DEEP EXECUTION - VALUE AND POLICY BASED REINFORCEMENT LEARNING FOR TRADING AND BEATING MARKET BENCHMARKS

Kevin Dabérius[1,2] , Elvin Granat[1,2] and Patrik Karlsson[*2]

[1] Department of Computer Science, Linkoping University, Sweden.
[2] Markets, SEB, Stockholm, Sweden.

April 21, 2019

ABSTRACT. In this article we introduce the term *Deep Execution* that utilize deep reinforcement learning (DRL) for optimal execution. We demonstrate two different approaches to solve for the optimal execution: (1) the deep double Q-network (DDQN), a value-based approach and (2) the proximal policy optimization (PPO) a policy-based approach, for trading and beating market benchmarks, such as the time-weighted average price (TWAP). We show that, firstly, the DRL can reach the theoretically derived optimum by acting on the environment directly. Secondly, the DRL agents can learn to capitalize on price trends (alpha signals) without directly observing the price. Finally, the DRL can take advantage of the available information to create dynamic strategies as an informed trader and thus outperform static benchmark strategies such as the TWAP.

Keywords: Algorithmic Trading, Deep Learning, Execution Algorithms, Reinforcement Learning, Optimal Execution.

1. INTRODUCTION

When investors, primarily institutional traders, wish to buy or sell a large order of a financial asset, they usually delegate their trades to an agency broker, from here on referred to as a *trading agent*. The trading agent then divides the large parent order into child orders of smaller size and executes these trades over a trading horizon. However, there are two underlying risks the trading agent handles in attempt to execute the trades optimally: the *market impact*, the unfavourable price movements caused by a trade and the *price risk*, the unfavourable price movement occurring while waiting to trade. Therefore executing orders over a trading horizon balancing these risks while minimizing the *execution cost* involves complex dynamic optimization problems to determine the size, frequency, and type of the order, formally called the *optimal execution* problem. Some seminal works are, [BL98] demonstrated optimal execution strategies in the presence of temporary price impacts. [AC01] extended this by considering the agent's risk aversion that arises from breaking up orders and slowly executing them. Risk-aversion increase the agent's urgency to execute and willingness to pay risk premium to reduce its risk, inform of temporary market impact. But higher (lower) market impact encourage the trading agent to reduce (increase) its trading phase. Generally, approaches for optimal execution are primarily derived from modelling the problem at hand in a financial model. The financial model describes number of shares the trading agent is holding (inventory), the price dynamics, and price impact etc. Then, given the financial model one can derive the optimal execution strategy. This is typically done by setting up a *stochastic control problem* and solving it using *dynamic programming*, see for instance [CJP15] for recent overviews and references therein.

If we consider the trading agent's decision making as discrete in time we can model the environment that the trading agent operates on as a Markov decision process (MDP). In this problem setting, the machine learning area of *Reinforcement Learning* (RL) can be applied to solve for the optimal execution. In contrast to the analytically derived strategies, RL-algorithms must not know the true environment to find near optimal behaviour if it can operate on the environment directly, see for instance [SB18]. RL therefore enables solving the problem in a *model-free* setting, where it learns an optimal trading strategy/policy by directly interacting with the environment to maximize its cumulative long-term reward.

Recently, deep learning, using powerful function approximates via deep *neural networks* has demonstrated huge potential in the field of RL, see for instance [MKS$^+$13, MKS$^+$15, vHGS15, SLA$^+$15, SWD$^+$17, HMvH$^+$17, Sil16].

RL for optimal execution has previously been studied in [NFK06, HW14, NLJ18]. In [NFK06] they trained an algorithm to optimally select price levels to passively place limit orders. [HW14] utilized the standard Almgren-Chriss

The initial work, *Can FX trading benefit from machine learning*, was presented by Patrik Karlsson at QuantMinds, Lissbon, Portugal May 17, 2018. He wishes to thank the organization for the invitation, the audience for their interest, valuable questions and comments. The continued work was carried out while Elvin and Kevin were researchers at SEB Markets during November 2018 through March 2019. They want to thank, Samuel Samama, Hanna Hultin and David Rydberg for valuable comments.

[*]email: patrik.x.karlsson@seb.se

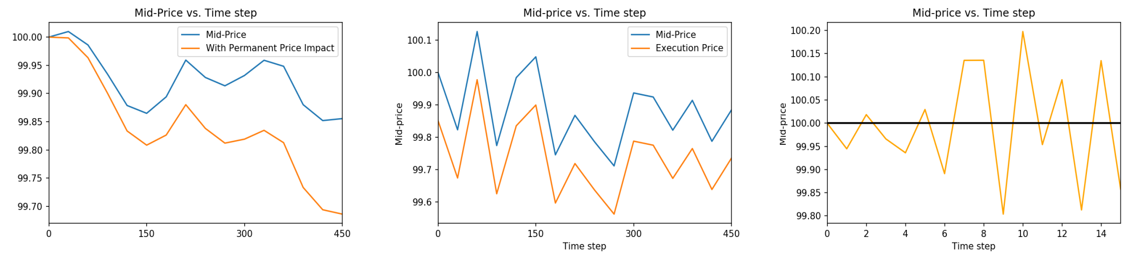


FIGURE 1. **Simulation Overview**. Left:mid-price is affected by permanent price impact. Middle: The execution price as compared to the mid-price. Right: Mean reversion around $S = 100$ ($\epsilon = 0.05$)

framework. [Fel16] demonstrated a workflow for calibration and back testing RL-algorithms using Q-learning. And finally [NLJ18] trained an agent using neural networks to outperform the time-weighted-average-price (TWAP) benchmark. For a great overview of some fundament RL for finance, we recommend chapter 13 in [Gui18].

This article focus on what we call *deep execution*, an execution sub-class that take advantage of deep reinforcement learning to solve for the optimal execution. The key is that we leave the task to define the features specification and optimization of the algorithm, instead of having it being "human programmed", like the traditional algorithmic trading methods described above. We chose to focus on the two main branches of RL, namely the *value-based* using proximal policy optimization (PPO) and the *policy-based* methods using deep double Q-network (DDQN). The value-based method attempts to learn the action-value function with a policy to greedily select the action with the highest action-value, whereas policy-based RL attempts to learn the optimal policy directly. We show their optimal execution while comparing their performance to a theoretically optimal baseline strategy.

The paper is organized as follows. Section 2 introduces notations, the general framework for algorithmic trading and the corresponding optimal execution problem. Section 3 formulates the deep reinforcement learning, the value- and policy-based methods. Section 4 connects the deep reinforcement learning and algorithmic trading, and describes the different simulation environments and the training process. In Section 5 we present the numerical examples to illustrate the methods and its efficiency compared to the traditional benchmark, the TWAP. Finally, we conclude in Section 6.

2. GENERAL FRAMEWORK FOR ALGORITHMIC TRADING

In this section we describe the some trading fundamentals, the financial model and the corresponding optimal execution control problem following [CJP15].

