendix

A.1 Static data for RL environments

As mentioned in the introduction section 1.1, RL environments can be represented with a dynamic simulation or with a static data set. In a dynamic simulation, the action a causally affects the next state s'. This causal relationship is illustrated with an arrow pointing from a to s' in the left diagram of figure A.1. *Video game* RL environments are a good example of this scenario: if the RL agent attacks its virtual opponents in Dota 2 (corresponding to action a), then the opponents will defend themselves (reflected in the next state s').

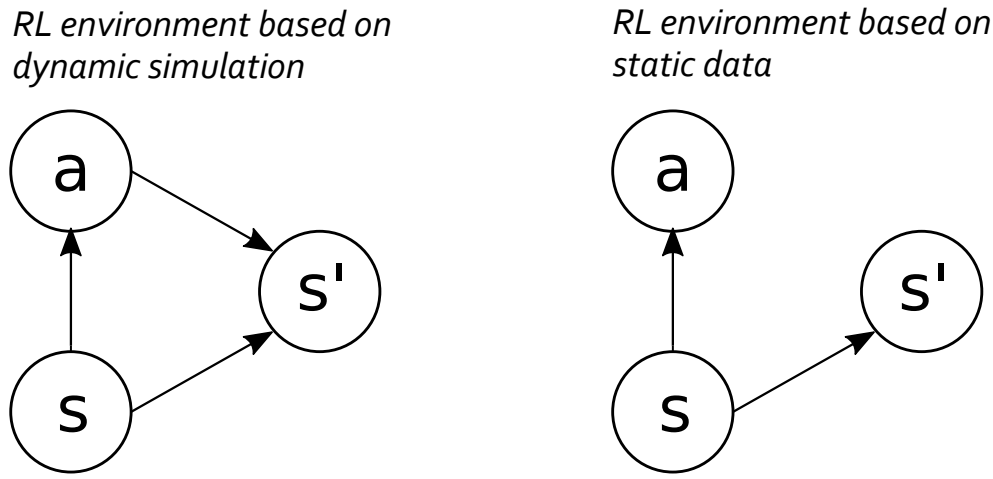


Figure A.1: Causal relationships in an RL environments based on a dynamic simulation (left) and static data (right). Arrows represent causal relationships. The arrows from state s to action a are due to the RL agent's action policy $\pi(a|s)$ described in section 2.1.4. The remaining arrows are due to the probability distribution $p(s', r|s, a)$ described in section 2.1.1. The figure does not display the reward r for simplicity.

If static data is used to represent the states in the RL environment, then the action a does *not* causally affect the next state s'. This is because the next state s' is already given due to the static data. The lack of a causal relationship is illustrated in the right diagram of figure A.1, where there is *no* arrow pointing from a to s'. *Trading* RL environments are a good example of this scenario. For instance, the states of a trading RL environment may be represented with historic market price time series. Obviously, the actions of the agent cannot affect the historic market prices since these are given as fixed input to the RL environment. Essentially, the agent iterates over predetermined states, traversing the historic market data across time.

If the RL environment is represented with static data, the joint probability $p(s', r|s, a)$ from section 2.1.2 can be represented as $p(s'|s)$ and $p(r|s, a)$. The next state s' only depends on the previous state s whereas the reward r depends on both, state s and action a. Importantly, this does not mean that RL is more or less suitable. All RL methods can be applied regardless of whether the RL environment is represented with static data or with a dynamic simulation. In both cases, the RL agent explores different actions a in different states s and thereby learns to take actions that maximize the sum of future discounted rewards, i.e. the return G_t from section 2.1.3. This makes sense in both cases because in both environment types, the reward r depends on action a and state s, reflected by $p(r|s, a)$. This allows the RL agent to learn which action to take in any given state in order to maximize future discounted rewards.

However, not all RL environments may be represented with static data. Static data only makes sense when action a can be assumed not to affect the next state s'. In the context of trading, this is the case because an individual agent can be assumed to have a negligible effect on the overall market, in particular if the agent only trades small volumes relative to the overall market volume. In contrast, using static data does not make sense in the context of video games since it lies in the nature of video games to react to the player's actions.

A.2 RL Algorithms

Algorithm 2: Critic-only pseudocode (1-step Q-learning) [17]

```
//Assume a differentiable action-value function parameterization q̂(s, a, w)
//Assume some step size α > 0 for gradient descent
//Initialize the weights w
for episode ∈ {1, ..., M} do
    for t ∈ {0, ..., T − 1} do
        Take action A_t as argmax_a q̂(S_t, A_t = a, w)
        Get reward R_{t+1} and next state S_{t+1} from the environment
        Update the weights w with gradient descent:
            G_t ← R_{t+1} + γ max_{a'} q̂(S_{t+1}, A_{t+1} = a', w)
            w ← w + α [G_t − q̂(S_t, A_t, w)] ∇q̂(S_t, A_t, w)
    end
end
```

Algorithm 3: Actor-only pseudocode (with Monte Carlo) [17]

```
//Assume a differentiable policy function parameterization π̂(a|s, θ)
//Assume some step size α > 0 for gradient ascent
//Initialize the weights θ
for episode ∈ {1, ..., M} do
    Collect an episode S_0, A_0, R_1, ..., S_{T−1}, A_{T−1}, R_T, S_T according to policy π̂(a|s, θ):
    for t ∈ {0, ..., T − 1} do
        Take action A_t according to the policy π̂(A_t|S_t, θ)
        Get reward R_{t+1} and next state S_{t+1} from the environment
    end
    Update the weights θ with gradient ascent:
    G_T ← 0
    for t ∈ {T − 1, ..., 0} do
        G_t ← R_{t+1} + γG_{t+1}
        θ ← θ + αG_t ∇ ln π̂(A_t|S_t, θ)
    end
end
```

Algorithm 4: Actor-only with baseline pseudocode (with Monte Carlo) [17]

```
//Assume a differentiable state-value function parameterization v̂(s, w)
//Assume a differentiable policy function parameterization π̂(a|s, θ)
//Assume step sizes α^w > 0 and α^θ > 0 for gradient descent and ascent
//Initialize the weights w and θ
for episode ∈ {1, ..., M} do
    Collect an episode S_0, A_0, R_1, ..., S_{T−1}, A_{T−1}, R_T, S_T according to policy π̂(a|s, θ):
    for t ∈ {0, ..., T − 1} do
        Take action A_t according to the policy π̂(A_t|S_t, θ)
        Get reward R_{t+1} and next state S_{t+1} from the environment
    end
    Update the weights w and θ:
    G_T ← 0
    for t ∈ {T − 1, ..., 0} do
        G_t ← R_{t+1} + γG_{t+1}
        δ ← G_t − v̂(S_t, w)
        w ← w + α^w δ ∇v̂(S_t, w)
        θ ← θ + α^θ δ ∇ ln π̂(A_t|S_t, θ)
    end
end
```

Algorithm 5: Actor-critic pseudocode [17]

//Assume a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
//Assume a differentiable policy function parameterization $\hat{\pi}(a|s, \boldsymbol{\theta})$
//Assume step sizes $\alpha^{\mathbf{w}} > 0$ and $\alpha^{\boldsymbol{\theta}} > 0$ for gradient descent and ascent
//Initialize the weights $\mathbf{w}$ and $\boldsymbol{\theta}$
for *episode* $\in \{1, ..., M\}$ **do**
  for $t \in \{0, ..., T-1\}$ **do**
    Take action A_t according to the policy $\hat{\pi}(A_t|S_t, \boldsymbol{\theta})$
    Get reward R_{t+1} and next state S_{t+1} from the environment
    Update the weights $\mathbf{w}$ and $\boldsymbol{\theta}$:
      $\delta \leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta})$
  end
end

Algorithm 6: A2C pseudocode for each agent thread

//Assume a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
//Assume a differentiable policy parameterization $\hat{\pi}(a|s, \boldsymbol{\theta})$
//Assume globally shared weights $\boldsymbol{\theta}$, $\mathbf{w}$ and globally shared counter T
//Assume thread-specific weights $\boldsymbol{\theta}'$, $\mathbf{w}'$ and thread-specific counter t
//Assume globally shared indicator I which is true if all agents finished collecting their batch of experiences for a parameter update and false otherwise
Initialize thread step counter $t \leftarrow 0$
while $T <= T_{max}$ **do**
  // **Step 1: Collect experience**
  Set indicator globally: $I \leftarrow false$
  Synchronize thread-specific weights $\boldsymbol{\theta}' = \boldsymbol{\theta}$ and $\mathbf{w}' = \mathbf{w}$
  Reset gradients: $d\boldsymbol{\theta} \leftarrow 0$ and $d\mathbf{w} \leftarrow 0$
  $t_{start} = t$
  Get state S_t
  while S_t not terminal ***and*** $t - t_{start} <= t_{max}$ **do**
    Perform A_t according to policy $\hat{\pi}(A_t|S_t, \boldsymbol{\theta}')$
    Receive reward R_{t+1} and new state S_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  end
  $G_t = \begin{cases} 0, & \text{if } S_t \text{ is terminal} \\ \hat{v}(S_t, \mathbf{w}')), & \text{if } S_t \text{ is not terminal} \end{cases}$
  for $i \in \{t-1, ..., t_{start}\}$ **do**
    $G_i \leftarrow R_{i+1} + \gamma G_{i+1}$
    $\delta \leftarrow G_i - \hat{v}(S_i, \mathbf{w}')$
    Accumulate gradients wrt $\mathbf{w}'$: $d\mathbf{w} \leftarrow d\mathbf{w} + \delta \nabla \hat{v}(S_t, \mathbf{w}')$
    Accumulate gradients wrt $\boldsymbol{\theta}'$: $d\boldsymbol{\theta} \leftarrow d\boldsymbol{\theta} + \delta \nabla \ln \hat{\pi}(A_i|S_i, \boldsymbol{\theta}')$
  end
  // **Step 2: Update weights**
  Wait until indicator is globally set to true: $I \leftarrow true$
  Send the gradients $d\boldsymbol{\theta}$ and $d\mathbf{w}$ to global worker, where the gradients of all agents are used together to perform synchronous update of $\boldsymbol{\theta}$ and $\mathbf{w}$
end

A.3 Feature Distributions

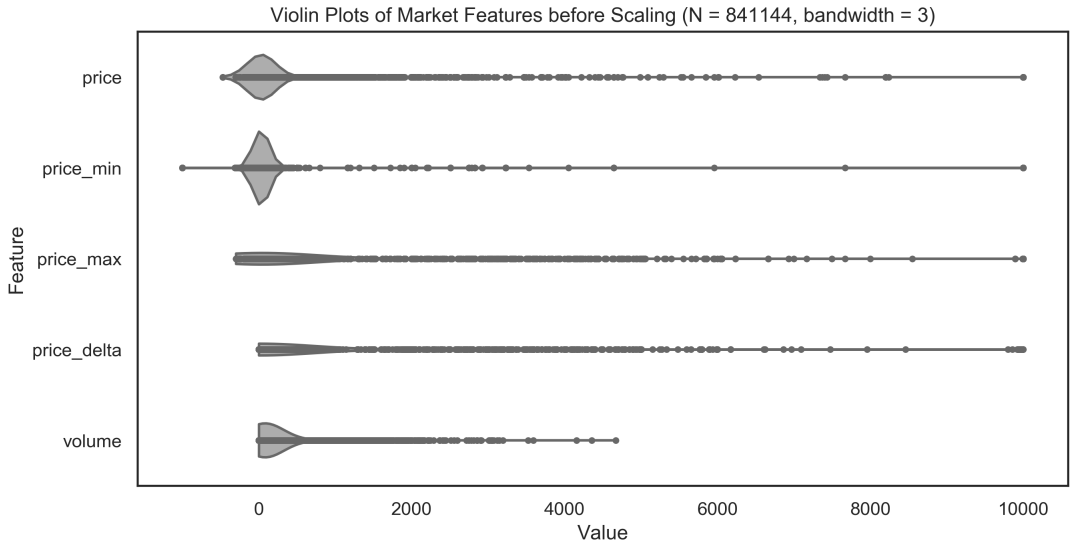


Figure A.2: Violin plots of market features before scaling. N = 841144 is the total number of timesteps across all episodes as described in table 4.2.

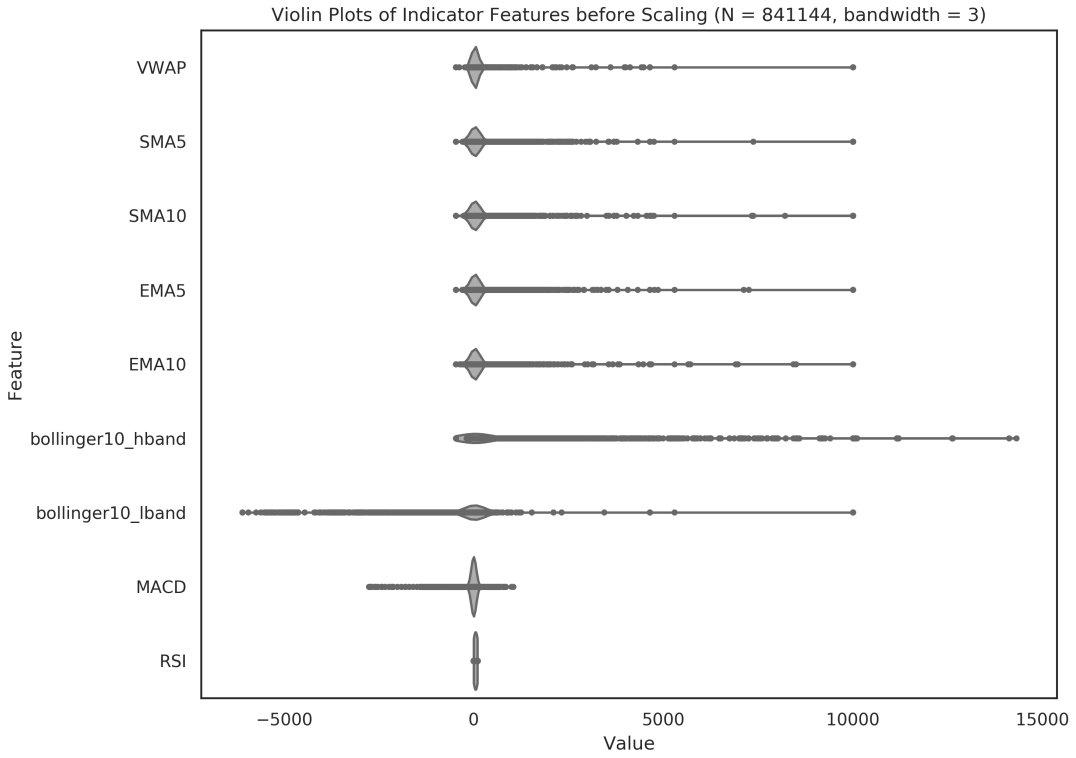


Figure A.3: Violin plots of indicator features before scaling. N = 841144 is the total number of timesteps across all episodes as described in table 4.2.