2.1. The Limit Order Book. A *Limit Order* (LO) $L = (\tau_L, q_L, p_L)$ of type $\tau_L = 1$ ($\tau_L = -1$) is a passive commitment to buy (sell) up to q_L units of an asset at a price no greater (less) than the price p_L. The *Limit Order Book* (LOB) Λ_t is the set of active limit orders L of an asset at time t. Since a LO must have a price p_L being a multiple of the tick size κ the LOB can include several orders that contain the same price, referred to as a *Price Level*. A price level is a price in the LOB that can contain multiple limit orders. There exists a price level for each positive multiple of tick size κ, i.e. the fixed price step between two neighbouring price levels is κ. The bid (ask) price is the highest (lowest) stated price on the buy (sell) side of the LOB at time t:

$$\left[S_t^{\text{bid}},\ S_t^{\text{ask}}\right] = \left[\max_{L\in\Lambda_t}\left\{p_L \middle| \tau_L = 1\right\},\qquad \min_{L\in\Lambda_t}\left\{p_L \middle| \tau_L = -1\right\}\right].$$

The mid-price S_t is the average of the bid-price and ask-price, and the bid-ask spread Δ_t, at time t, are given by

$$S_t = \left(S_t^{ask} + S_t^{bid}\right)/2, \qquad \Delta_t = S_t^{ask} - S_t^{bid},$$

A *Market Order* (MO) $M = (\tau_M, q_M)$ of type $\tau_M = 1$ ($\tau_M = -1$) is a commitment to buy (sell) q_L units of an asset. It is usually considered an aggressive order as it will execute a trade immediately without a price criterion. When submitting a market order to buy (sell) q_L units the order is immediately matched with LOs within the LOB, starting with matching at the ask-price (bid-price) with number of units q and then *"walking-the-book"* until the MO is completely filled or there is no volume left in the LOB to be matched against.

2.2. The Financial Model. The optimal execution problem, from the trading agent's perspective, is a function of the agent's inventory, the price dynamics (including e.g., mean-reversion and short-term alpha) and the agent's market impact. Under simple assumption the optimal strategy can be found analytically via the dynamic programming principle, which later will enable us to simulate the environment for our RL-algorithms to find the optimal execution strategies, empirically.

Market impact is the adverse impact that a trading agent's trades have on the price. It is especially important to consider for when submitting large orders since it can severely move the price due to the market response. Market impact can be divided into two portions: a *permanent* and a *temporary* market impact, see [AC01]. When buying (selling) an asset the permanent market impact is the trade's induced upward (downward) pressure on the price which in turn impacts the future price of the asset. The temporary market impact is the process whereby a large market order executes against increasingly worse price levels in the order book. This occurs when the number of units on the ask (bid) side of the order book q_{ask} (q_{bid}) is less than the number of units (q_M) of a market order, which is annotated ν_t here. It has been widely observed that many financial asset prices exhibit mean reversion [EH89, PS88, Sch97, Gro04]. An asset is said to be mean reverting if its returns are negatively correlated [PS88]. Prediction of the mean reversion opens up for better strategies for the trader. Inclusion of the mean reversion in the price process calls for a dynamic trading strategy, where the trading agent utilizes information available at decision point, which in turn makes TWAP sub-optimal given information of mean-reversion, which enables us to see if we can outperform TWAP using RL in a problem setting that is arguably more realistic.

The key stochastic processes for $t \in \{0 \leq t \leq T\}$ are:

- The **trading rate** ν_t: the speed of which the trading agent is buying (selling) an asset. The trading rate ν_t is the only variable the trading agent controls in the optimization problem.
- The **inventory** Q_t^ν: which is affected by the trading rate ν_t. When buying (selling) units of an asset with the trading rate ν_t the trading agent's inventory process Q_t^ν is given by
$$dQ_t^\nu = \pm \nu_t dt,$$
- The **mid-price process** S_t^ν: composed of a mean-reversion component α_t and a permanent market impact function $g(\cdot)$,
$$dS_t^\nu = \pm g(\nu_t) + \alpha_t + \sigma dW_t^S, \tag{2.1}$$
$$d\alpha_t = -\zeta \alpha_t + \beta W_t^\alpha + \epsilon dM_t^+ - \epsilon dM_t^-, \tag{2.2}$$
where W_t is a *Brownian motion*, Z_t^S and Z_t^α are independent standard normal distribution, $M_t^\pm$ are incoming market orders, Poisson distributed with mean $\lambda^\pm$ ($M_n^\pm \sim Po(\lambda^\pm)$) and ϵ is a constant. A simple but yet powerful assumption is to assume a linear permanent market impact function $g(\nu_t) = b\nu_t$, where the parameter b is often is called *Kyle's Lambda* [Kyl85].
- The **execution price** $\hat{S}_t^\nu$, affected by the trading rate ν_t. The execution price of a buy (sell) MO may be different from the ask (bid) since q_M can be larger than the volume q_{ask} (q_{bid}) available at the ask (bid) price
$$\hat{S}_t^\nu = S_t^\nu \pm (\tfrac{1}{2}\Delta + f(\nu_t)),$$
where $f(\nu_t)$ denotes the *temporary market impact* that the trading rate has on the price, which we also write on linear form $f(\nu_t) = k\nu_t$, where k is the coefficient tuned for each individual temporary impact estimation method. From now on, we assume that the bid-ask spread Δ is 0.
- The **cash process**: X_t^ν resulting from the agent's trading strategy is given by:
$$dX_t^\nu = \hat{S}_t^\nu \nu_t dt,$$
and with the terminal cash X_T^ν, i.e. the cash the agent will have at the end of trading given by
$$X_T^\nu = \int_0^T \hat{S}_t^\nu \nu_t dt. \tag{2.3}$$

2.3. **Optimal Execution.** The optimal execution problem concerns finding the trading strategy to optimally sell (buy) a specified amount units of an assets, during the time horizon $\{0 \leq t \leq T\}$, that minimizes the execution cost. A trading strategy is the realized trading rate ν_t which is the speed of which the trading agent places market orders to buy (sell) the number of units q_0 in time $\{0 \leq t \leq T\}$. We distinguish between two types of trading strategies, dynamics and static trading startegies. A *static trading strategy* is determined using the information available before the trading starts and a *dynamic trading strategy* one that utilizes information available at each decision point, e.g., [AC01]. For now, we define the control problem(s) in the perspective of selling, ignoring short-term alpha and only using MOs. With terminal cash in mind we can then setup an execution cost based on the initial state of the problem and the terminal cash. Where the execution cost which the trading agent seeks to minimize among all admissible strategies ν. We add a penalty if the agent has inventory Q_T^ν at the terminal state, where the remaining inventory will then be sold at the terminal price S_T^ν with a penalty factor $\varphi > 0$. We also add a running inventory penalty, which yields the following execution cost:
$$EC^\nu = q_0 s_0 - \mathbb{E}_{t,S,q}\Big[X_T^\nu + Q_T^\nu(S_T^\nu - \varphi Q_T^\nu) + \phi \int_t^T (Q_u^\nu)^2 \, du\Big], \tag{2.4}$$

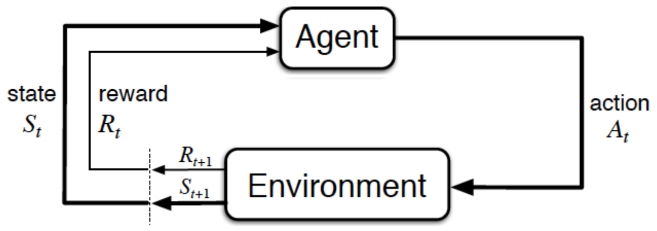


FIGURE 2. The RL-problem. The the agent takes an action in an environment and receives a reward and a new state.[SB18]

where $\mathbb{E}_{t,S,q}[\cdot]$ is the expectation conditional on $S_t = S$ and $Q_t = q$. The running inventory penalty is not an execution cost, rather it introduces the ability to incorporate the urgency for executing the trade, where a higher ϕ yields a quicker strategy.