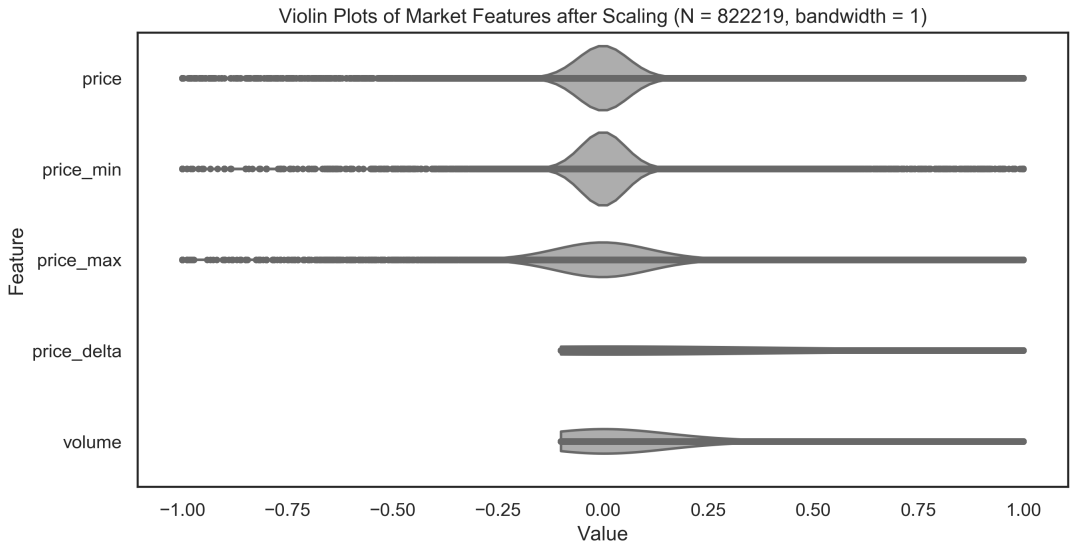


Figure A.4: Violin plots of market features after scaling. N = 822219 is the total number of timesteps after removing the 1st timestep of each episode to compute the percentage changes.

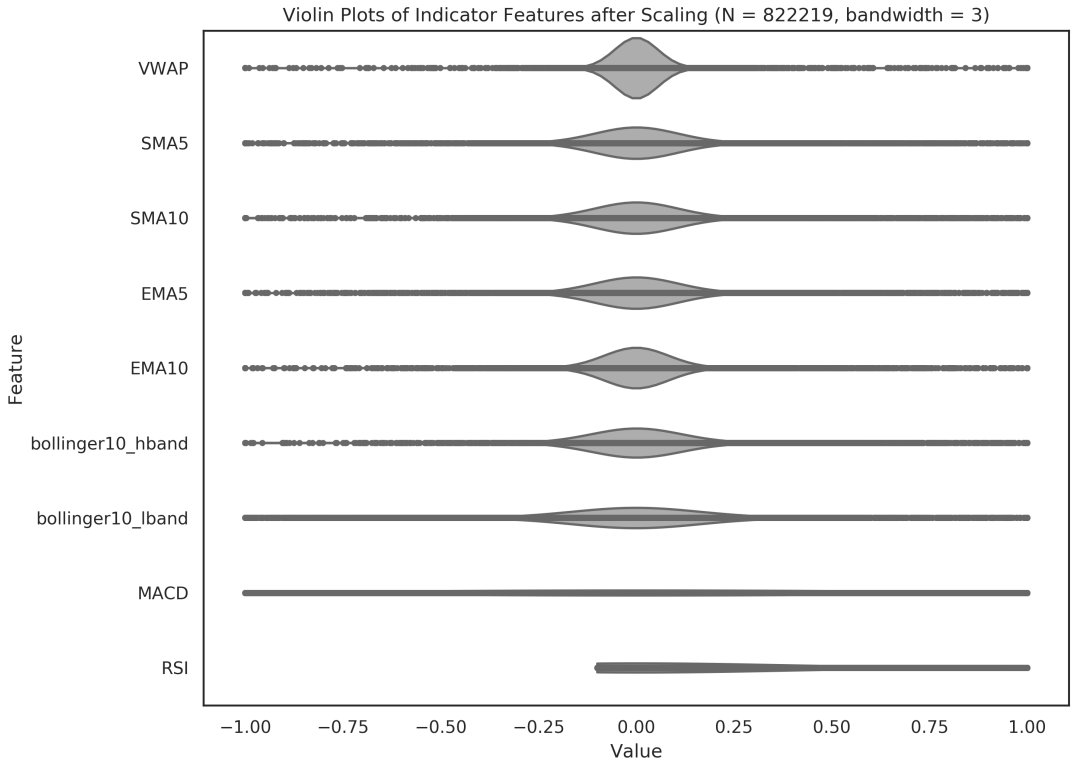


Figure A.5: Violin plots of indicator features after scaling. N = 822219 is the total number of timesteps after removing the 1st timestep of each episode to compute the percentage changes.

A.4 Q-Q Plots

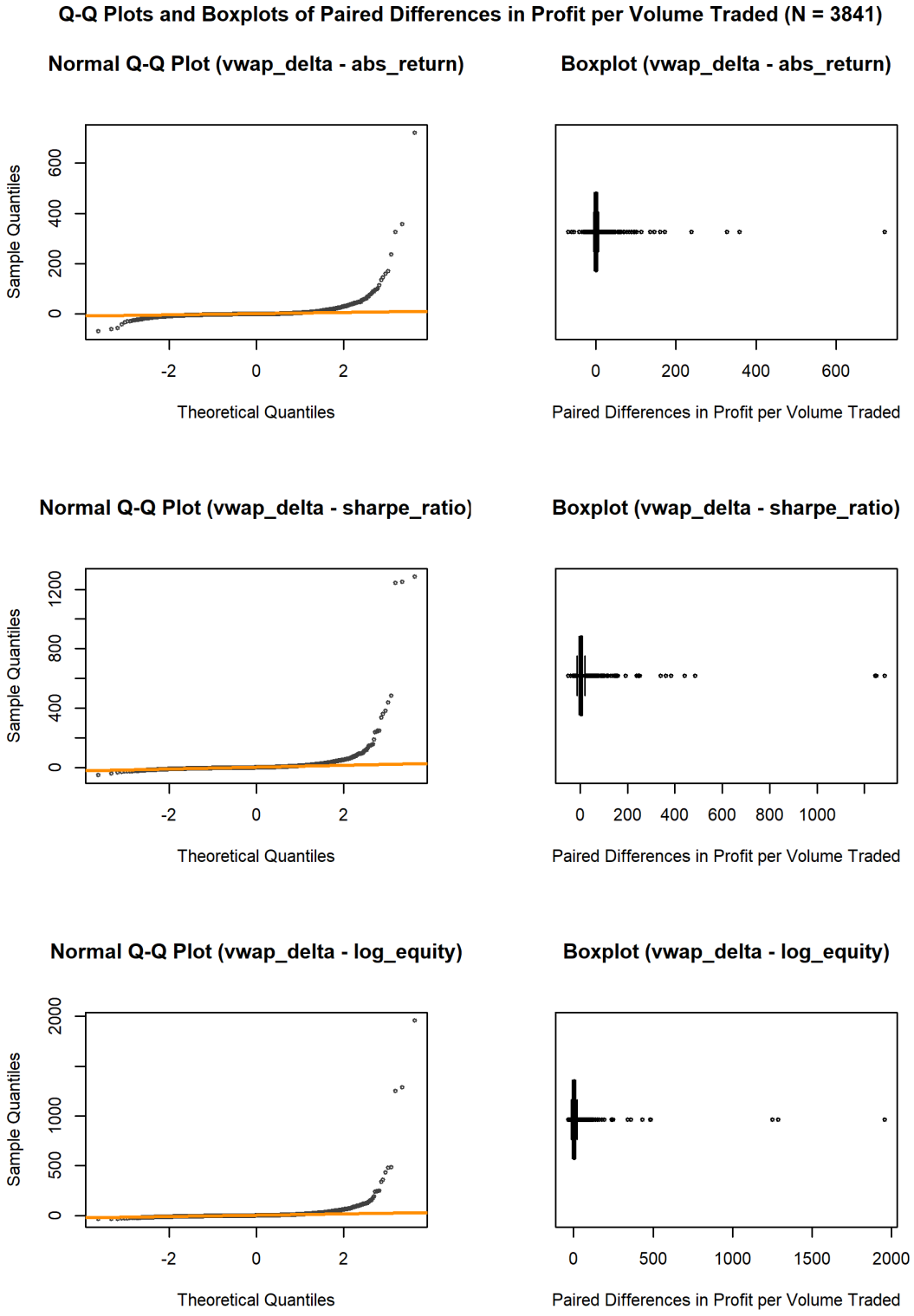


Figure A.6: Question 1 - Q-Q Plots and Boxplots of Paired Differences in Profit per Volume Traded

A.5 Training Metrics

Various training metrics were tracked to evaluate the the learning progress during training. This was mainly done to check whether the agents did learn something during the training period. The metrics used to evaluate A2C are described in section A.5.1 and the training metrics used to evaluate DQN are described in section A.5.2 below. All metrics are computed per timestep and not per episode.

A.5.1 A2C

In order to evaluate the A2C learning progress during training, the following four learning metrics were plotted: discounted rewards, entropy loss, state-value loss, policy loss. The paper about A3C [4], the text book by Sutton and Barto [17] and the source code of A2C [108] were used as reference for formulating these learning metrics.

To illustrate the expected behaviors of the four learning metrics across the training period, the agents were trained on a single episode, i.e. on one single contract. This scenario is much simpler than the actual scenario of this thesis, where numerous different contracts are traded. Due to this simple scenario, the learning curves are very clean, representing a good reference for how the loss functions should look like under ideal, simplified circumstances. The visualizations of these exemplary, ideal learning curves can be found in figure A.8.

Metric 1: Discounted rewards
Discounted rewards are large when the observed reward values in the current batch of (s, a, r) tuples collected by the agent are large. Since the rewards are expected to increase over time with the agent learning which actions to take, the discounted rewards are also expected to increase over time across the training period. Note how the equation below resembles the return G_t in the episodic case from equation 2.4. The difference is that here not the rewards until time T, the last timestep of the episode, are used but only the rewards until t_{end}, the last timestep of the current batch of (s, a, r) tuples collected by the agent.

$$\mathcal{L}_t^{(1)} = \sum_{k=0}^{t_{end}-t-1} \gamma^k R_{t+k+1} \tag{A.1}$$

Metric 2: Entropy loss
The entropy loss is largest when the agent takes all actions with equal probability and smallest (taking the value of 0) when the agent takes one action with probability 1 and all other actions with probability 0. At the beginning of the learning phase, entropy is generally large because the agent does not have any reason to prefer any action over other actions, not having learned anything yet. However, over the course of the training period, entropy is expected to decrease with the agent having learned to prefer certain actions over others in certain states. Furthermore, the entropy loss is generally expected not to reach the value 0 because this could mean that the agent always takes the exact same action, which is likely not optimal.

$$\mathcal{L}_t^{(2)} = \sum_{a \in \mathcal{A}(S_t)} -\hat{\pi}(A_t|S_t, \boldsymbol{\theta}) \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta}), \tag{A.2}$$

where the sum is over all possible actions a that can be taken in state S_t.

Metric 3: State-value loss
The state-value loss is small when the observed rewards correspond to a return G_t that matches with the expected return predicted by the state-value function $\hat{v}(S_t, \mathbf{w})$. In contrast, the state-value function loss is large when unexpected rewards are observed. For instance, unexpected rewards may occur when the agent sells or buys electricity when the price spikes

up. Overall, the state-value loss is expected to decrease, assuming that the state-value function gets better at predicting the future expected return. In practice, one may observe that the state-value loss first increases (in the time period when the policy keeps changing) and then decreases (when the policy does not change anymore).

$$\mathcal{L}_t^{(3)} = \left(G_t - \hat{v}(S_t, \mathbf{w})\right)^2, \tag{A.3}$$

where G_t is computed as a combination of the discounted rewards and the future expected return from thereon based on the state-value function: $G_t = \mathcal{L}_t^{(1)} + \gamma^{t_{end}-t}\hat{v}(S_{t_{end}}, \mathbf{w})$.

Metric 4: Policy loss
The policy loss is large when the policy receives a major update and small when the policy does not change. For instance, a major update can be caused by a large state-value function loss. The absolute value of the policy loss is expected to decrease across the training period.

$$\mathcal{L}_t^{(4)} = \left(G_t - \hat{v}(S_t, \mathbf{w})\right) \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta}) \tag{A.4}$$

The policy loss is based on the policy gradient theorem. However, instead of the gradient, the corresponding loss is expressed. In the implementation of A2C, TensorFlow is used which requires a loss function instead of a gradient function. For reference, the gradient according to the policy gradient theorem is formulated as $\left(G_t - \hat{v}(S_t, \mathbf{w})\right)\nabla \ln \hat{\pi}(A_t|S_t, \boldsymbol{\theta})$.

A.5.2 DQN

In order to evaluate the DQN learning progress during training, the following two learning metrics were plotted: rewards and action-value loss. The source code of DQN [108] and the paper about prioritized experience replay [25] were used as main reference for formulating these learning metrics below.

Metric 1: Rewards
Rewards are simply defined as the rewards R_{t+1} returned by the environment in response to the corresponding action A_t. Rewards are expected to increase across the training period.

$$\mathcal{L}_t^{(1)} = R_{t+1} \tag{A.5}$$

Metric 2: Action-value loss
This is the loss used for updating the Q-network weights. Note that instead of using the squared error loss function, the Huber loss function is used because it is more robust with respect to outliers (see [122] for reference). Similar to the state-value loss of A2C, the action-value loss of DQN reflects how well the model can predict the expected future return $q_\pi(s, a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$.

Note that the return would be 0 if no trades are taken by the agent (and hence the action-value loss would also be zero). Placing no trades would not be optimal with respect to the return G_t however. In contrast, the return G_t would be large if the agent does place trades, in particular if it sells when prices spike up. In this particular case however, the expected future return is *more difficult* to predict and consequently, the losses may *go up*. Therefore, the action-value loss should be interpreted only along with the rewards from A.5 since the main purpose of training is that the rewards increase over time.

$$\mathcal{L}_t^{(2)} = \begin{cases} \frac{1}{2}a^2, & \text{if } |a| \leqslant 1 \\ |a| - \frac{1}{2}, & \text{otherwise.} \end{cases} \tag{A.6}$$

Here, a is defined as the weighted TD-error previously explained in section 2.5.2 about experience replay: $a = w_t\delta_t$, where w_t is the weight and δ_t is the TD-error (computed with the target network as in 2.33).