One can obtain the optimal execution via the *dynamic programming principle* (DPP) and the related *Hamilton-Jacobi-Bellman* (HJB) equation. DPP allows the stochastic control problem to be solved from the terminal date and iterate backwards, see for instance [CJP15]. An optimal execution strategy for a trading agent using only MOs to optimally sell q_0 shares between $t = 0$ and T, utilizing both both temporary- and permanent linear market impact and without short-term alpha, $\alpha_t = 0$. This can be acheived by minimizing the trading agent's execution cost is equivalent to maximizing the expected revenues from the target sale of the q_0 units. This leads to the trading agent's value function:

$$H(t,S,q) = \sup_{\nu\in A} \mathbb{E}_{t,S,q}\Big[X_T^\nu + Q_T^\nu(S_T^\nu - \varphi Q_T^\nu) + \phi\int_t^T (Q_u^\nu)^2\,du\Big],$$

where A is the set of admissible strategies. [CJP15] shows that the optimal solution is given by

$$\nu_t^* = \gamma\frac{\zeta e^{\gamma(T-t)} + e^{-\gamma(T-t)}}{\zeta e^{\gamma T} - e^{-\gamma T}} q_0, \qquad \text{where } \gamma = \sqrt{\frac{\phi}{k}}, \quad \zeta = \frac{\alpha - \frac{1}{2}b + \sqrt{k\phi}}{\alpha - \frac{1}{2}b - \sqrt{k\phi}},$$

We can see that if some parameters have extreme values the relationship between the rest becomes simpler,

$$\nu_t^* \xrightarrow{\phi\to 0} \frac{q_0}{T + \frac{k}{\varphi}}, \qquad \nu_t^* \xrightarrow[\varphi\to\infty]{} \gamma\frac{\cosh\big(\gamma(T-t)\big)}{\sinh\big(\gamma(T-t)\big)} q_0, \qquad \nu_t^* \xrightarrow[\varphi\to\infty]{\phi\to 0} \frac{q_0}{T}. \tag{2.5}$$

In particular, if the termination penalty α is large ($\alpha \to \infty$) and the running penalty ϕ is small ($\phi \to 0$) which tells that the trading agent's optimal trading rate ν_t is constant for all t, this strategy is known as the *time-weighted average price* (TWAP). We see that the more things that are taken into account the more complex the solutions becomes. We also observe that if the trading agent get a large penalty for having units left at the termination with no penalty for any running inventory that TWAP is the optimal strategy. With this knowledge and the fact that TWAP is easy to understand and commonly used in the industry which leads to the choice of using TWAP to be the baseline strategy of choice in this report. This gives us a key advantage when training our RL-algorithm in environments where TWAP is optimal, in those environments the RL-algorithm should perform equally as well as TWAP and thus optimally.

3. DEEP REINFORCEMENT LEARNING

In this section we introduce some fundamental Deep Reinforcement Learning and introduce the two powerful architectures, the value-based DDQN and the policy-based PPO. For basic introduction to RL in quantitative investment, we refer the reader to chapter 13 in [Gui18].

3.1. **Reinforcement Learning.**

Dynamic Programming with the equivalent HJB formulation can be used to compute optimal policies for under simplified assumption. But in practice, dynamic programming is limited due to the assumption of knowledge of the true MDP distribution and its expensive computation. The area of *Reinforcement Learning* enables for solving control problems in a *model-free* setting, where it learns a policy by interacting with an environment directly, to maximize a cumulative long-term reward. Unlike dynamic programming the RL-algorithms is characterized by having an *agent* in *state* S_t, that iteratively interact with an *environment* in discrete time steps $t \in \{1,2,...,T\}$ by taking an *action* A_t and thus moving to a new state S_{t+1} and receiving *reward* r_{t+1}, see for intance Figure 2. The goal is to learn the optimal *policy* π^*, i.e., which action A_t to execute given the current state space S_t, for a given environment to maximize, not the immediate reward R_t, but the total cumulative reward R_t, formally known as *the reward hypothesis*:

$$R_t \quad = \quad r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Given a policy π, the value of being in a state $s \in \mathscr{S}$ is given by the *state-value function* $V_\pi(s)$, the expected total reward from then onward and the value of executing an action a while being in a state s is given by the *action-value function* $Q_\pi(s,a)$:

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| S_t = s\right], \forall s \in \mathscr{S}, \tag{3.1}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| S_t = s,\ A_t = a\right], \tag{3.2}$$

where the discount factor $\gamma \in [0,1]$ is a weight to the future expected rewards. When γ are close to 0 the future expected rewards are less important leading to a *"myopic"* evaluation, for γ close to 1, one obtains a more *"far-sighted"* evaluation, making future expected rewards more important, see [SB18, ADBB17].

Both the state-value function $V_\pi(s)$ and the action-value function $Q_\pi(s,a)$ are driven by a policy π at state s. A policy is a mapping from states to probabilities for selecting each possible action. An *optimal policy* is the policy equal to or greater to all other policies, where corresponding optimal state-value function is given by $V_\pi^*(s,a) = \max_\pi V_\pi(s,a)$ and optimal action-value function is given by $Q_\pi^*(s,a) = \max_\pi Q_\pi(s,a)$. This learning can be performed using either via *value-based* or *policy-based* methods. The value-based attempts to learn the action-value function with a policy to greedily select the action with the highest action-value, whereas the policy-based RL attempts to learn the optimal policy directly.

3.2. **Value-Based Methods.**

3.2.1. *Q-learning.* Q-learning [WD92], a tabular approach, is an *off-policy* temporal difference algorithm that use behavioural policy (e.g. ε-greedy) when exploring the environment to estimate a target policy, resulting in Q-learning directly approximating Q_*. This enables early convergence proofs since all that is required for convergence is that all state-action pairs continue to be updated. Q-learning is often solved using the following:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \eta(R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)), \tag{3.3}$$

where η is a step-size parameter, and each S_t and A_t pair with its corresponding estimated action-value $Q(S_t,A_t)$ is stored in a table, with a row for each state and column for each action.

3.2.2. *Deep Q-network.* Recently there have been extensively research in powerful function approximator utilizing *artificial neural networks* to solve for the optimal action-value function $Q_*(s,a)$, and the optimal policy π_*. [MKS$^+$13, MKS$^+$15, vHGS15, SLA$^+$15, SWD$^+$17, HMvH$^+$17, Sil16]. Similarly to regular Q-learning, a Deep Q-network (DQN) approximates the action-value function for state action pairs but instead of utilizing the tabular approach it uses neural networks. A strength of DQN is its ability to represent high-dimensional spaces to approximate of the action-value function $Q(s,a)$ compared to the tabular approach. A drawback of DQN when utilizing neural networks in RL is that it is rather unstable. However, DQN can be modified by including experience replay and a target network to mitigate these issues, see [ADBB17]. Experience replay is storing the agent's experience in a replay memory. It requires both a model-free and off-policy algorithm, thus it works well with Q-learning. At each time step, batches are randomly sampled from the replay memory to update the Q-value. This yields smoother gradients when updating the parameters of the NN while also removing correlations in the observation sequence, see [MKS$^+$15, ADBB17, SB18]. Another source of instability is when the target is a function of the same weights that are being updated. The idea behind target and Q-network is to make the RL-algorithm behave more as a supervised learning algorithm where the target does not depend on the weights that are being updated. This is done by creating a target-network which the Q-network train towards, whilst still being able to bootstrap. The target-network initially contains the weights of the Q-network who are kept frozen for a predefined number of time steps until they are updated again to the new weights of the Q-network. The process is repeated until training termination. [SB18, MKS$^+$15, ADBB17]

3.2.3. *Deep Double Q-network.* The problem with DQN is that it sometimes overestimates the action values due to the introduced positive bias from taking the maximum action value as an approximation for the maximum of the expected action value. These overestimations are both common and harmful to performance but can be prevented [vHGS15]. Instead of having one estimator, as in DQN, to both determine the maximizing action A_{t+1}^* and its estimated value $Q(S_{t+1},A_{t+1}^*)$, DDQN uses two NN, Q_1 and Q_2, to decouple the process. The two estimators alternate in determining A_{t+1}^* and the $Q(S_{t+1},A*_{t+1})$, e.g., when Q_1 is used to determine the maximizing action $A_{t+1}^* = \text{argmax}_a Q_1(a)$ then Q_2 estimates its action value $Q_2(S_{t+1},A_{t+1}^*) = Q_2(S_{t+1},\text{argmax}_a Q_1(S_{t+1},a))$. After each time step only the weights of one estimator is updated. One downside using double estimators is that it doubles the memory requirement and tends to underestimate the maximum expected action value function; which tend to yield better performance in some situations. One of these situations is when there is a state s with the true value $Q(s,a) = 0$, but where the estimated values are uncertain and, therefore, some of the estimated values $Q(s,a) \leq 0$ and the others are $Q(s,a) \geq 0$. The maximum of the true values are 0, but the maximum from the estimated are positive [SB18]. The described situation is similar to a stochastic price simulation in Equation 2.1, where assuming that permanent price impact is zero, the expected change in price is zero, but in any one price simulation the price is uncertain and might be greater to or less than zero. Furthermore, DDQN has been shown to perform well in optimized trade execution on historical data [NLJ18].

3.2.4. *Exploration vs. Exploitation.* The key in RL-training is the trade-off between *exploration* and *exploitation.* Exploration concerns taking actions with aim of gathering more data to potentially improve the policy, these actions may involve short-term sacrifices and exploitation refers to taking an action from the best policy so far. The dilemma arises in that neither exploration or exploitation can be pursued exclusively to succeed at learning the optimal policy. There do however exist strategies to balance the trade-off between exploration and exploitation, see[SB18]. One approach to handle the trade-off between exploration and exploitation is known as the ε-greedy strategy. The ε-greedy strategy is alternating between greedily taking the best action from the action-value function with taking a random action:

$$a_t = \begin{cases} \text{argmax}_a Q_t(a, s_t), & \text{with probability } 1-\varepsilon, \\ \text{random action } a, & \text{with probability } \varepsilon, \end{cases}$$

where the parameter $\varepsilon \in (0, 1)$. More applicable variants of ε-greedy have decaying ε values over time which anneals the initial ε value to a smaller number $\varepsilon^{'}$. This results in large exploration at the start but as it learns it exploits that knowledge more over time [SB18].

3.3. **Policy-Based Methods.** Instead of a value-based approach it is also possible find a parameterized optimal policy that can maximize the long-term reward without the requirement of a value function (however PPO described later includes this though), these methods are called policy-based and are more suitable for problems with continuous action spaces. *Policy gradient methods* are learning the policy parameter θ based on the gradient of some scalar performance measure $L(\theta)$ seeking to maximize long-term reward. These methods often start with an objective

$$L^{PG}(\theta) = \mathbb{E}\left[\log \pi_\theta(a_t|S_t)\hat{A}_t\right],$$

where π_θ is the stochastic policy and $\hat{A}_t$ the some advantage function estimator. There exists a family of policy gradient methods for estimate the corresponding gradient. Starting with simple "vanilla" gradient methods such as REINFORCE [Wil92].

Using vanilla method, such as the Monte Carlo REINFORCE, that directly optimize the policy without utilizing the value function, is often not suitable as these methods often have very high variance and therefore can lead to large updates in the policies, where a single bad step can collapse the policy performance More recent, has been on so called *actor-critic* methods, that consists of two components. A *critic* component that measures how good the action taken is (value-based) an *actor* component that controls how well the agent behaves (policy-based). The idea is that instead of waiting until the end an episode as in Monte Carlo REINFORCE, one can perform an update at each step, using temporal difference learning. Trust region policy optimization (TRPO) is one such method that try to reduce the variance by determine the maximum exploration step size and then find the optimal point within that *trust region.* This is done by penalize large policy updates as well as including a constraint on the KL-divergence between the new and the old policy which then can be solved by the conjugate gradient algorithm. The policy is updated by approximately solving this optimization problem each iteration. It enabled monotonic updates that guarantees small policy improvement as long as it is optimized within the trusted region, see e.g., [SLA⁺15]. However, it scales poorly and requires a large batch of rollouts for accurate approximation, see [WMG⁺17].

Recall that, Q is the total reward from taking action a in state s_t and V is the total reward from s_t These actor-critic methods defines an advantage function:

$$\hat{A}_t = \hat{Q}_{\pi_\theta}(a_t, s_t) - \hat{V}_\theta(s_t),$$

that tells us how much better action a_t is in state s_t. For T steps the advantage function can be specified as:

$$\hat{A}_t^{(T)} = r_t + \gamma r_{t+1} + \ldots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t}\hat{V}_\theta(s_{t+T}) - \hat{V}_\theta(s_t),$$

where $\hat{V}_\theta(s)$ is the learning state variable. In short it is the empirical returns plus estimation of the last step minus the value function baseline of step t. The advantage function is large if the observed returns r_k moving onward from s for T steps are larger the estimated value of being in the state s.

An online policy gradient methods known as *Proximal Policy Optimization* (PPO) is based on similar ideas as TRPO, it is more general, have better sample complexity and thends to be more data efficient and give rise to more reliable performance, see [SWD⁺17]

Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, so $r_t(\theta_{old}) = 1$. In TRPO on would maximize the objective $L^{TRPO}(\theta) = \mathbb{E}_t\left[r_t(\theta)\hat{A}_t\right]$ where the trust region constraint prohibits excessively large policy updates. PPO first modifies the objective to instead penalize changes to the policy that move $r_t(\theta)$ away from 1. This through the new objective

$$L^{CLIP}(\theta) = \mathbb{E}_t\big[\min(\hat{A}_t r_t(\theta), \hat{A}_t \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\big],$$

where ϵ is a positive constant. Looking at the terms in the minimization the first one is the unclipped $L^{TRPO}(\theta)$ objective while the second clips the probability ratio. Clipping removes the incentive to move r_t outside the interval $\big[1-\epsilon, 1+\epsilon\big]$ which constraints large updates. Whereas the minimum on the unclipped and clipped objective puts a pessimistic lower bound of the final objective on the unclipped objective. Thus, clipping is only active when the objective is getting worse. When computing the advantage function estimators make use of a learning state-value function $\hat{v}(s)$ are hen included in the loss, this is important especially when using a neural network that shares parameters between the policy and value function. This is done by including the loss

$$L_t^{VF}(\theta) = \mathbb{E}_\pi\big[(G_t - \hat{V}_\theta(s_t)\big],$$

The objective can further be augmented to ensure sufficient exploration, this through an entropy bonus $S[\pi_\theta](s_t)$, which then states the PPO objective function:

$$L_t^{PPO}(\theta) = \mathbb{E}\Big[L_t^{CLIP} - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\Big],$$

where c_1 and c_2 are regulating coefficients, and the objective is then approximately maximized each iteration. The PPO Algorithm is defined in Algorithm 2.

3.4. **Optimization.** Tuning of non-linear policies such as neural networks are historically viewed as unstable in training policy-based methods [BM95]. The two main challenges are the large number of data required and unstable improvement of the policy despite the non-stationary data [SML+15]. The first challenge is tackled by utilizing the *GAE* in [SML+15]. Whereas recent research has attempted at stabilizing the algorithms [Rie05, SLA+15, VHGS16, MBM+16, WMG+17]. These approaches generally work around the non-stationarity of the observed data see citemnih2016asynchronous. Some methods store the data in an experience replay memory and batching data [BM95, SLA+15] or randomly sampling from different steps [VHGS16]. These methods that aggregate over memory reduces non-stationary and decorrelates the data, however it limits the methods to off-policy learning. Another method is to utilize asynchronous execution of multiple agents in parallel on multiple instances of the environment. This parallelism also decorrelates the data but in a more stationary process which in turn enables on-policy RL-algorithms, see [MBM+16].