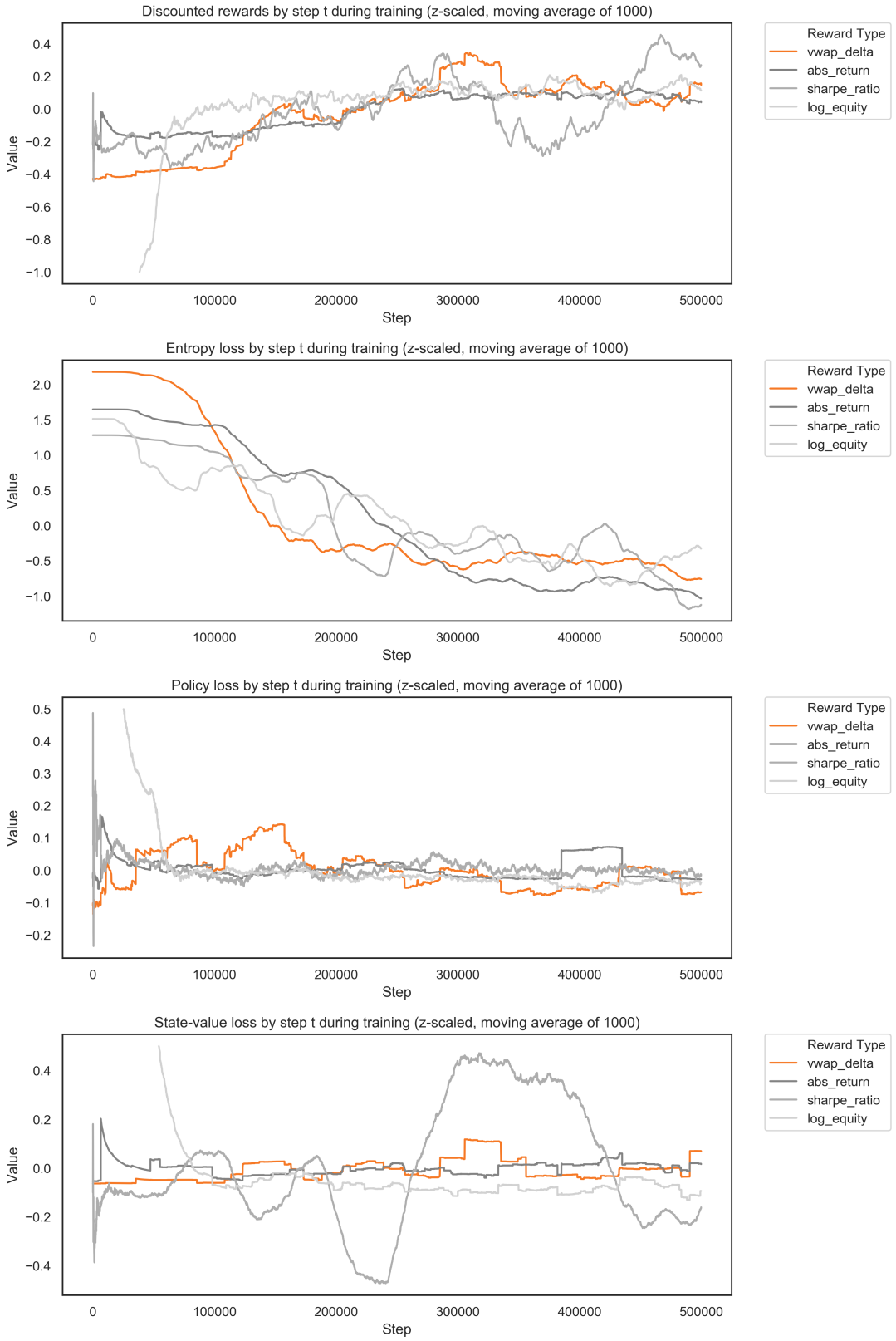


Figure A.7: Question 1 - Learning curves from training phase (training data)

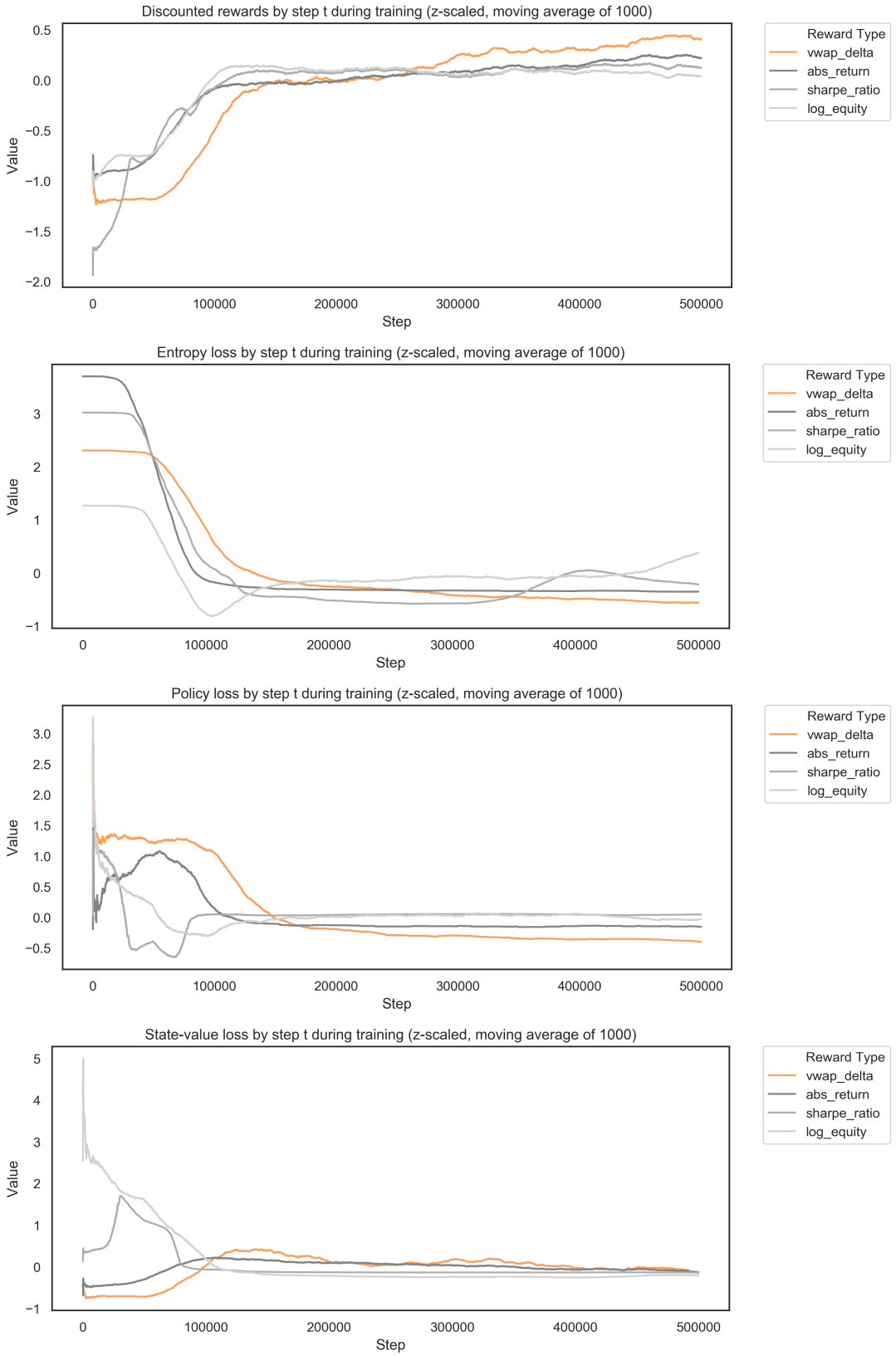


Figure A.8: Question 1 - Learning curves from training with 1 single contract (training data)

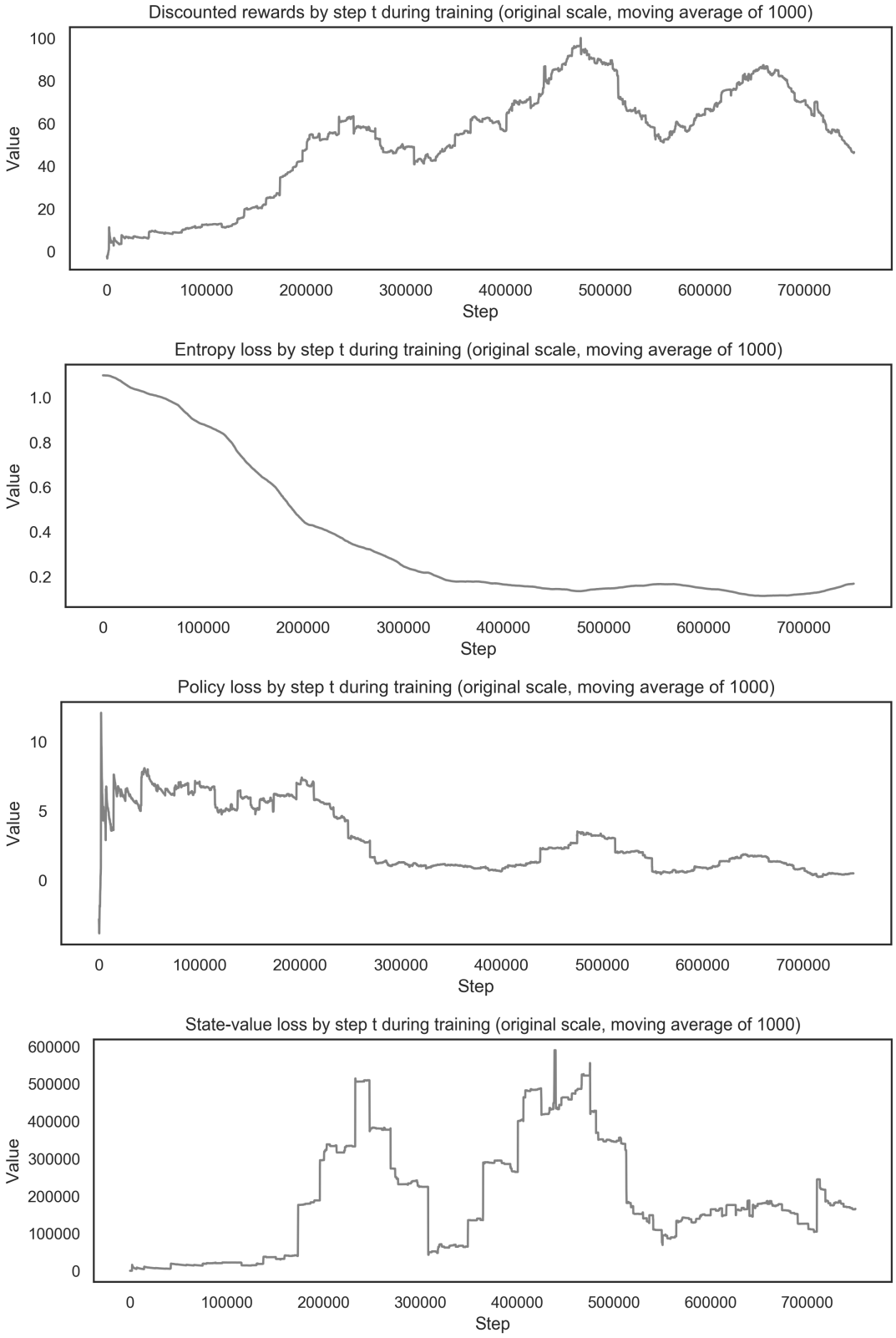


Figure A.9: Question 2 - A2C learning curves from training phase (training data)

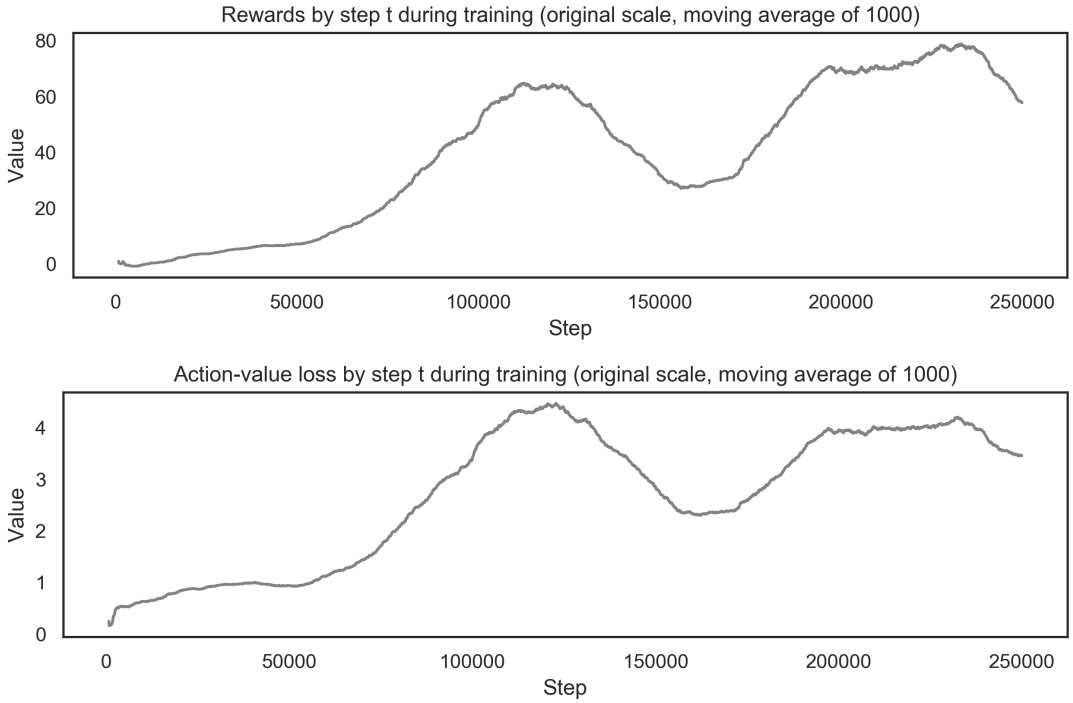


Figure A.10: Question 2 - DQN learning curves from training phase with 250K steps for the *final model* (training data)

A.6 Episode Categorization

The test episodes were categorized into groups of episodes with different price patterns (spikes, uptrend, downtrend, no trend) in order to gain a deeper understanding of the behavioral strategies of the RL agents. The following rules were used to categorize the episodes. These rules were chosen by the author because they seemed to categorize episodes appropriately based on visual inspection: figure A.11 in the appendix shows 10 randomly sampled test episodes for each of the four categories (where all test episodes were categorized according to the rules below).

1. Episodes were categorized as *episodes with spike* if they contained at least 1 sequence of 3 timesteps $t-1, t, t+1$
   - where the price increased by $\geqslant 25\%$ from $t-1$ to t, and
   - where the price decreased by $\leqslant -25\%$ from t to $t+1$, and
   - where the absolute value of the price at $t-1$ was larger than 1.
2. Episodes were categorized as *episodes with uptrend*
   - if they were not categorized as episodes with spike, and
   - if the last price of the episode is larger than the first price, i.e. $P_0 < P_T$, and
   - if in more than 2/3 of the timesteps t of the episode, the price is *larger* than the simple moving average price based on 10 timesteps, i.e. $P_t > \frac{P_{t-9}+...+P_{t-1}+P_t}{10}$
3. Episodes were categorized as *episodes with downtrend*
   - if they were not categorized as episodes with spike, and
   - if the last price of the episode is smaller than the first price, i.e. $P_0 > P_T$, and
   - if in more than 2/3 of the timesteps t of the episode, the price is *smaller* than the simple moving average price based on 10 timesteps, i.e. $P_t < \frac{P_{t-9}+...+P_{t-1}+P_t}{10}$
4. Episodes were categorized as *episodes with no trend*
   - if they were not categorized as episodes with spike, uptrend or downtrend.

A.7 Example Episodes

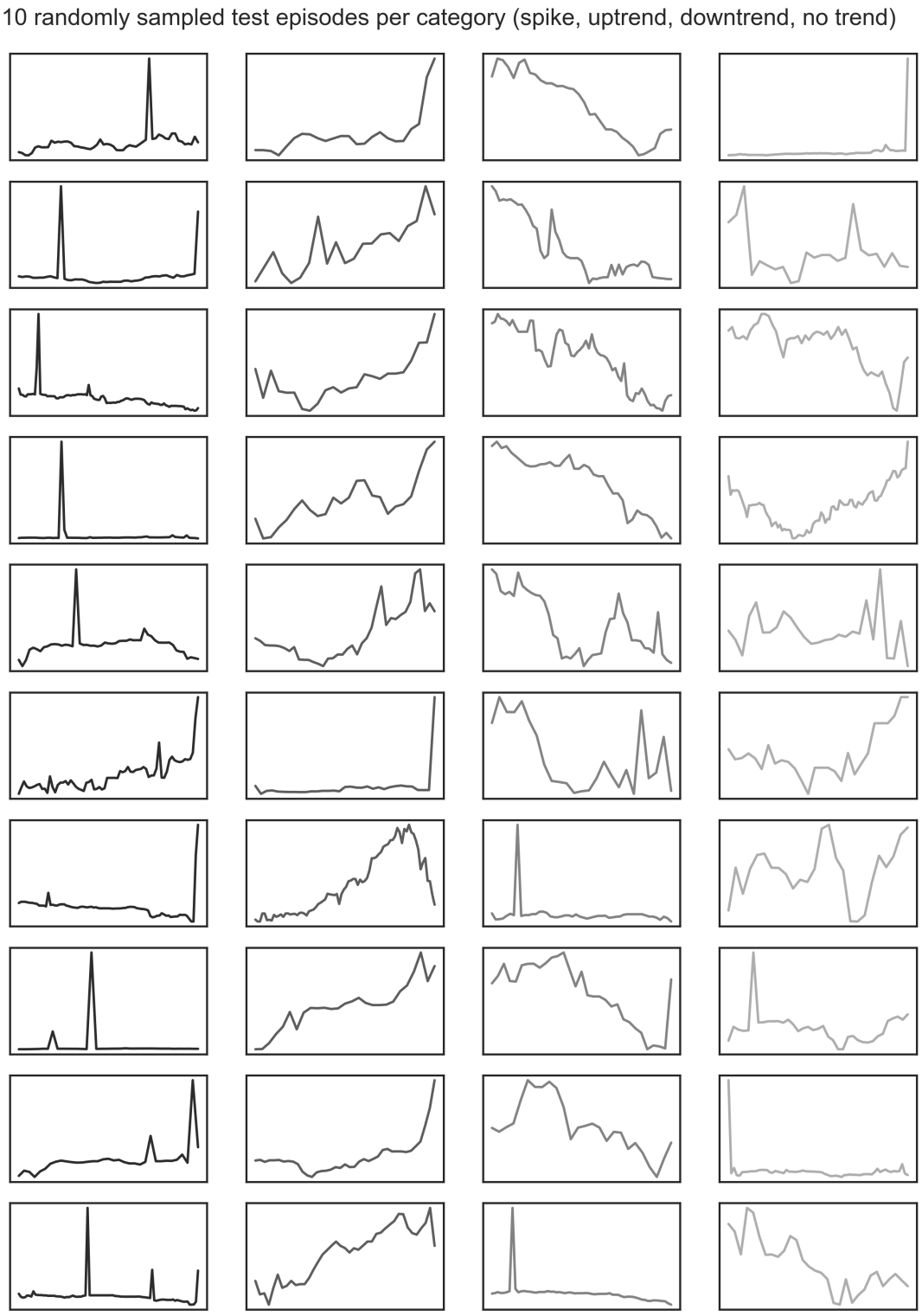


Figure A.11: 10 randomly sampled test episodes for each episode type: spike, uptrend, downtrend and no trend (from the leftmost to the rightmost column respectively)

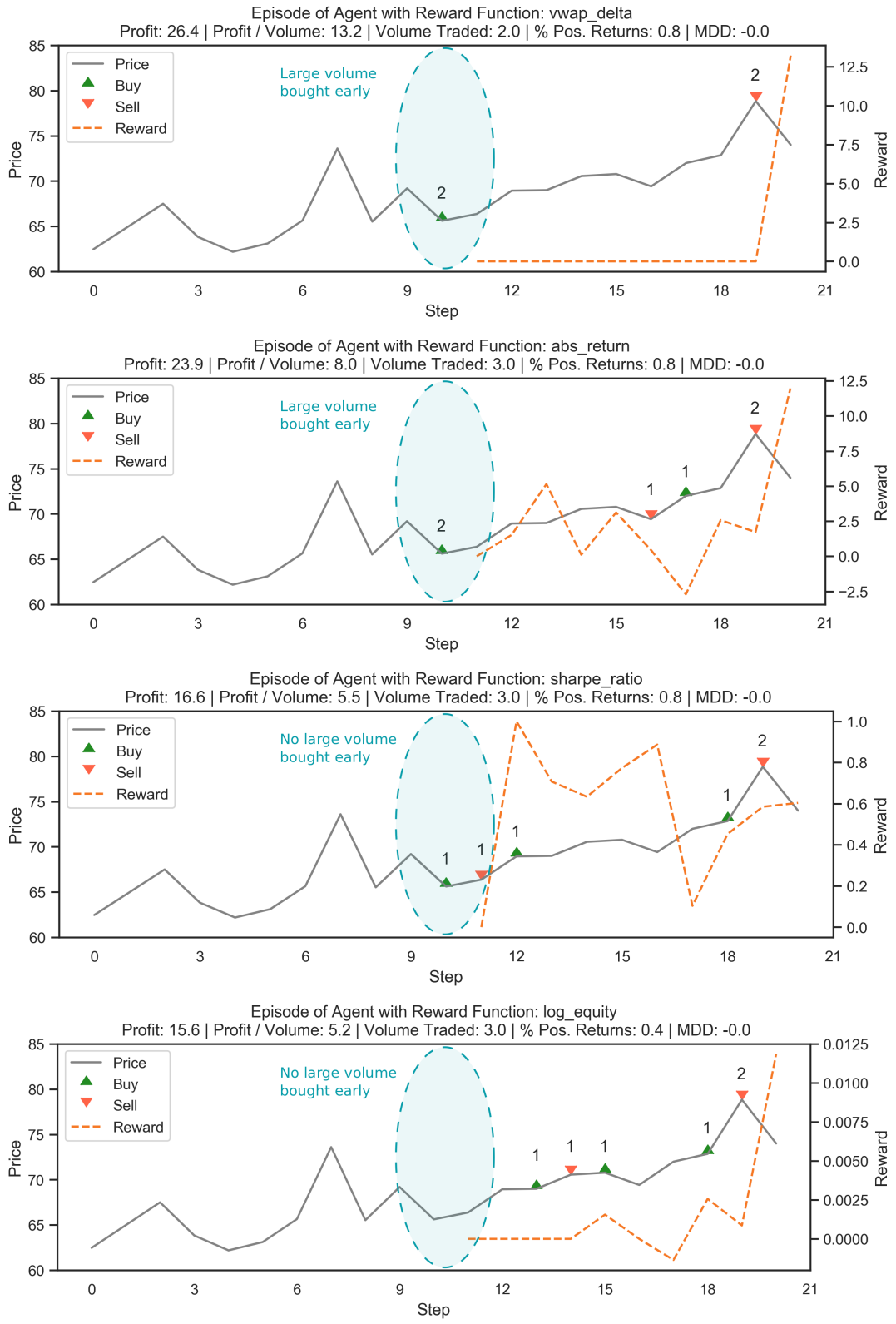


Figure A.12: Question 1 - Example of episode with *increasing* trend for all 4 reward functions (test data). The figure illustrates that the agents trained with profit based reward functions (vwap delta and absolute return) tend to buy a large volume early (allowing for larger profits in episodes with uptrend), whereas the the agents trained with risk-adjusted reward functions (Sharpe ratio and log equity) tend not to buy a large volume early.

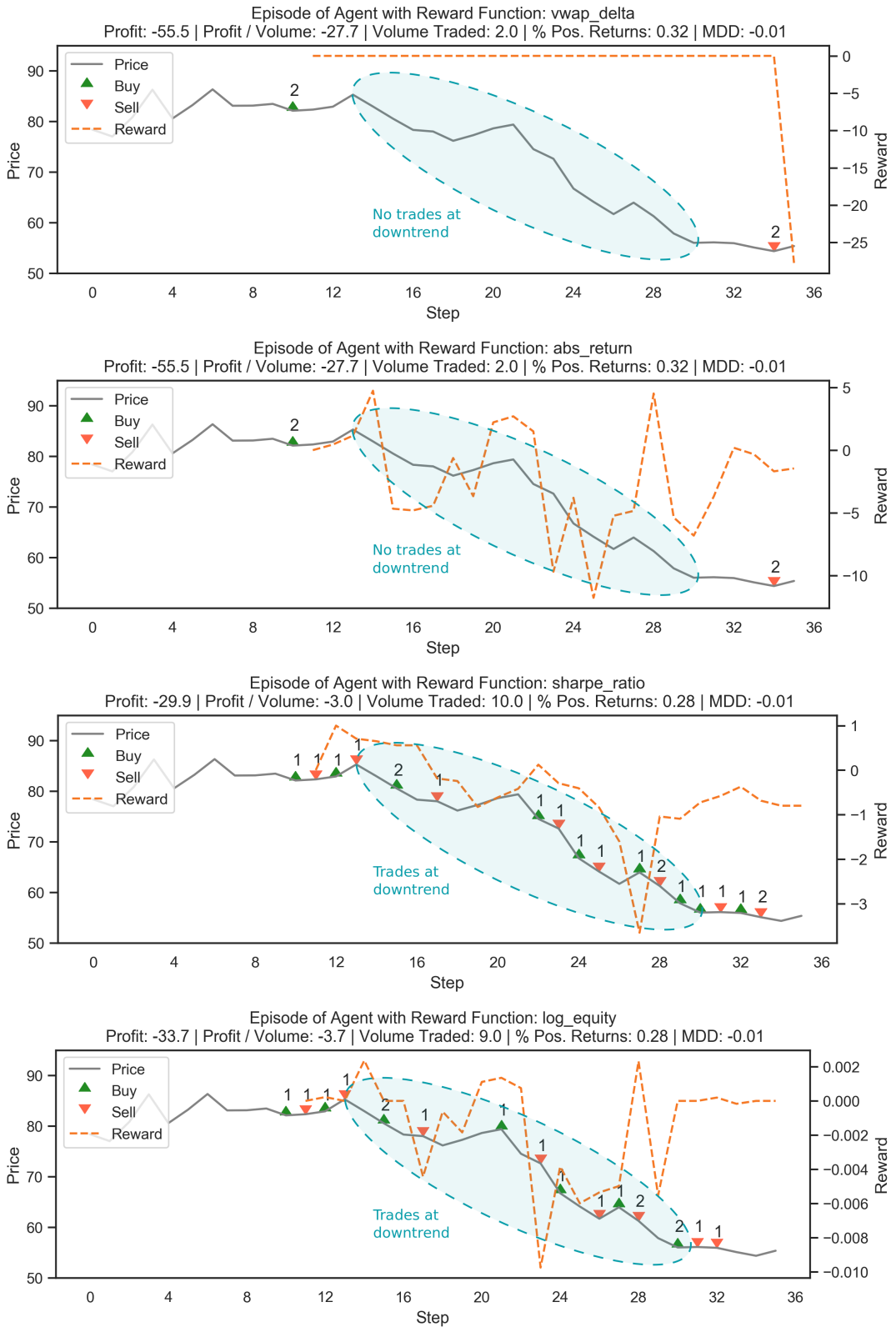


Figure A.13: Question 1 - Example of episode with *decreasing* trend for all 4 reward functions (test data). The figure illustrates that the agents trained with profit based reward functions (vwap delta and absolute return) tend to hold their volume in case of a downtrend (resulting in larger losses), whereas the the agents trained with risk-adjusted reward functions (Sharpe ratio and log equity) keep trading during the downtrend.

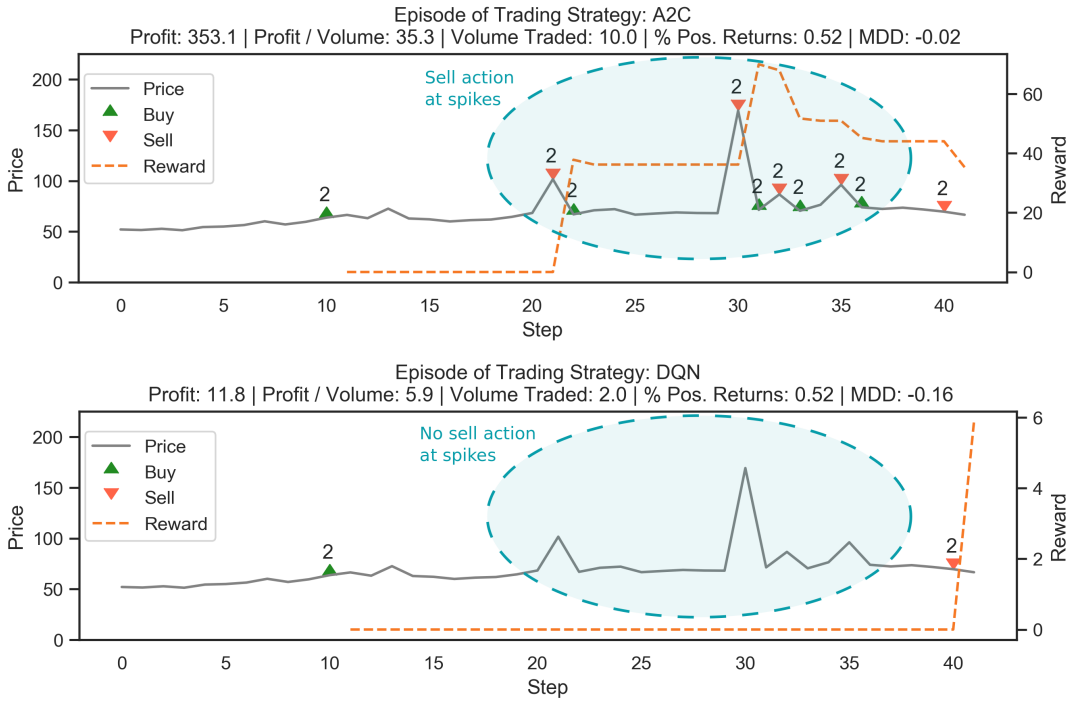


Figure A.14: Question 2 - Example of episode with *spikes* for RL agents trained with A2C and DQN (test data). This figure illustrates that in *some* cases, DQN does not sell at the spikes when A2C does. This episode can be considered an outlier because here, DQN does not sell at any spikes at all.

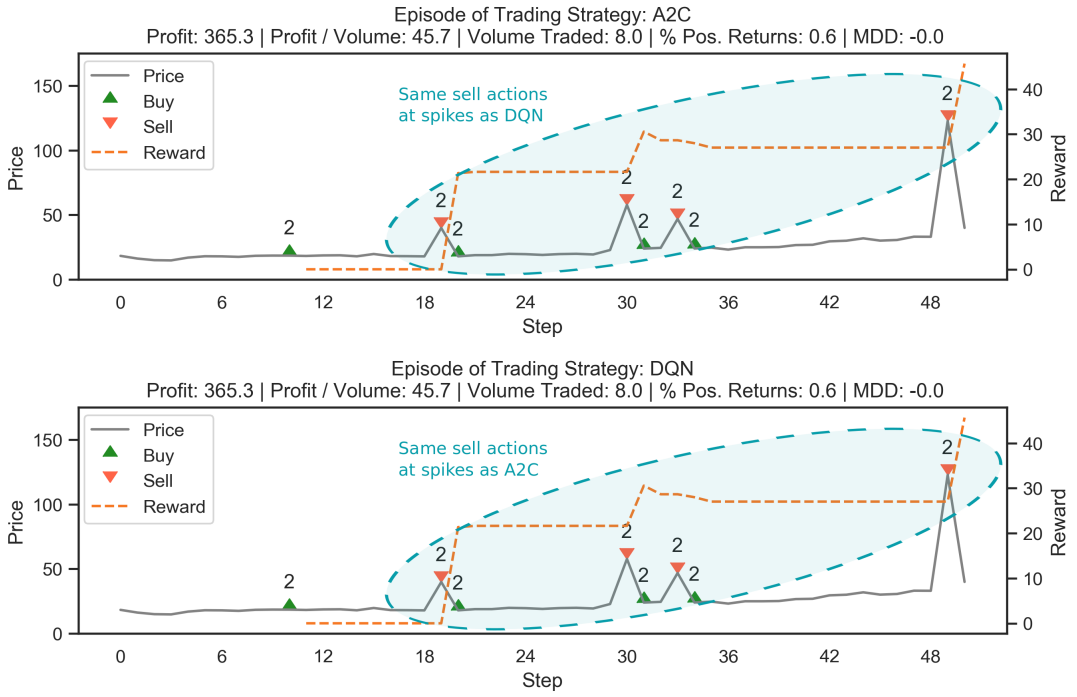


Figure A.15: Question 2 - Additional example of episode with *spikes* for RL agents trained with A2C and DQN (test data). This figure illustrates that in *most* cases both agents, trained with A2C and DQN, sell at the spikes and even take the exact same actions doing so.