NN are normally trained using iterative gradient-based optimization using *backward propagation* to minimize the cost function. There are currently many optimization algorithms that perform well on a non-convex optimization problem and there is no consensus on what optimization algorithm to use, instead, it is recommended to use the one that is most familiar with the user. Adaptive moment estimation (Adam) is an optimization algorithm with first-order gradient-based optimization algorithm of stochastic objective functions, based on adaptive estimates of lower-order moments. It is appropriate for non-stationary objectives with noisy and/or sparse gradients, and problems with large data sets and many parameters. The method computes individual adaptive learning rates from estimates of the exponential moving averages gradient s and from the squared gradient gradients r with $\rho_1, \rho_2 \in [0,1)$ who control the exponential decay rates of the moving averages. The moving averages are biased towards zero, especially during the initial time steps. This bias is counteracted with bias-corrected estimates $\hat{s}$ and $\hat{r}$, see [KB14]. Adam tends to yield better performance with the PPO-model [SWD+17] and the DQN [HMvH+17].

The performance of a NN is dependent on many factors such as the number of hidden layers, number of nodes per layer, learning rate in the loss function, and more. One simple method for optimization of hyperparameters is *grid search*. Grid search is an exhaustive search of all the hyperparameters in form of a Cartesian product, which can quickly lead to an exploding search space. Smart hyperparameter optimization methods have been developed based of Bayesian learning, genetic algorithms, early stopping, etc. showing promising results of decreased time for hyperparameter tuning. Such methods include *Median Stopping Rule, Google2017stopping , Population Based Training* [JDO+17] and finally *AsyncHyperband* which is a distributed algorithm for smart early stopping [LJR+18].

3.5. **Challenges in RL.** There are many challenges when working with RL, see for instance [ADBB17, MKS+15, KLM96]. Often low performance compared to domain-specific algorithms. The reward design is difficult, making the RL-algorithm do exactly what you want is not always trivial. Even with a good reward, it can be easy to find local optima and get stuck in it. When the RL works, it might be because it has over fitted to the specific problem. This makes it hard to generalize and perform well in other tasks. It learns through trial-and-error interactions with the environment. Actions taken now can cause delayed rewards. When neural networks are used as function approximators then reinforcement learning becomes unstable due to: The existing correlation in the sequence of observations, small updates to Q may change the policy and therefore change the data distribution and supervised learning has a fixed dataset compared to RL which receives correlations between the action-values Q and the target values $r_{t+1} + \gamma \max_a Q(S_{t+1}, a)$.

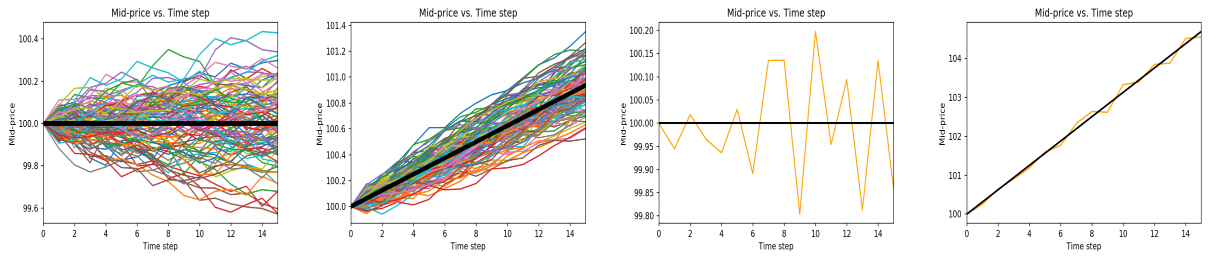


FIGURE 3. **Simulation Environments**. 1: Environment 1, 100 paths with no drift ($\mu = 0$). 2: Environment 2, 100 paths with low drift ($\mu = 0.01$) 3: Environment 3, 1 path of a large and small mean price reversion around $S = 100$. 4: Environment 4, 1 path of a large mean reversion ($\epsilon = 0.05$) around a price with high drift ($\mu = 0.05$).

The financial markets can be seen as partially observable markov process with not having enough information about the market and the the distribution of the data can change, making it difficult for a DRL-algorithm to take action.

4. REINFORCEMENT LEARNING FOR OPTIMAL EXEUTION

In this section we first present the different environments for which our RL agents act on, the corresponding hyper parameters.

4.1. **Environments.**

Before trying to beat the baseline strategy, we first made sure that DRL can converge to that strategy in an environment where the baseline strategy is the optimal strategy. We consider 4 different environments to iteratively train our DRL-algorithm on:

(1) *Environment 1*: Stochastic mid-price movement with temporary price impact. This is the most basic environment where the TWAP is the optimal strategy, as showed in equation (2.5). The goal of the DRL-algorithm is to converge to TWAP. We set the volatility of the asset to 10 times larger than the other environments in an attempt to challenge the convergence capability even further. A representation of the environment can be seen in Figure 3.

(2) *Environment 2*: Stochastic mid-price movement with both temporary price impact and permanent price impact and different drifts. A more complex controlled environment where TWAP is the optimal strategy under no drift, as seen in equation (2.5). In [CJP15] they typically observe the relationship $b \ll k$, where k is the temporary price impact and b is the permanent price impact. In our setting $b < 2k$ which is when TWAP is the optimal strategy, see Appendix C. Hence, if the price movement is purely martingale with 0 drift ($\mu = 0$) the DRL-algorithm's goal is once again to converge to TWAP. However, we also tested whether the agent could beat TWAP under a mid-price movement with two positive mid-price percentage drift ($\mu > 0$). Therefore, we tested $\mu = [0, 0.01, 0.05]$. A representation of the environment can be seen in Figure 3.

(3) *Environment 3*: Stochastic mid-price movement with temporary price impact, permanent price impact and mean-reversion ϵ. Two ϵ values were used to create an environment with large mean reversion $\epsilon = 0.05$ and on with small mean reversion $\epsilon = 0.0005$ An informed trader can find a dynamic strategy where TWAP is no longer the optimal strategy and, therefore, the DRL-algorithm's goal is to perform better than TWAP by being an informed trader. More input features were added to help the DRL-algorithm predict future price movement. A representation of the environment can be seen in Figure 3.

(4) *Environment 4*: Stochastic mid-price movement with temporary price impact, permanent price impact, large drift and high variance mean-reversion. This can be seen as the most extreme environment with the highest drift from Environment 2 and the largest mean reversion movement from Environment 3. A representation of the environment can be seen in Figure 3.

4.2. **Training Process.**

4.2.1. *Performance Evaluation.* The benchmarking is performed against the TWAP, a static strategy that is decided before trading and where the trading rate is given by:

$$v_t^{TWAP} = \frac{q_0}{T},$$

where q_0 is the initial inventory and T is the trading horizon. TWAP is the optimal for the first two environments when $\mu = 0$ and the agent's goal is thus to reach the same performance. Whereas for environment 3 and 4 the agent's goal is to beat TWAP. The performance of the RL algorithms and baseline strategies are evaluated by a cost. This cost is evaluated on the unrealistic scenario of selling all volume q_0 at the initial mid-price s_0 of the start of the trading period.