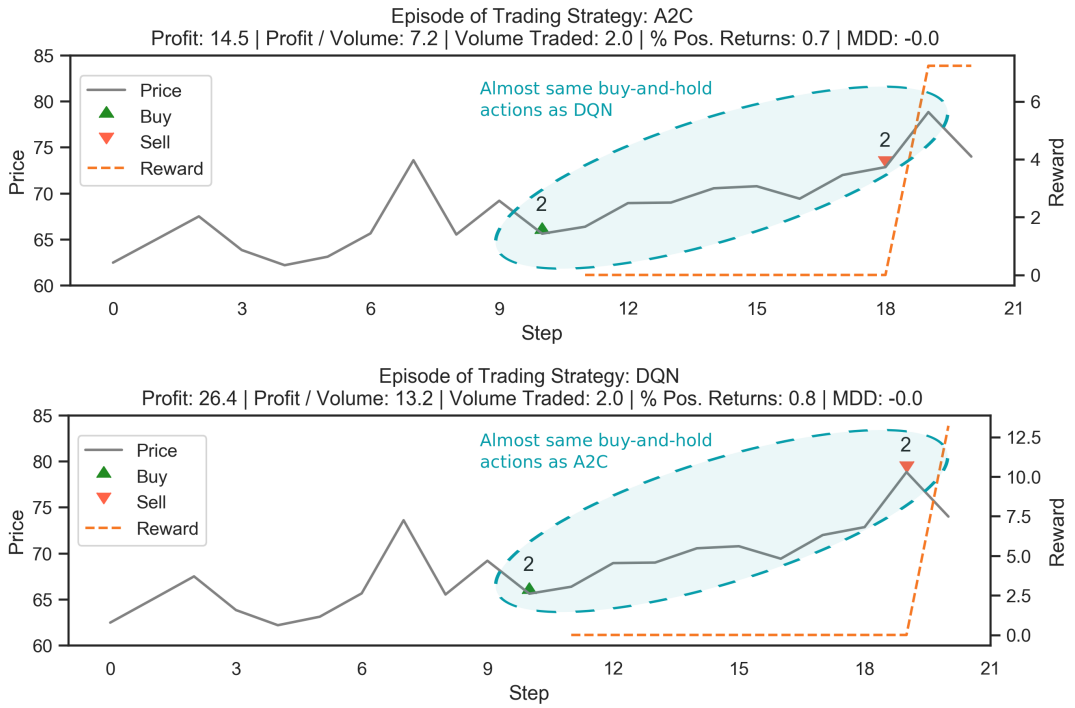


Figure A.16: Question 2 - Example of episode with *increasing* trend for RL agents trained with A2C and DQN (test data). This figure illustrates that the actions of both agents, trained with A2C and DQN, are generally very similar for episodes with uptrend.

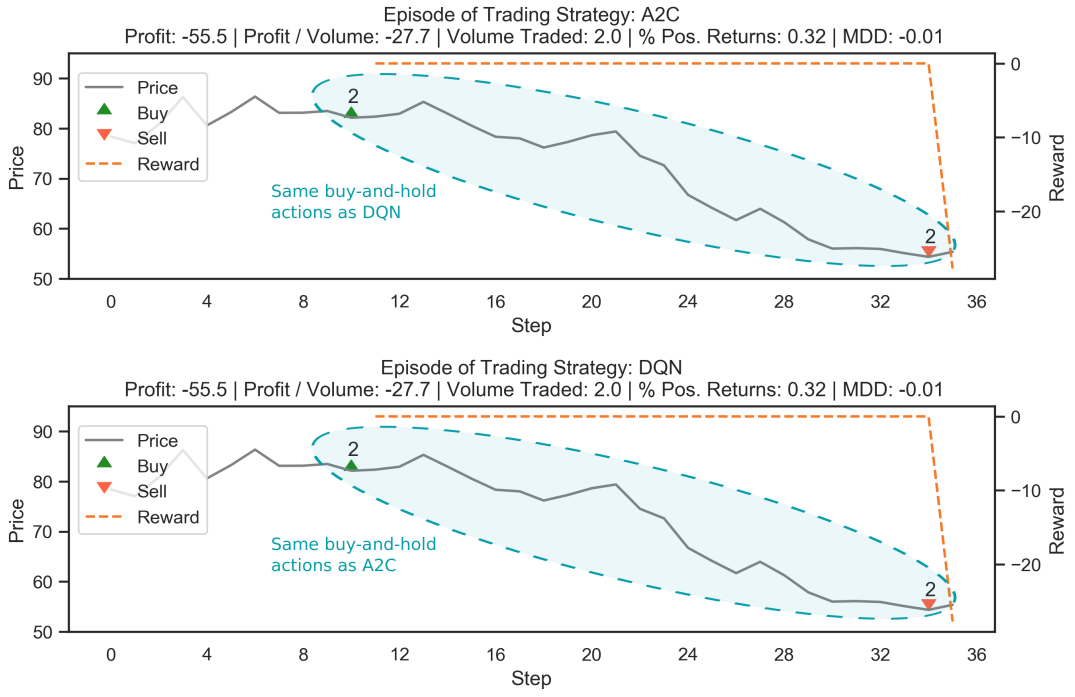


Figure A.17: Question 2 - Example of episode with *decreasing* trend for RL agents trained with A2C and DQN (test data). This figure illustrates that the actions of both agents, trained with A2C and DQN, are generally very similar for episodes with downtrend.

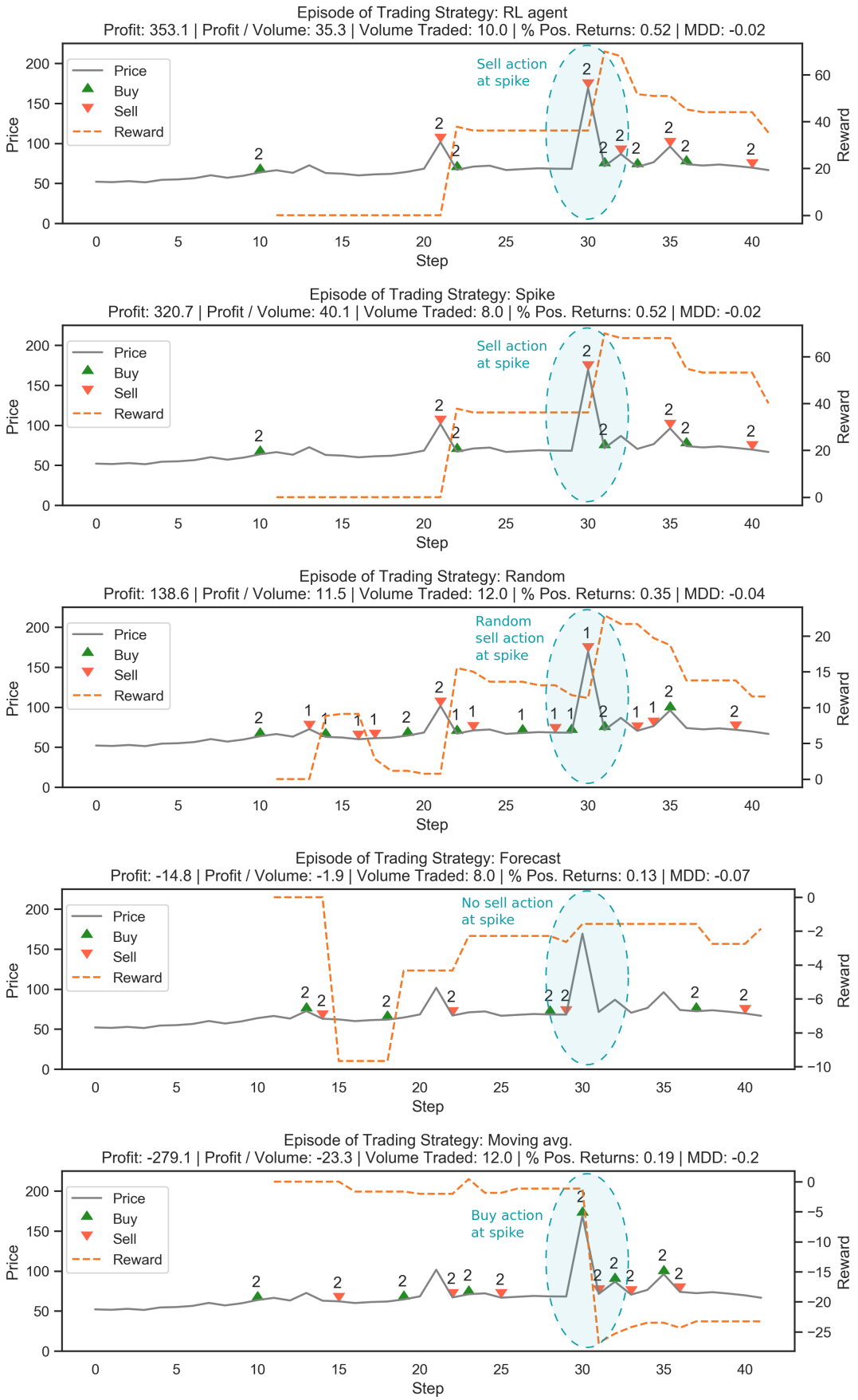


Figure A.18: Question 3 - Example of episode with *spikes* for RL agent and 4 baseline strategies (test data). The figure illustrates that the RL agent and spike trader sell at spikes, the random trader may sell at spikes by chance, the forecast trader tends not to sell at spikes and the moving average trader tends to buy at spikes.

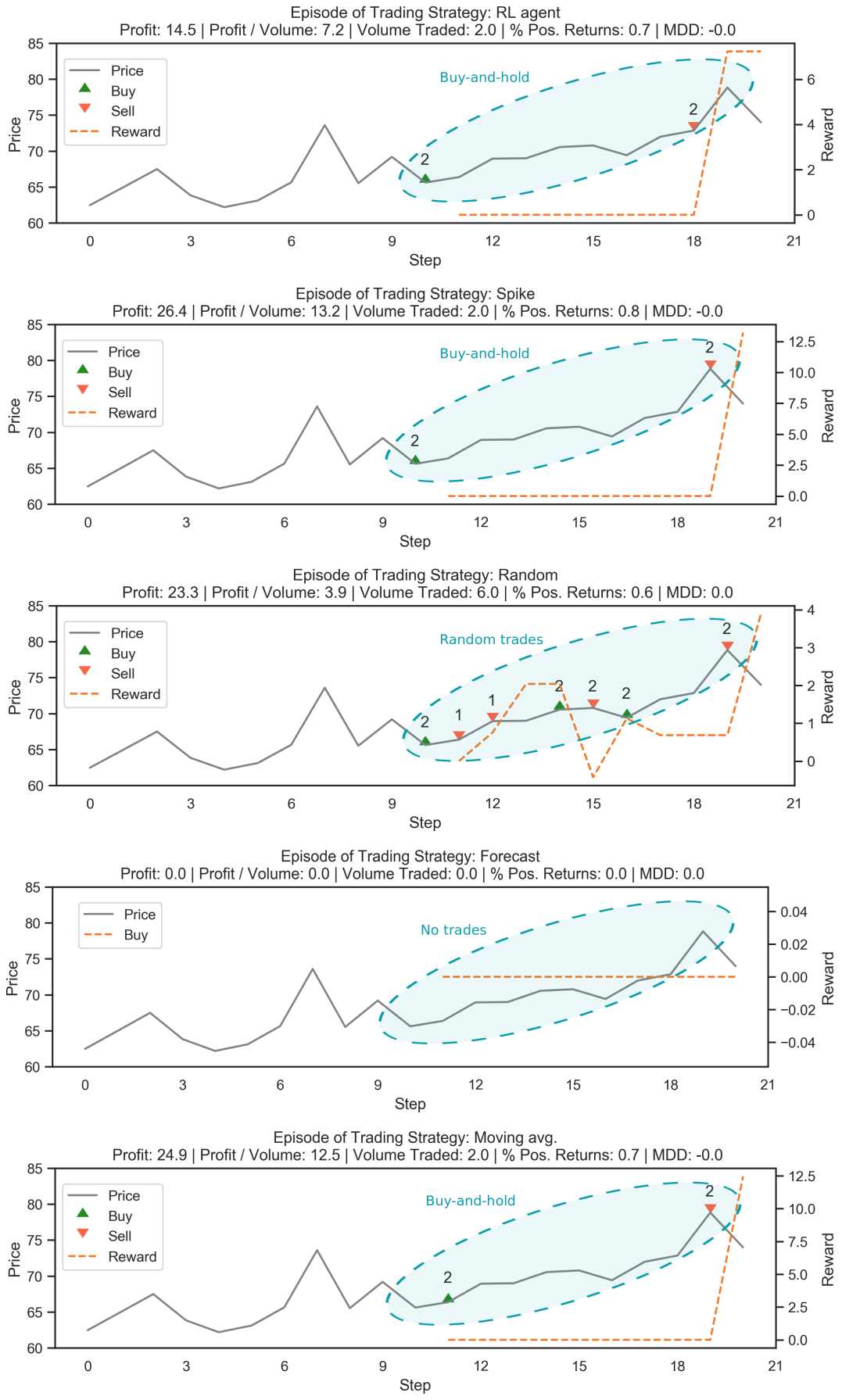


Figure A.19: Question 3 - Example of episode with *increasing* trend for RL agent and 4 baseline strategies (test data). The figure illustrates that RL agent, spike trader and moving avg. trader use the buy-and-hold strategy for episodes with uptrend. The random trader places random trades and the forecast trader may not trade at all.

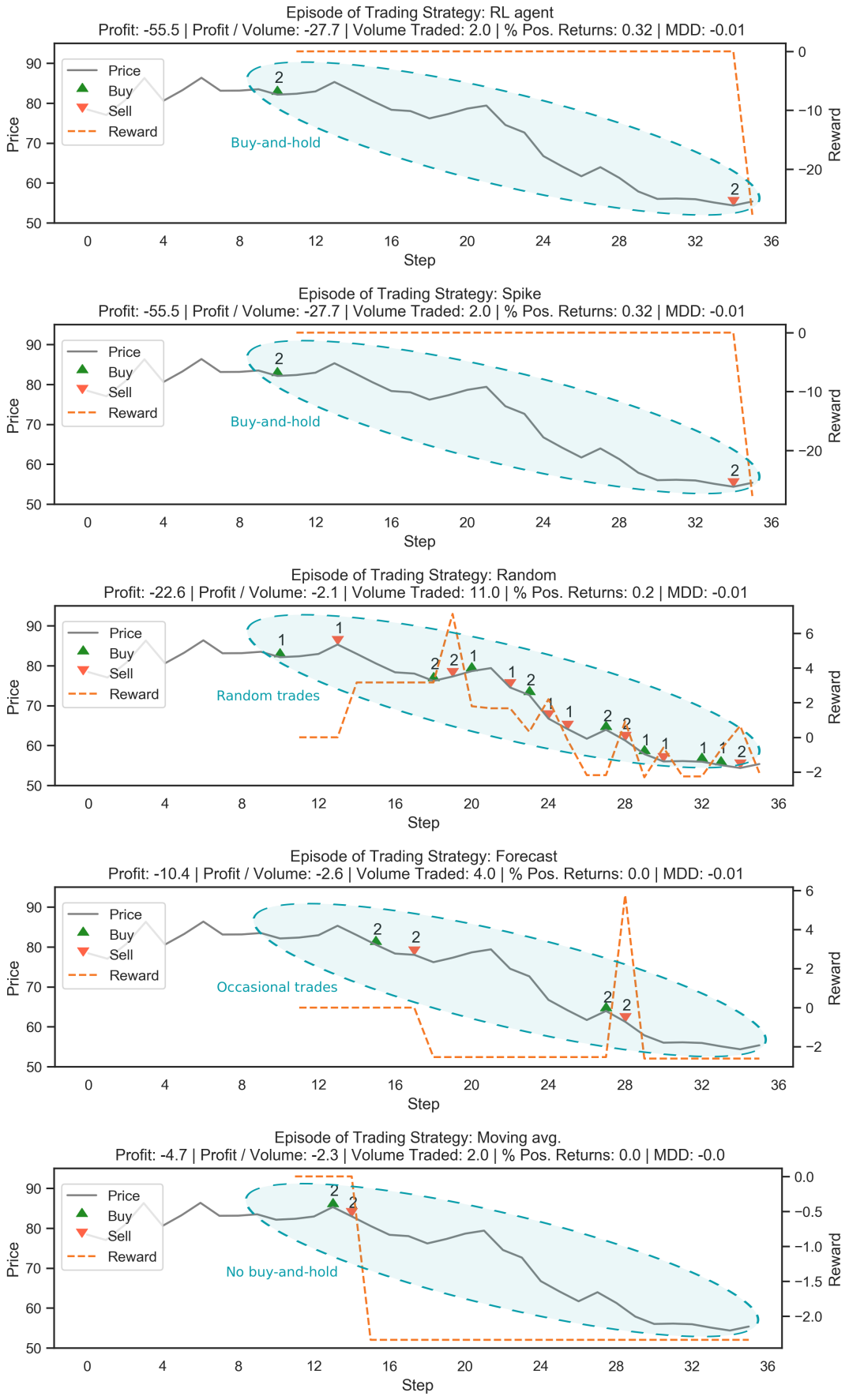


Figure A.20: Question 3 - Example of episode with *decreasing* trend for RL agent and 4 baseline strategies (test data). The figure illustrates that RL agent and spike trader use a buy-and-hold strategy for episodes with downtrend. The random trader places trades randomly and the forecast trader occasionally based on forecasts. The moving avg. trader may buy some volume at price increases but it never holds the volume long.

A.8 Results by Episode Type

Table A.1: Question 1 - Profit per volume traded by episode and reward type (test data)

Episode Type	Reward Type	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
Spike	vwap_delta	1402	23.53	80.91	-57.66	4.58	10.62	23.42	1957.21
	abs_return	1402	16.11	73.11	-50.54	2.88	6.90	15.24	1957.21
	sharpe_ratio	1402	3.93	23.52	-48.29	-0.12	1.20	4.58	712.21
	log_equity	1402	1.10	12.55	-77.59	-0.82	0.21	1.99	325.17
Uptrend	vwap_delta	374	5.92	8.09	-4.48	2.18	3.99	7.67	111.42
	abs_return	374	4.30	5.51	-3.65	1.39	3.05	5.38	62.76
	sharpe_ratio	374	1.78	2.35	-3.59	0.52	1.27	2.36	18.83
	log_equity	374	1.09	1.64	-3.87	0.28	0.72	1.53	11.96
Downtrend	vwap_delta	478	-3.85	5.57	-27.74	-6.08	-3.18	-0.72	22.10
	abs_return	478	-2.74	4.89	-27.74	-4.17	-1.73	-0.25	23.67
	sharpe_ratio	478	-0.89	1.86	-8.63	-1.45	-0.58	-0.14	24.78
	log_equity	478	-0.69	4.29	-88.71	-0.96	-0.44	-0.05	23.07
No trend	vwap_delta	1227	1.45	6.77	-39.75	-1.02	0.75	3.13	120.88
	abs_return	1227	1.04	5.17	-19.63	-0.56	0.53	2.22	104.64
	sharpe_ratio	1227	0.33	2.01	-8.48	-0.35	0.14	0.77	22.11
	log_equity	1227	0.12	1.60	-10.48	-0.39	0.01	0.47	21.01

Table A.2: Question 1 - Profit by episode and reward type (test data)

Episode Type	Reward Type	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
Spike	vwap_delta	1402	135.10	440.06	-461.29	27.99	65.24	138.37	9000.00
	abs_return	1402	126.51	416.48	-454.88	25.86	61.54	134.43	9000.00
	sharpe_ratio	1402	54.85	260.99	-478.22	-1.72	18.59	62.66	7834.30
	log_equity	1402	20.20	157.48	-775.88	-17.04	4.39	40.41	3392.52
Uptrend	vwap_delta	374	22.98	35.88	-24.51	7.63	14.90	25.99	445.70
	abs_return	374	22.07	35.95	-20.08	7.06	13.38	25.06	439.35
	sharpe_ratio	374	13.34	22.58	-21.76	3.39	8.28	15.55	300.51
	log_equity	374	10.23	18.73	-91.26	2.35	6.65	14.35	175.48
Downtrend	vwap_delta	478	-8.41	16.72	-55.49	-14.37	-9.07	-2.85	154.73
	abs_return	478	-7.24	15.68	-55.49	-13.60	-7.99	-1.44	142.05
	sharpe_ratio	478	-5.03	12.24	-50.56	-9.90	-4.82	-1.06	173.47
	log_equity	478	-5.01	15.29	-177.42	-9.29	-4.66	-0.51	161.52
No trend	vwap_delta	1227	8.60	26.10	-79.50	-3.65	3.27	14.41	305.75
	abs_return	1227	9.38	26.55	-43.58	-3.46	3.75	14.39	316.15
	sharpe_ratio	1227	4.61	23.04	-121.78	-3.31	1.48	7.72	336.93
	log_equity	1227	2.87	25.19	-196.39	-4.69	0.12	6.25	334.20

Table A.3: Question 2 - Profit per volume traded by episode and model type (test data)

Episode Type	Model Type	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
Spike	A2C	1402	27.76	85.12	-35.08	6.29	13.69	28.28	1957.21
	DQN	1402	36.01	124.76	-81.02	6.44	17.56	37.49	2521.85
Uptrend	A2C	374	7.68	8.42	-4.48	3.29	5.63	9.25	77.56
	DQN	374	11.00	16.65	-4.48	3.96	7.45	12.32	210.25
Downtrend	A2C	478	-4.32	6.08	-36.69	-6.62	-3.84	-0.94	35.73
	DQN	478	-5.90	7.47	-88.71	-8.18	-5.16	-2.58	46.75
No trend	A2C	1227	2.24	8.05	-22.70	-1.03	1.33	4.10	143.42
	DQN	1227	2.36	12.08	-92.36	-2.54	0.59	4.25	143.42

Table A.4: Question 2 - Profit by episode and model type (test data)

Episode Type	Model Type	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
Spike	A2C	1402	141.33	440.45	-210.46	31.81	69.95	148.43	9000.00
	DQN	1402	142.37	453.72	-204.37	18.94	64.00	151.31	9000.00
Uptrend	A2C	374	24.65	37.30	-8.96	8.59	15.86	28.03	465.38
	DQN	374	22.44	33.99	-8.96	7.92	14.91	24.72	420.51
Downtrend	A2C	478	-8.28	16.15	-73.37	-14.81	-8.48	-2.61	142.91
	DQN	478	-11.07	18.79	-177.42	-16.36	-10.33	-5.15	187.02
No trend	A2C	1227	10.67	28.82	-55.70	-2.98	4.22	16.26	308.94
	DQN	1227	6.58	30.80	-184.73	-5.10	1.21	9.20	385.68

Table A.5: Question 3 - Profit per volume traded by episode type and trading strategy (test data)

Episode Type	Strategy	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
Spike	RL agent	1402	27.76	85.12	-35.08	6.29	13.69	28.28	1957.21
	Spike	1402	32.82	100.10	-161.46	7.95	16.56	34.32	2250.00
	Random	1402	0.38	14.10	-316.68	-1.16	0.07	1.60	250.00
	Forecast	1402	1.56	17.80	-77.20	-0.91	0.03	1.39	344.51
	Moving avg.	1402	-6.15	17.50	-171.55	-8.35	-2.33	0.00	216.64
Uptrend	RL agent	374	7.68	8.42	-4.48	3.29	5.63	9.25	77.56
	Spike	374	10.81	17.88	-4.48	3.88	7.25	12.19	248.46
	Random	374	1.07	1.60	-4.24	0.20	0.72	1.50	11.93
	Forecast	374	2.27	5.23	-5.82	0.00	0.77	2.79	61.34
	Moving avg.	374	5.04	9.28	-5.46	0.76	2.56	6.10	102.49
Downtrend	RL agent	478	-4.32	6.08	-36.69	-6.62	-3.84	-0.94	35.73
	Spike	478	-6.00	6.50	-88.71	-8.05	-5.08	-2.43	5.22
	Random	478	-0.81	4.15	-88.71	-1.02	-0.49	-0.12	4.26
	Forecast	478	-1.15	3.21	-41.19	-1.80	-0.44	0.00	20.36
	Moving avg.	478	-1.67	4.90	-88.71	-2.12	-1.02	-0.16	13.17
No trend	RL agent	1227	2.24	8.05	-22.70	-1.03	1.33	4.10	143.42
	Spike	1227	1.75	11.10	-92.36	-2.44	0.58	3.96	152.80
	Random	1227	0.03	1.36	-11.38	-0.41	-0.02	0.44	12.97
	Forecast	1227	-0.03	3.51	-32.55	-0.83	0.00	0.75	47.49
	Moving avg.	1227	0.03	3.83	-16.44	-1.35	-0.34	0.72	35.84

Table A.6: Question 3 - Profit by episode type and trading strategy (test data)

Episode Type	Strategy	Episodes	Mean	Std. Dev.	Min	25%	Median	75%	Max
Spike	RL agent	1402	141.33	440.45	-210.46	31.81	69.95	148.43	9000.00
	Spike	1402	148.63	447.16	-645.83	36.18	72.57	151.95	9000.00
	Random	1402	16.70	288.46	-774.37	-23.14	1.32	32.14	9000.00
	Forecast	1402	13.14	196.70	-926.41	-6.85	0.21	10.06	5512.09
	Moving avg.	1402	-54.29	131.05	-910.61	-73.61	-24.82	0.00	1733.13
Uptrend	RL agent	374	24.65	37.30	-8.96	8.59	15.86	28.03	465.38
	Spike	374	23.59	37.53	-8.96	7.89	14.76	26.54	496.91
	Random	374	10.43	26.20	-144.29	1.58	6.32	13.69	321.98
	Forecast	374	8.07	15.00	-20.46	0.00	3.32	10.91	122.69
	Moving avg.	374	15.36	32.65	-68.01	3.40	9.43	19.21	409.98
Downtrend	RL agent	478	-8.28	16.15	-73.37	-14.81	-8.48	-2.61	142.91
	Spike	478	-12.27	13.52	-177.42	-16.96	-10.51	-5.06	20.89
	Random	478	-6.29	14.86	-177.42	-9.98	-4.99	-1.57	144.75
	Forecast	478	-4.01	14.49	-145.25	-7.54	-1.98	0.00	162.84
	Moving avg.	478	-7.30	15.74	-177.42	-8.70	-3.33	-0.46	131.68
No trend	RL agent	1227	10.67	28.82	-55.70	-2.98	4.22	16.26	308.94
	Spike	1227	4.77	25.95	-184.73	-5.24	1.21	9.73	305.60
	Random	1227	0.48	19.19	-194.59	-5.59	-0.24	5.62	156.00
	Forecast	1227	0.84	18.82	-170.46	-4.15	0.00	3.98	284.93
	Moving avg.	1227	-2.31	31.31	-193.68	-8.85	-1.85	3.62	679.89

A.9 Forecast Baseline

To compare the approach of reinforcement learning with supervised learning, the neural network used by the forecast trader is displayed in figure A.21. This network is *not* based on reinforcement learning. Instead, it is based on supervised learning. Nevertheless, it resembles the other three reinforcement learning based networks from figures 2.3, 2.5 and 2.4. However, instead of giving an expected return or a policy as output, this network returns a price forecast as described in section 5.4.2.

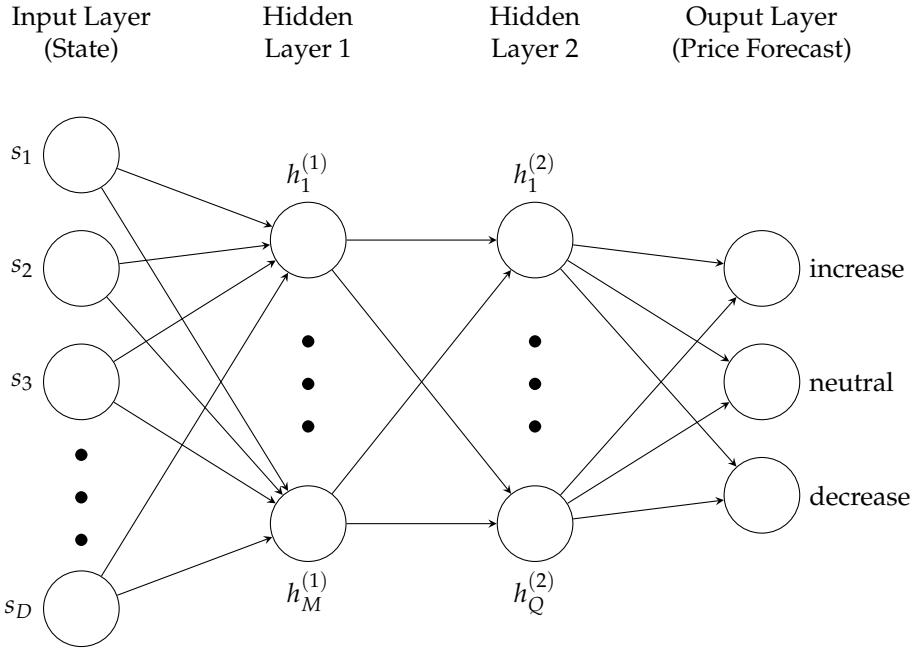


Figure A.21: NN for classifying price percentage changes from time t to $t+1$ into three classes: increase, decrease, neutral. In this thesis, these classes correspond to price percentage changes of $> 3\%$, $< -3\%$ or inbetween. This forecast model is trained with *supervised learning*, not with reinforcement learning. Forecasts can be mapped to trading actions explicitly, forming the baseline trading strategy called *forecast trader*.