Secondly of the terminal cash of the agent from, described in Equation (2.3), which is the cash generated during the trading period. Which in discrete time is defined as:

$$X_T = \sum_{t=0}^{T} \hat{s}_t \nu_t,$$

where $\hat{s}_t$ is the execution price. Based on the performance criteria we wished to compare the performance of the agent and the benchmark. It was calculated as follows:

$$I_{BPS} = 10,000 \times \frac{X_T^{Agent} - X_T^{TWAP}}{X_T^{TWAP}}, \tag{4.1}$$

where X_T is the terminal cash for the different strategies. This yields a basis point (BPS) performance improvement to the baseline strategy TWAP in terms of cash earned. For hyperparameter tuning the configuration that yielded the highest improvement I_{BPS} was considered best.

4.2.2. *Hyperparameter tuning.* All Neural networks have two fully connected layers with the ReLU as activation function. However, to enable the models, PPO and DDQN, to converge in Environment 1 with a reasonable speed we performed a search over which hyperparameters to use for the models. In Environment 2 a new hyperparameter search was performed to find better hyperparameters based on the information gathered in hyperparameter searching for Environment 1. The hyperparameters for Environment 1 can be seen in Table 10. The hyperparameter search is done using AsyncHyperband for smart early stopping. The hyperparameters found in Environment 2 was then used for Environments 3 and 4. For each environment the models are trained in the same way to give consistency.

(1) *Hyperparameter search*: Trained each parameter combination on 2^{15} episodes, taking the mean improvement to benchmark I_{BPS}, as seen in equation 4.1, from six different seeds to increase the robustness of the results due to DRL-algorithms being unstable.
(2) *Train*: Trained on the parameter combination that gave the highest mean improvement to benchmark I_{BPS} from the previous step by training on 2^{16} episodes and using 16 different seeds to increase the robustness of the mean results.
(3) *Evaluation*: Once the training is completed, the metrics are evaluated, visualized and compared to the performance of TWAP as to identify if the DRL-algorithm has found TWAP, which is optimal in environment 1 & 2 (when there is no drift), or performed better then TWAP in environment 2 (when there is a drift), 3 & 4.

4.2.3. *State Space.* The state variables of the environment observable by the agent are referred to as *private state variables* and *market state variables* with some limitations. The private state variables are the elapsed time $t_n \in [0, T]$, where T is the trading time horizon and remaining volume in inventory $q_n \in (0, q_0]$, where q_0 is the initial volume in inventory The market state variables are the mid-price s_n, order flow imbalance $ofi_n = M_n^+ - M_n^-$ and the price return $r_n^{\%} = \frac{s_n}{s_{n-1}}$. For further details see Appendix B.

4.2.4. *Action Space.* If the trading agent is to trade a large volume the state and action space becomes large. To tackle this issue we introduced *lots* of assets with lot size ℓ. Similarly, the agent's chosen trading rate ν is the number of lots to trade instead than number of assets to trade. Therefore, the only action is the number of units MOs the agent would like to trade, which is limited by the number of units it has in inventory, $\nu_n \in [0, 2\frac{q_0}{T}\frac{1}{\ell}], \nu_T = q_T$: the number of lots with lot size ℓ to trade in a market order. We also add a limitation that the agent is only allowed to sell twice as large volume as TWAP to speed up convergence and constrain unrealistic strategies. Also note the constraint that the agent must sell all remaining inventory at the terminal time step T.

4.2.5. *Reward.* Based on the performance evaluation, which we wish to minimize, the reward is defined as a continuous scalar of the difference between the cash generated from the trade at the current step and the cash that would have been generated if the volume would have been sold at the starting price instead:

$$r_n = \hat{s}_n \nu_n - s_0 \nu_n,$$

where s_0 is the initial mid-price and $\hat{s}_n$ the execution price. Other parameters can be added to the reward function, a common one being risk [BL98, DBK$^+$17, SFSK18]. In our case we chose to not include it so the implementation would stay close to [CJP15].

4.2.6. *Input features.* For Environment 1 and 2 only elapsed time (t_n) and remaining inventory (q_n), i.e. the private state variables, were as inputs. For Environment 3 the market state variables were used to enable informed trading, similarly to hyperparameter searching above we searched which input state variables were valuable. Environment 3 also included the short-term alpha variable α, which gives the movement for the mean reversion, when searching for the best input combination as a reference point for what values to expect when the DRL-algorithm has full knowledge of the price movement; although, it was not used when training Environment 3 as it would give perfect information of the price movement, something that is seen as inplausible to achieve in the real markets. For more details on the inputs see Appendix D. The data was also preprocess, see Appendix B.

4.3. **Implementation.** The simulation environment was developed utilizing OpenAI's Gym API as it is a commonly used standard in RL research and testing [BCP+16]. It has a large community and is, therefore, a common requirement for most RL libraries. The RLlib library, based on Ray, was used to tune the hyperparameters and train the different RL-algorithms. Where Ray is a distributed system for AI-applications [MNW+17]. We used ADAM for training the DDQN, but SGD (a variant of ADAM) for PPO as it is not supported by Ray yet. Google Cloud Compute Engines were used for large scale hyperparameter tuning and training. With limited amount of episodes the random draws Z_t were generated by a T dimensional Generalized Halton Sequence [FL09].

5. Numerical Results

In this section we demonstrated the strategies of the two different models acting on four different environments. The resulting hyperparameters search are used in the final training of the agent on same environment.

5.1. **Setup.** For ease of replicating and extending our study we use the parameters in [CJP15], as shown in appendix E.

5.2. **Environment 1.** Environment 1, the simplest setup, with only temporary price impact but with 10 times large volatility compared to the other environments to ensure that conversion to TWAP can be achieved with added noise. The optimal strategy for Environment 1 is TWAP, where the strategy is static and thus determined before trading to be the same at each time step ($v_n = \frac{q_0}{T}$). Figure 4a illustrates the remaining inventory and Figure 4b the average action taken per time step for TWAP.

5.2.1. *DDQN.* The best hyperparameters for DDQN are presented in Table 1 and the corresponding training results presented in table 2. Given that the improvement I_{BPS} is negative, it can be seen that the DDQN agent does not converge to the optimal TWAP. Observing the mean episode reward in Figure 6a we see that the agent does not fully converge to the theoretical optimal average episode reward. One can argue that that this will increasing over time and it has therefor not fully reached its highest potential value. To get a better understanding of the agent trading we observe the average inventory over time in Figure 5a. Compared to TWAP, it decays in similar speed but seemingly not at a constant speed. A detailed depiction of the agent's average execution strategy is showed in Figure 5c. Here it becomes more apparent as to why agent's average strategy is not constant. When comparing to TWAP we can see the agent does not use the same strategy as TWAP, however, the actions v seemingly cluster around the actions taken by TWAP.

5.2.2. *PPO.* The best hyperparameters for PPO are presented in Table 1, and the corresponding training results are presented in table 2. We can see that the PPO agent's average strategy does not fully converge to the optimal strategy TWAP. Observing PPOs mean episode reward in Figure 6b we see that the agent tends to converge rather fast to the theoretical optimal reward, as shown by the dashed line. However, some runs diverge from optimum over time. Observing the PPO agent's average inventory in Figure 5b it becomes apparent that it is very near that of TWAP, see Figure 4a, with a nearly constant decay rate of inventory. When we finally compare the strategy used at each time step in 5d to that of TWAP we can clearly see that the PPO agent has a very similar strategy. One can even argue that by a vote of majority it has fully converged to TWAP since the usage of the TWAP strategy is a majority. Looking into the results of each individual training one can even find cases where agents uses a strategy fully converged at TWAP, see $\max \overline{v}_{agent}$, $\overline{v}_{agent}$ and $\min \overline{v}_{agent}$ of the trained PPO agents of different seeds in Appendix D.

5.3. **Environment 2.** Environment 2 has both temporary and permanent price impact, but a lower volatility than Environment 1 and with three different drifts $\mu = [0, 0.01, 0.05]$, recall that TWAP is optimal when Environment 2 has no drift ($\mu = 0$).