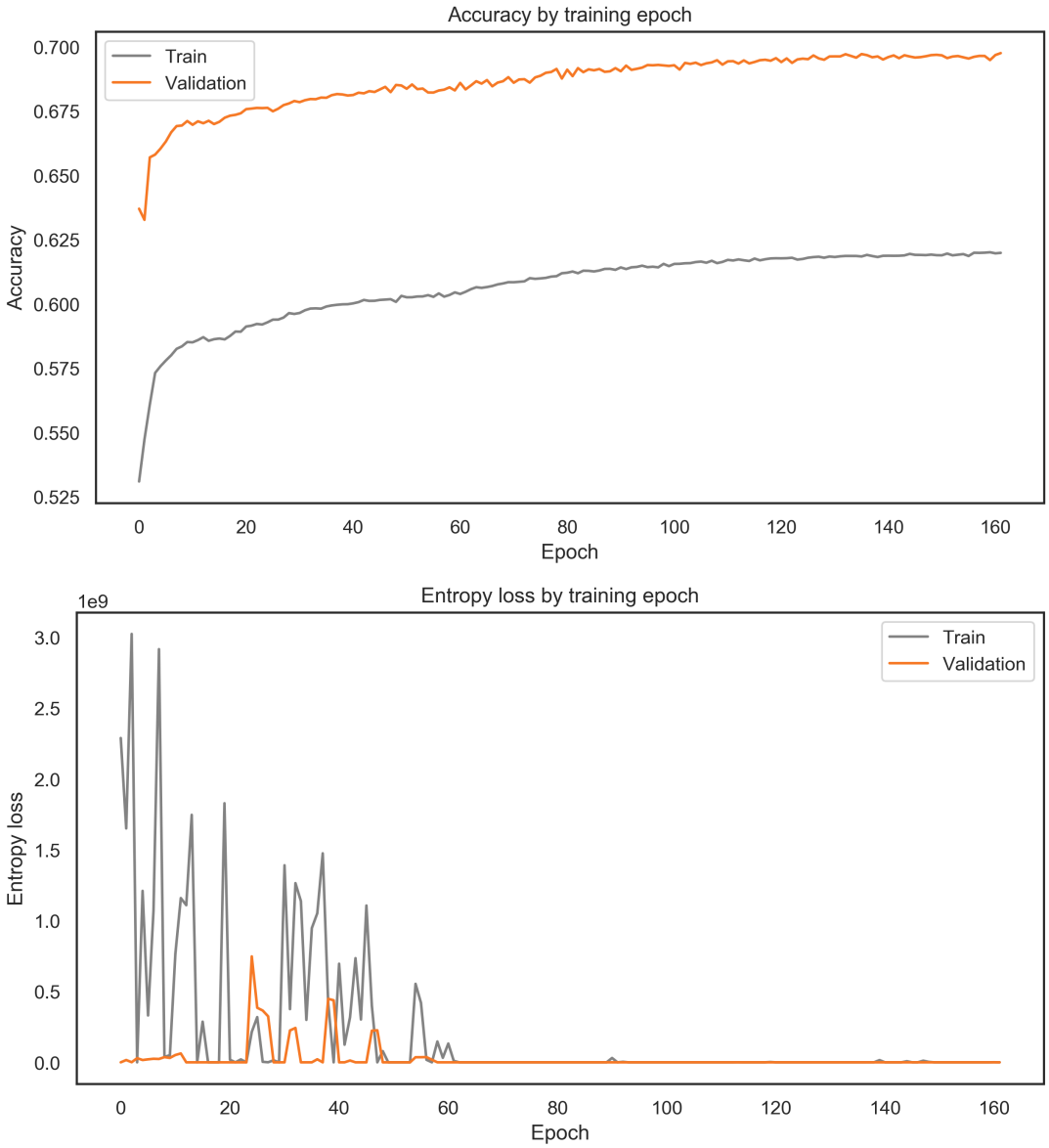


Figure A.22: Question 3 - Learning curves of the forecast model for the forecast baseline strategy (training data)

Table A.7: Question 3 - Classification accuracy of forecast model and majority classifier

	Forecast model	Majority classifier
Training	0.64	0.59
Test	0.73	0.70

Table A.8: Question 3 - Forecast confusion matrix (training data)

		Predicted			
		decr	same	incr	Total
True	decr	0.05	0.14	0.02	0.21
	same	0.01	0.57	0.01	0.59
	incr	0.02	0.17	0.02	0.20
	Total	0.08	0.88	0.04	1.00

Table A.9: Question 3 - Forecast classification report (training data)

	Precision	Recall	F1-score	Support
decr	0.64	0.26	0.37	21133
incr	0.43	0.07	0.13	21809
same	0.75	0.98	0.85	98848
Macro avg.	0.61	0.44	0.45	141790

Table A.10: Question 3 - Forecast confusion matrix (test data)

		Predicted			
		decr	same	incr	Total
True	decr	0.04	0.10	0.01	0.15
	same	0.01	0.68	0.01	0.70
	incr	0.01	0.13	0.01	0.15
	Total	0.06	0.91	0.03	1.00

Table A.11: Question 3 - Forecast classification report (test data)

	Precision	Recall	F1-score	Support
decr	0.62	0.24	0.35	105394
incr	0.45	0.09	0.15	104563
same	0.65	0.97	0.78	300147
Macro avg.	0.57	0.43	0.43	510104

A.10 Hyperparameters

Table A.12: Overview of hyperparameters (DQN)

Parameter	Symbol	Description
seq_len	-	Number of past timesteps used in the state space
total_timesteps	-	Number of steps taken during training phase in total
double_q	-	Indicator specifying if Double-Q learning is used
prioritized_replay	-	Indicator specifying if prioritized replay buffer is used
prioritized_replay_alpha	-	Parameter for prioritized replay sampling
prioritized_replay_beta0	-	Parameter for prioritized replay sampling
prioritized_replay_eps	-	Epsilon to add to TD errors in priorized replay
gamma	γ	Discount factor
learning_rate	α	Learning rate
exploration_fraction	ϵ	Proportion of randomly sampled actions for exploration
buffer_size	-	Size of the replay buffer
target_network_update_freq	-	Number of steps before every update of target network
train_freq	-	Number of steps before every update
batch_size	-	Size of batched sample from replay buffer for update
learning_starts	-	Number of steps before the first update

Table A.13: Search space of hyperparameters (DQN)

Parameter	Default	Best	Search Space	Type
seq_len	10	10	{10, 15, 20}	categorical
total_timesteps	500K	250K	{25K, 50K, 100K, 250K, 500K, 750K}	categorical
double_q	True	False	{True, False}	categorical
prioritized_replay	False	True	{True, False}	categorical
prioritized_replay_alpha	0.6	0.28	[0.01, 0.99]	float
prioritized_replay_beta0	0.4	0.74	[0.01, 0.99]	float
prioritized_replay_eps	1e-06	0.01	[1e-08, 0.01]	float
gamma	0.99	0.94	[0.9, 1]	float
learning_rate	0.0005	0.006	[5e-05, 0.01]	float
exploration_fraction	0.1	0.15	[5e-05, 0.2]	float
buffer_size	50000	32497	[10, 100000]	integer
target_network_update_freq	500	1744	[100, 2000]	integer
train_freq	1	8	[1, 10]	integer
batch_size	32	22	[5, 45]	integer
learning_starts	1000	795	[10, 3000]	integer

Table A.14: Overview of hyperparameters (A2C)

Parameter	Symbol	Description
seq_len	-	Number of past timesteps used in the state space
total_timesteps	-	Number of steps taken during training phase in total
lr_schedule	-	Type of learning rate update during training
learning_rate	η	Learning rate
gamma	γ	Discount factor
alpha	α	Decay parameter for parameter updates with RMSProp
epsilon	ϵ	Proportion of randomly sampled actions for exploration
ent_coef	-	Entropy coefficient for loss calculation
vf_coef	-	State-value function coefficient for loss calculation
n_steps	-	Number of steps to run for each environment per update

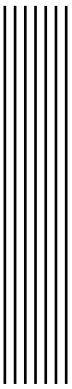
Table A.15: Search space of hyperparameters (A2C)

Parameter	Default	Search Space	Type
seq_len	10	{10, 15, 20}	categorical
total_timesteps	500K	{25K, 50K, 100K, 250K, 500K, 750K}	categorical
lr_schedule	constant	{linear, constant, middle_drop}	categorical
learning_rate	0.0007	[0.0001, 0.01]	float
gamma	0.99	[0.9, 1]	float
alpha	0.99	[0.9, 1]	float
epsilon	1e-05	[1e-06, 0.01]	float
ent_coef	0.01	[0.001, 1]	float
vf_coef	0.25	[0.001, 1]	float
n_steps	5	[1, 10]	integer

A.11 Selected Features

Table A.16: Question 3 - Selected features with Bayesian optimization (for A2C and DQN)

Index	Feature	Type	Selected for DQN	Selected for A2C
1	price	market	✓	✓
2	price_min	market	✗	✗
3	price_max	market	✗	✓
4	price_delta	market	✓	✗
5	volume	market	✓	✗
6	VWAP	indicator	✓	✓
7	SMA5	indicator	✓	✗
8	SMA10	indicator	✓	✗
9	EMA5	indicator	✓	✗
10	EMA10	indicator	✓	✗
11	bollinger10_hband	indicator	✓	✓
12	bollinger10_lband	indicator	✗	✗
13	MACD	indicator	✗	✓
14	RSI	indicator	✓	✗
15	time_e_month	time	✓	✓
16	time_e_weekday	time	✓	✗
17	time_e_hour	time	✗	✗
18	time_instrument	time	✗	✗
19	time_delta_delivery_s	time	✗	✓
20	time_delta_delivery_e	time	✗	✗
21	time_delta_e_s	time	✗	✗



Bibliography

[1] J. E. Moody, M. Saffell, Y Liao, and L Wu, "Reinforcement Learning for Trading Systems and Portfolios.", in *KDD*, 1998, pp. 279–283.

[2] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios", *Journal of Forecasting*, vol. 17, no. 5-6, pp. 441–470, 1998.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning", *arXiv preprint arXiv:1312.5602*, 2013.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", in *International conference on machine learning*, 2016, pp. 1928–1937.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and others, "Mastering the game of Go with deep neural networks and tree search", *nature*, vol. 529, no. 7587, p. 484, 2016, Publisher: Nature Publishing Group.

[6] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, and others, "Dota 2 with Large Scale Deep Reinforcement Learning", *arXiv preprint arXiv:1912.06680*, 2019.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, and others, "Grandmaster level in StarCraft II using multi-agent reinforcement learning", *Nature*, vol. 575, no. 7782, pp. 350–354, 2019, Publisher: Nature Publishing Group.

[8] Z. Zhang, S. Zohren, and S. Roberts, "Deep Reinforcement Learning for Trading", *arXiv preprint arXiv:1911.10107*, 2019.

[9] G. Bertrand and A. Papavasiliou, "Reinforcement-Learning Based Threshold Policies for Continuous Intraday Electricity Market Trading", in *IEEE PES General Meeting, Atlanta*, 2019.

[10] A. Weidlich and D. Veit, "Bidding in interrelated day-ahead electricity markets: Insights from an agent-based simulation model", in *Proceedings of the 29th IAEE international conference, Potsdam*, Citeseer, 2006.

[11] A. Rahimi-Kian, B. Sadeghi, and R. J. Thomas, "Q-learning based supplier-agents for electricity markets", in *IEEE Power Engineering Society General Meeting, 2005*, IEEE, 2005, pp. 420–427.

[12] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[13] J. Moody and M. Saffell, "Learning to trade via direct reinforcement", *IEEE transactions on neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[14] I. Boukas, D. Ernst, A. Papavasiliou, and B. Cornélusse, "Intra-day Bidding Strategies for Storage Devices Using Deep Reinforcement Learning", in *International Conference on the European Energy Market, \Lódź 27-29 June 2018*, 2018, p. 6.

[15] T. G. Fischer, "Reinforcement learning in financial markets - a survey", Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics, FAU Discussion Papers in Economics 12/2018, 2018. [Online]. Available: `https://ideas.repec.org/p/zbw/iwqwdp/122018.html`.

[16] X. Gao and L. Chan, "An algorithm for trading and portfolio management using Q-learning and sharpe ratio maximization", in *Proceedings of the international conference on neural information processing*, 2000, pp. 832–837.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] M. Gail, K. Krickeberg, J Samet, A. Tsiatis, and W. Wong, *Statistics for biology and health*. Springer, 2007.

[19] R. L. Winkler, *An introduction to Bayesian inference and decision*. Probabilistic Pub., 2003, ISBN: 0-9647938-4-9. [Online]. Available: `https://login.e.bibl.liu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=cat00115a&AN=lkp.992737&lang=sv&site=eds-live&scope=site`.

[20] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[21] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning", *arXiv preprint arXiv:1811.03378*, 2018.

[22] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms", in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[23] *OpenAI Baselines: ACKTR & A2C*, en, Aug. 2017. [Online]. Available: `https://openai.com/blog/baselines-acktr-a2c/` (visited on 02/10/2020).

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, Publisher: Nature Publishing Group.

[25] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay", *arXiv preprint arXiv:1511.05952*, 2015.

[26] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning", en, *arXiv:1511.06581 [cs]*, Apr. 2016, arXiv: 1511.06581. [Online]. Available: `http://arxiv.org/abs/1511.06581` (visited on 04/11/2020).

[27] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning", in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[28] R. Neuneier, "Enhancing Q-learning for optimal asset allocation", in *Advances in neural information processing systems*, 1998, pp. 936–942.

[29] W. F. Sharpe, "Mutual fund performance", *The Journal of business*, vol. 39, no. 1, pp. 119–138, 1966.

[30] M. Bendtsen and J. M. Peña, "Learning gated Bayesian networks for algorithmic trading", in *European Workshop on Probabilistic Graphical Models*, Springer, 2014, pp. 49–64.

[31] C. Gold, "FX trading via recurrent reinforcement learning", in *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.*, IEEE, 2003, pp. 363–370.

[32] M. Corazza and F. Bertoluzzo, "Q-learning-based financial trading systems with applications", *University Ca'Foscari of Venice, Dept. of Economics Working Paper Series No*, vol. 15, 2014.

[33] F. Bertoluzzo and M. Corazza, "Testing different reinforcement learning configurations for financial trading: Introduction and applications", *Procedia Economics and Finance*, vol. 3, pp. 68–77, 2012.

[34] A. A. Sherstov and P. Stone, "Three automated stock-trading agents: A comparative study", in *International Workshop on Agent-Mediated Electronic Commerce*, Springer, 2004, pp. 173–187.

[35] O. Jin and H. El-Saawy, "Portfolio management using reinforcement learning", *Stanford University*, 2016.

[36] Z. Jiang and J. Liang, "Cryptocurrency portfolio management with deep reinforcement learning", in *2017 Intelligent Systems Conference (IntelliSys)*, IEEE, 2017, pp. 905–913.

[37] J. Lago, F. De Ridder, and B. De Schutter, "Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms", *Applied Energy*, vol. 221, pp. 386–405, 2018, Publisher: Elsevier.

[38] T. Jamasb and M. Pollitt, "Electricity market reform in the European Union: Review of progress toward liberalization & integration", *The Energy Journal*, vol. 26, no. Special Issue, 2005, Publisher: International Association for Energy Economics.