5.3.1. *DDQN.* The best hyperparameters are presented in Table 4 and these are used to train the DDQN agent on three different versions drifts. The inventory of the different drifts can be seen in Figure 7a. For the different drifts:

- $\mu = 0$: Looking at the improvement I_{BPS} in Table 4 we can see that it is close to converging to the execution strategy of TWAP. The inventory decays at near constant rate, similar to TWAP. From the volume traded at each time step we can see in Figure 7c that it has not fully converged to TWAP, however its strategy is closer to that of TWAP as compared to Environment 1.

- $\mu = 0.01$: With a positive drift the improvement I_{BPS} is positive in Table 4. Thus, DDQN found a strategy that outperforms that of TWAP's. The inventory at each time step in in Figure 7a decays non-linearly, were the agent saves inventory for later time steps as to gradually sell more at a more favourable price, which can be seen in Figure 7e. At the last time step it utilizes, to its advantage, the fact that our simulation forces the agent to sell its entire remaining inventory ($v_T = q_T$) which on average is ~ 20 lots. The agent initially sells 0 lots, it then starts selling at around time step ~ 3 where it gradually increase the amount it sells. When it approximately reaches time step 12 it starts selling the maximum allowed amount of 15 lots until it reaches the final time step where it is forced to sell the entire remaining inventory.
- $\mu = 0.05$: Similarly to $\mu = 0.01$ the DDQN agent has a positive improvement I_{BPS}. The improvement is much larger as it is able to find an execution strategy that is able capitalize on the large drift. We can see in Figure 7g that the agent saves all inventory until time step ~ 9, where it starts selling at a drastically increasing rate. Similar to $\mu = 0.01$ it utilizes, to its advantage, the fact that during the last time step our simulation forces the agent to sell its entire remaining inventory ($v_T = q_T$) which on average is ~ 50 lots. The agent sells 0 lots during the first ~ 7 time steps. It then reaches a critical point where it starts selling the maximum allowed amount of 15 lots until it reaches the final time step where it is forced to sell of all remaining inventory.

5.3.2. *PPO.* The best hyperparameters, presented in Table 3 were used to train the PPO agent on Environment 2 with the different drifts. The inventory of the different drifts can be seen in Figure 7b. For the different drifts:

- $\mu = 0$: Looking at the improvement I_{BPS} in Table 4, and Figure 7d we can see that it is close to converging to the execution strategy of TWAP. We see that the remaining inventory at each time step decays at a slower rate than TWAP, given it has ~ 18 lots left at the terminal time step.
- $\mu = 0.01$: With a positive drift the improvement I_{BPS} is positive in table 4. Thus, PPO found a strategy that outperforms that of TWAP's. In Figure 7f we observe that the agent saves inventory for later time steps to gradually sell more as the price increases. Looking at the terminal time step we can see that the agent utilizes the constraint of being forced to sell all remaining inventory ($v_T = q_T$) to its advantage. It, therefore, sells ~ 40 lots at the terminal time step. The agent sells more over time. Initially it sells 0 lots until it reaches ~ 3 time step where it increases the amount it sells over time until it reaches ~ 11 time step where it sells the maximum allowed lots per time step until termination. At termination it utilizes, to its advantage, the fact that the simulation forces it to sell all remaining lots of inventory.
- $\mu = 0.05$: Similarly to $\mu = 0.01$ the PPO agent has a positive improvement I_{BPS}. The improvement is much larger as it is able to find an execution strategy that is able to capitalize on the large drift. In Figure 7h we observe that the agent saves inventory until a critical point where it starts selling at a drastically increasing rate. It also utilizes, to its advantage, the constraint of being forced to sell all remaining inventory ($v_T = q_T$) during the last time step to sell ~ 70 lots. The agent initially sells 0 lots until reaching ~ 10 time step where it drastically increases its trading rate until it reaches ~ 11 time step where it sells the maximum allowed lots per time step. At termination it utilizes, to its advantage, the fact that the simulation forces it to sell all remaining lots of inventory.

5.4. **Environment 3.**

Environment 3 has both temporary and permanent price impact, low variance and two different mean reversion variances $\epsilon = 0.0005, 0.05$. The hyperparameter search for Environment 3 is over the inputs that helps predict the mean reversion movement. The best resulting hyperparameters are presented in Table 5. For a complete list of the parameter search results see Appendix D.

5.4.1. *DDQN.* From the hyperparameter search for the small mean reversion movement with $\epsilon = 0.0005$ we did not see any indication of change in execution strategy that could outperform TWAP. Primarily due to the low improvement but also the execution strategy trend depicted in Table 5. However, for a larger mean reversion movement ($\epsilon = 0.05$) the improvement I_{BPS} is positive and deemed more interesting to explore further. When observing the complete results from hyperparameters search we concluded that even though excluding OFI performed best for DDQN the large improvement of including OFI in PPO justified including it as a state variable when training DDQN. Especially since the difference between including and excluding OFI is so low for DDQN. This enables us to fairly compare the two models later on. The training results for the large mean reversion movement can be viewed in Table 5. With a positive improvement I_{BPS} we can see that the agent outperforms TWAP and has thus learned to capitalize on the mean reversion movement to earn more reward. When observing the agent's inventory over time in Figure 8a we can see that on average, even under a large mean reverting movement, the price decays in a rather linear matter. To observe the agent's execution strategy and how the new inputs affect it:

- **Action vs. Time step:** illustrated in Figure 9a. During the initial time step ~7 lots are sold. During the remaining time steps there is a higher concentration of ~ 0 and ~ 15 lots sold. This is most likely due to the mean reverting properties and non-existing drift, making it lucrative to sell when the price is high and do nothing when the price is low. An interesting observation is that during the terminal time step there is a higher probability it will sell of ~ 0 lots than the maximum amount. This might be due to the simulation forcing the agent to sell all remaining inventory terminal inventory and, therefore, it understands that it does not need to act itself as it gets help.

- **Action vs. Mid-price:** illustrated in Figure 9c. We observe that the more the mid-price moves from its initial mid-price, the more certain the agent becomes on what action to take. The higher the price goes, the more certain the agent becomes on selling, and the lower the price the more certain the agent becomes on not selling, After the mid-price has moved enough to one side the agent comes close to ~ 80-90% certain on what action to take.
- **Action vs. Order Flow Imbalance:** illustrated in Figure 9e. We observe that agent has found a slight pattern to how the price moves due to the OFI. The greater the OFI are the more certain the algorithm becomes on how to act, with the agent selling less when there is a higher OFI and more when it is lower. A more detailed representation is shown in Figure 9e.
- **Action vs. Price Return:** illustrated in Figure 9g. The results are close to identical as Figure 9c.

5.4.2. *PPO.* From the hyperparameter search, there was no indication that the agent could outperform TWAP when the environment had a small mean reversion movement ($\epsilon = 0.0005$). However, for a larger mean reversion movement ($\epsilon = 0.05$) the improvement I_{BPS} is positive and is deemed more interesting to explore further. The training results for the large mean reversion movement have a positive improvement I_{BPS} and the agent outperforms TWAP and has thus learned to capitalize on the mean reversion movement to earn more reward. When observing the agent's inventory over time in Figure 8b we can see that on average, even under a large mean reverting movement, the price decays in a rather linear matter. To observe the agent's execution strategy and how the new inputs affect it:

- **Action vs. Time step:** illustrated in Figure 9b. During the initial time step ~7 lots are sold. During the remaining time steps there is a high concentration of ~ 0 and ~ 15 lots sold.
- **Action vs Mid-price:** illustrated in Figure 9d. This figure shows that the more the mid-price moves from its initial mid-price, the more certain the agent becomes on what action to take. The higher the price goes, the more certain the agent becomes on selling, and the lower the price the more certain the agent becomes on not selling, After the mid-price has moved enough to one side the agent comes close to ~ 80-90% certain on what action to take.
- **Action vs. Order Flow Imbalance:** illustrated in Figure 9f. The figure shows that agent has found a pattern to how the price moves from the ofi. The greater the ofi the more certain the algorithm becomes on how to act, with the agent selling less when there is a higher ofi and more when the it is lower. A more detailed representation is shown in Figure 10b.
- **Action vs. Price Return:** illustrated in Figure 9h. The results are close to identical as Figure 9d.