[39] F. Ziel, "Modeling the impact of wind and solar power forecasting errors on intraday electricity prices", in *2017 14th International Conference on the European Energy Market (EEM)*, IEEE, 2017, pp. 1–5.

[40] *Why does the electricity grid have to stay in balance?*, en, Library Catalog: www.energuide.be. [Online]. Available: `https://www.energuide.be/en/questions-answers/why-does-the-electricity-grid-have-to-stay-in-balance/2136/` (visited on 05/14/2020).

[41] M. Sewalt and C. De Jong, "Negative prices in electricity markets", *Commodities Now*, vol. 7, pp. 74–77, 2003.

[42] Y. Zhou, A. Scheller-Wolf, N. Secomandi, and S. Smith, "Electricity trading and negative prices: Storage vs. disposal", *Management Science*, vol. 62, no. 3, pp. 880–898, 2016, Publisher: INFORMS.

[43] I. Richardson, M. Thomson, D. Infield, and C. Clifford, "Domestic electricity use: A high-resolution energy demand model", *Energy and buildings*, vol. 42, no. 10, pp. 1878–1887, 2010, Publisher: Elsevier.

[44] R. Weron, "Electricity price forecasting: A review of the state-of-the-art with a look into the future", *International journal of forecasting*, vol. 30, no. 4, pp. 1030–1081, 2014, Publisher: Elsevier.

[45] R. Huisman, C. Huurman, and R. Mahieu, "Hourly electricity prices in day-ahead markets", *Energy Economics*, vol. 29, no. 2, pp. 240–248, 2007, Publisher: Elsevier.

[46] N. V. Karakatsani and D. W. Bunn, "Modelling the volatility of spot electricity prices", in *2nd Energy Risk Management Seminar*, 2004.

[47] R. Kiesel and F. Paraschiv, "Econometric analysis of 15-minute intraday electricity prices", *Energy Economics*, vol. 64, pp. 77–90, 2017, Publisher: Elsevier.

[48] A. Ciarreta and A. Zarraga, "Modeling realized volatility on the Spanish intra-day electricity market", *Energy Economics*, vol. 58, pp. 152–163, 2016, Publisher: Elsevier.

[49] S. Hagemann, "Price determinants in the german intraday market for electricity: An empirical analysis", *Journal of Energy Markets*, 2015.

[50] N. V. Karakatsani and D. W. Bunn, "Fundamental and behavioural drivers of electricity price volatility", *Studies in Nonlinear Dynamics & Econometrics*, vol. 14, no. 4, 2010, Publisher: De Gruyter.

[51] ——, "Intra-day and regime-switching dynamics in electricity price formation", *Energy Economics*, vol. 30, no. 4, pp. 1776–1797, 2008, Publisher: Elsevier.

[52] K. Maciejowska and R. Weron, "Short-and mid-term forecasting of baseload electricity prices in the UK: The impact of intra-day price relationships and market fundamentals", *IEEE Transactions on power systems*, vol. 31, no. 2, pp. 994–1005, 2015.

[53] G. Bertrand and A. Papavasiliou, "Adaptive Trading in Continuous Intraday Electricity Markets for a Storage Unit", *IEEE Transactions on Power Systems*, 2019, Publisher: IEEE.

[54] D. Cao, W. Hu, X. Xu, T. Dragičević, Q. Huang, Z. Liu, Z. Chen, and F. Blaabjerg, "Bidding strategy for trading wind energy and purchasing reserve of wind power producer–A DRL based approach", *International Journal of Electrical Power & Energy Systems*, vol. 117, p. 105 648, 2020, Publisher: Elsevier.

[55] B. M. Radhakrishnan, D. Srinivasan, Y. F. A. Lau, B. G. Parasumanna, A. K. Rathore, S. K. Panda, and A. Khambadkone, "A reinforcement learning algorithm for agent-based computational economics (ACE) model of electricity markets", in *2015 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2015, pp. 297–303.

[56] H. Zhao, Y. Wang, S. Guo, M. Zhao, and C. Zhang, "Application of a gradient descent continuous actor-critic algorithm for double-side day-ahead electricity market modeling", *Energies*, vol. 9, no. 9, p. 725, 2016, Publisher: Multidisciplinary Digital Publishing Institute.

[57] H. Zhao, Y. Wang, M. Zhao, C. Sun, and Q. Tan, "Application of Gradient Descent Continuous Actor-Critic Algorithm for Bilateral Spot Electricity Market Modeling Considering Renewable Power Penetration", en, *Algorithms*, vol. 10, no. 2, p. 53, May 2017, ISSN: 1999-4893. DOI: 10.3390/a10020053. [Online]. Available: http://www.mdpi.com/1999-4893/10/2/53 (visited on 03/27/2020).

[58] M. Peters, W. Ketter, M. Saar-Tsechansky, and J. Collins, "Autonomous data-driven decision-making in smart electricity markets", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2012, pp. 132–147.

[59] ——, "A reinforcement learning approach to autonomous decision-making in smart electricity markets", *Machine learning*, vol. 92, no. 1, pp. 5–39, 2013, Publisher: Springer.

[60] S. Vandael, B. Claessens, D. Ernst, T. Holvoet, and G. Deconinck, "Reinforcement learning of heuristic EV fleet charging in a day-ahead electricity market", *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1795–1805, 2015, Publisher: IEEE.

[61] X. Du, J. Zhai, and K. Lv, "Algorithm trading using q-learning and recurrent reinforcement learning", *positions*, vol. 1, p. 1, 2016.

[62] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading", *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 653–664, 2016.

[63] D. Gorse, "Application of stochastic recurrent reinforcement learning to index trading", ESANN, 2011.

[64] F. Bertoluzzo and M. Corazza, "Making Financial Trading by Recurrent Reinforcement Learning", 2007, pp. 619–626. DOI: 10.1007/978-3-540-74827-4_78.

[65] S. D. Bekiros, "Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach", *Journal of Economic Dynamics and Control*, vol. 34, no. 6, pp. 1153–1170, 2010, Publisher: Elsevier.

[66] M. A. Dempster, T. W. Payne, Y. Romahi, and G. W. Thompson, "Computational learning techniques for intraday FX trading using popular technical indicators", *IEEE Transactions on neural networks*, vol. 12, no. 4, pp. 744–754, 2001.

[67] R. G. Bates, M. A. Dempster, and Y. S. Romahi, "Evolutionary reinforcement learning in FX order book and order flow analysis", in *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.*, IEEE, 2003, pp. 355–362.

[68] Y. Dai, C. Wang, I. Wang, and Y. Xu, "Reinforcement Learning for FX trading", en, Tech. Rep., 2019, p. 14.

[69] O Jangmin, J. Lee, J. W. Lee, and B.-T. Zhang, "Adaptive stock trading with dynamic asset allocation using reinforcement learning", *Information Sciences*, vol. 176, no. 15, pp. 2121–2147, 2006.

[70] Y. Shen and Y. Zhao, "Deep Reinforcement Learning for Pairs Trading Using Actor-critic", en, Tech. Rep., 2017, p. 7.

[71] Y. Deng, Y. Kong, F. Bao, and Q. Dai, "Sparse coding-inspired optimal trading system for HFT industry", *IEEE Transactions on Industrial Informatics*, vol. 11, no. 2, pp. 467–475, 2015.

[72] W. Si, J. Li, P. Ding, and R. Rao, "A multi-objective deep reinforcement learning approach for stock index future's intraday trading", in *2017 10th International symposium on computational intelligence and design (ISCID)*, vol. 2, IEEE, 2017, pp. 431–436.

[73] J. E. Moody and M. Saffell, "Reinforcement learning for trading", in *Advances in Neural Information Processing Systems*, 1999, pp. 917–923.

[74] C. Mitchell, *Volume Weighted Average Price (VWAP) Definition*, en. [Online]. Available: https://www.investopedia.com/terms/v/vwap.asp (visited on 02/16/2020).

[75] A. Hayes, *Simple Moving Average (SMA) Definition*, en. [Online]. Available: https://www.investopedia.com/terms/s/sma.asp (visited on 02/16/2020).

[76] ——, *Exponential Moving Average - EMA*, en. [Online]. Available: https://www.investopedia.com/terms/e/ema.asp (visited on 02/16/2020).

[77] ——, *Bollinger Band®*, en. [Online]. Available: https://www.investopedia.com/terms/b/bollingerbands.asp (visited on 02/16/2020).

[78] ——, *Moving Average Convergence Divergence – MACD Definition*, en. [Online]. Available: https://www.investopedia.com/terms/m/macd.asp (visited on 02/16/2020).

[79] J. Chen, *Relative Strength Index – RSI*, en. [Online]. Available: https://www.investopedia.com/terms/r/rsi.asp (visited on 02/16/2020).

[80] R. W. Colby and T. A. Meyers, *The encyclopedia of technical market indicators*. Dow Jones-Irwin Homewood, IL, 1988.

[81] R. Weron and A. Misiorek, "Forecasting spot electricity prices: A comparison of parametric and semiparametric time series models", *International journal of forecasting*, vol. 24, no. 4, pp. 744–763, 2008, Publisher: Elsevier.

[82] B. Lim, S. Zohren, and S. Roberts, "Enhancing time-series momentum strategies using deep neural networks", *The Journal of Financial Data Science*, vol. 1, no. 4, pp. 19–38, 2019.

[83] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection", *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[84] T. Yu and H. Zhu, *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. 2020, _eprint: 2003.05689.

[85] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[86] S. Khan, *What is the best ways to tune multiple parameters? - Quora*, 2016. [Online]. Available: `https://www.quora.com/What-is-the-best-ways-to-tune-multiple-parameters` (visited on 02/16/2020).

[87] M. Feurer and F. Hutter, "Hyperparameter optimization", in *Automated Machine Learning*, Springer, 2019, pp. 3–33.

[88] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and others, "Mastering the game of go without human knowledge", *Nature*, vol. 550, no. 7676, pp. 354–359, 2017, Publisher: Nature Publishing Group.

[89] L. Li, J. D. Williams, and S. Balakrishnan, "Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection", in *Tenth Annual Conference of the International Speech Communication Association*, 2009.

[90] D.-R. Liu, H.-L. Li, and D. Wang, "Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey", *International Journal of Automation and Computing*, vol. 12, no. 3, pp. 229–242, 2015.

[91] T. Nguyen, Z. Li, T. Silander, and T. Y. Leong, "Online feature selection for model-based reinforcement learning", in *International Conference on Machine Learning*, 2013, pp. 498–506.

[92] P. I. Frazier, "A tutorial on bayesian optimization", *arXiv preprint arXiv:1807.02811*, 2018.

[93] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization", *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015, Publisher: IEEE.

[94] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[95] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization", in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[96] A. E. Biondo, A. Pluchino, A. Rapisarda, and D. Helbing, "Are random trading strategies more successful than technical ones?", *PloS one*, vol. 8, no. 7, e68344, 2013.

[97] V. Zakamulin, "Market timing with moving averages: Anatomy and performance of trading rules", *Available at SSRN 2585056*, 2016.

[98] A. C. Davison and D. V. Hinkley, *Bootstrap methods and their application*. Cambridge university press, 1997, vol. 1.

[99] T. Hesterberg, D. S. Moore, and S. Monaghan, "Bootstrap Methods and Permutation Tests* 14.1 the Bootstrap Idea 14.2 First Steps in Using the Bootstrap 14.3 How Accurate Is a Bootstrap Distribution? 14.4 Bootstrap Confidence Intervals 14.5 Significance Testing Using Permutation Tests Introduction", 2004.

[100] *Unpaired Two-Samples T-test in R - Easy Guides - Wiki - STHDA*, en. [Online]. Available: http://www.sthda.com/english/wiki/unpaired-two-samples-t-test-in-r (visited on 03/02/2020).

[101] *Paired Samples Wilcoxon Test in R - Easy Guides - Wiki - STHDA*, en. [Online]. Available: http://www.sthda.com/english/wiki/paired-samples-wilcoxon-test-in-r (visited on 03/02/2020).

[102] *Paired Samples T-test in R - Easy Guides - Wiki - STHDA*, en. [Online]. Available: http://www.sthda.com/english/wiki/paired-samples-t-test-in-r (visited on 03/02/2020).

[103] J. G. MacKinnon, "Bootstrap hypothesis testing", *Handbook of computational econometrics*, vol. 183, p. 213, 2009, Publisher: Wiley Online Library.

[104] P. Hall and S. R. Wilson, "Two guidelines for bootstrap hypothesis testing", *Biometrics*, pp. 757–762, 1991, Publisher: JSTOR.

[105] B. Efron, "Better bootstrap confidence intervals", *Journal of the American statistical Association*, vol. 82, no. 397, pp. 171–185, 1987.

[106] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA: 2015, vol. 2018.

[107] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014, Publisher: JMLR. org.

[108] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable Baselines*. GitHub, 2018. [Online]. Available: https://github.com/hill-a/stable-baselines.

[109] *Interquartile range*, en. Apr. 2020, Publication Title: Wikipedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Interquartile_range&oldid=953495961 (visited on 05/15/2020).

[110] B. Beers, *How a Buy-and-Hold Strategy Works*, en, Library Catalog: www.investopedia.com. [Online]. Available: https://www.investopedia.com/terms/b/buyandhold.asp (visited on 05/07/2020).

[111] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay", in *Advances in neural information processing systems*, 2017, pp. 5048–5058.

[112] Investopedia, *A Look at the Buy Low, Sell High Strategy*, en, Library Catalog: www.investopedia.com. [Online]. Available: https://www.investopedia.com/articles/investing/081415/look-buy-low-sell-high-strategy.asp (visited on 04/28/2020).

[113] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network", *arXiv preprint arXiv:1503.02531*, 2015.

[114] T. Hailesilassie, "Rule extraction algorithm for deep neural networks: A review", *arXiv preprint arXiv:1610.05267*, 2016.

[115] H. Zhang and Q. Zhang, "Trading a mean-reverting asset: Buy low and sell high", *Automatica*, vol. 44, no. 6, pp. 1511–1518, 2008, Publisher: Elsevier.

[116] A. Cartea and M. G. Figueroa, "Pricing in electricity markets: A mean reverting jump diffusion model with seasonality", *Applied Mathematical Finance*, vol. 12, no. 4, pp. 313–335, 2005, Publisher: Taylor & Francis.

[117] E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. Vetrov, and A. G. Wilson, "Improving stability in deep reinforcement learning with weight averaging", in *Uncertainty in Artificial Intelligence Workshop on Uncertainty in Deep Learning*, vol. 5, 2018.

[118] J. E. Dayhoff and J. M. DeLeo, "Artificial neural networks: Opening the black box", *Cancer: Interdisciplinary International Journal of the American Cancer Society*, vol. 91, no. S8, pp. 1615–1635, 2001, Publisher: Wiley Online Library.

[119] *Lethal Autonomous Weapons Pledge - Future of Life Institute*. [Online]. Available: `https://futureoflife.org/lethal-autonomous-weapons-pledge/?cn-reloaded=1` (visited on 04/28/2020).

[120] M. P. Wellman and U. Rajan, "Ethical issues for autonomous trading agents", *Minds and Machines*, vol. 27, no. 4, pp. 609–624, 2017, Publisher: Springer.

[121] *How to start trading*. [Online]. Available: `https://www.eex.com/en/access/admission-neu/how-to-start-trading` (visited on 04/29/2020).

[122] *Huber loss*, en, Page Version ID: 950230412, Apr. 2020. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Huber_loss&oldid=950230412` (visited on 04/16/2020).