5.5. **Environment 4.** Environment 4 has temporary and permanent price impact, low variance, large drift ($\mu = 0.05$) and a large mean reversion variance ($\epsilon = 0.05$). No hyperparameter search was performed for Environment 4, instead the best architecture for the hyperparameter search on Environment 2 was used and with the best inputs from Environment 3.

5.5.1. *DDQN.* Observing the performance in Table 7 we see that the DDQN agent drastically outperformed TWAP with a large improvement I_{BPS}. Looking at the remaining inventory over time in Figure 11a we see that the agent trades more at later time steps. It also utilizes the terminal condition $v_T = q_T$ to sell a large quantity at the terminal time step. The agent's realised execution strategy, depicted for each time step action pair in Figure 11c, reveals that the agent tends to sell 0 at the initial time steps. Over time it starts selling larger quantities v but still sometimes selling 0.

5.5.2. *PPO.* We can see in Table 7 that PPO outclassed TWAP in Environment 4 with a large improvement I_{BPS} value. From observing the remaining inventory over time for PPO on Environment 4 in Figure 11b we can see that the agent, on average, increases its trading rate over time. It utilizes the terminal constraint $v_T = q_T$ to sell a large quantity of ~ 50 lots at the terminal time step. The detailed depiction of the agent's execution strategy in Figure 11d shows that the agent is very likely to sell 0 lots at the initial time step while gradually decreasing the probability to sell 0 while increasing the probability to sell 15 lots.

5.6. **Numerical Results Summary.** In environment 1 both agents seem to be able to identify the portion of the action space of which the optimal strategy exists. On average, they trade at a similar rate as TWAP, whereas PPO has an easier time to converge at the globally optimal execution strategy.

In environment 2 similarly to 1, PPO had better convergence to the global optimum under no drift ($\mu = 0$) than DDQN. One may argue that DDQN had a better convergence in Environment 2 than Environment 1, when comparing its execution strategy, which may be the result of Environment 2 having a lower volatility or that DDQN had better hyperparameters. More interestingly, we also learned that both agents were able to beat TWAP by capitalizing the price trend moving upwards on a price drifting upwards. We also learned that limiting the action space of the agent for faster convergence may limit it from finding better strategies.

In environment 3: firstly, we learned that DRL can be used to make informed decisions by utilizing a dynamic strategy that outperforms TWAP. Secondly, we learned that an agent can only be an informed trader if it has information available, but is still able to find a reasonable strategy, similar to TWAP, when there is not enough information available. Finally, we learned that the discrete action space with fixed lot size limited the agent from outperforming TWAP when the reversion movement was small.

In Environment 4 we learned that the agents are able to combine different knowledge, both the long term trend and mean reversion, to greatly outperform TWAP.

6. Conclusion

In this article we trained two DRL agents, using two popular approaches, the value based DDQN and the policy-based PPO, to tackle the optimal execution problem by trading against the popular market benchmark, the TWAP. We showed that the PPO was able to converge to the baseline strategy TWAP in the environments where TWAP was optimal whereas DDQN seemed to either be unable to converge or required longer training time. However, both agents were able to outperform TWAP in environments where TWAP was not optimal, such as with drift, mean reversion and a combination of the two. In combination with these the agent's performances are a lower bound to the optimal behaviour due to factors such as hyperparameter tuning and training time are not exhaustive, thus better performing agents can be identified. We show that: firstly, that deep reinforcement learning can reach the theoretically derived optimum through acting on the environment directly. Secondly, that the agents can learn to capitalize on long term price trends without directly observing the price. Finally, that they can exploit available information to create dynamic strategies as an informed trader and thus outperform static strategies such as TWAP.

Given that all environments were relatively simple, although powerful, it is natural to explore more complex configurations. Similarly, to TWAP being the theoretically optimal execution strategy under a simple environment configuration it would be interesting to see if these DRL agents can converge to optimal strategies in more complex environments, but also when interacting on real-life trading environments. Another extension would be to explore the scalability of a DRL-algorithm in optimized trade execution by increasing the complexity of the problem formulation, e.g., having a larger action space allowing the agent to also post LOs at different levels in the LOB to save the "spread" and minimizing the frequency of crossing in the spread and "walking-the-book". Adding more "venues", as stocks typically are traded at several venues simultaneously could also be one direction, to learn venue specific characteristics, as no venues are the other like. Also, other benchmarks such at the volume-weighted average price (VWAP) would be an additional benchmark, since the average traded volume through the day is not constant.

To conclude, the key with deep execution, i.e. DRL for optimal execution, is that we leave the task of defining the feature specification and optimization to the algorithm, instead of having it being "human programmed" as with traditional algorithmic trading methods where one in order to obtain an optimal value has to base assumptions on the underlying price in a physical way, and often restrict it to the Markovian case, and don't utilize past memory, trading knowledge, reasoning, intuition and so forth. For a successful trader this is key. One can, therefore, not rely on a uniform physical model, as psychology also plays a big part. Since we as humans are Darwinian's, artificial intelligence will play a major role in the future of financial investing and execution. With cheaper access to data storage and computational power, we have already seen what DRL, in its infant stage, are capable of and With the state of the art implementations used for DeepMind's AlphaGo and OpenAI's Dota2, shows a bright future.

Disclaimer: There are no conflicts of interest exist, the views expressed here are those of the authors and do not necessarily reflect the position of their employer.

TABLE 1. Environment 1, best performing hyperparameters

	η	λ	ϵ	M	Minibatch	Hidden Layers	$\bar{G}_{\text{agent}}$	$\bar{G}_{\text{TWAP}}$	I_{BPS}
DDQN	0.00005			32		(20×6)	-0.0210	-0.0189	-1.6570
PPO	0.00005	0.9	0.3		256	(20×6)	-0.0186	-0.0186	-0.0416

TABLE 2. Environment 1, the training results.

	$\bar{G}_{\text{agent}}$	$\bar{G}_{\text{TWAP}}$	I_{BPS}
DDQN	-0.0204	-0.0186	-1.3857
PPO	-0.0189	-0.0187	-0.1448

TABLE 3. Environment 2, best performing hyperparameters ($\mu = 0$)

	η	λ	ϵ	M	Minibatch	Hidden Layers	$\bar{G}_{\text{agent}}$	$\bar{G}_{\text{TWAP}}$	I_{BPS}
DDQN	0.00001			32		16×8	-0.0306	-0.0293	-0.9803
PPO	0.00005	0.9	0.3		256	16×8	-0.0293	-0.0291	-0.1281

TABLE 4. Environment 2, training results using different drifts μ

μ		$\bar{G}_{\text{agent}}$	$\bar{G}_{\text{TWAP}}$	I_{BPS}
0.05	DDQN	0.4368	0.2638	135.5690
	PPO	0.4372	0.2638	135.8601
0.01	DDQN	0.0460	0.0294	13.2445
	PPO	0.0461	0.0294	13.3342
0.00	DDQN	-0.0295	-0.0292	-0.2877
	PPO	-0.0293	-0.0292	-0.0778

TABLE 5. Environment 3, best performing inputs

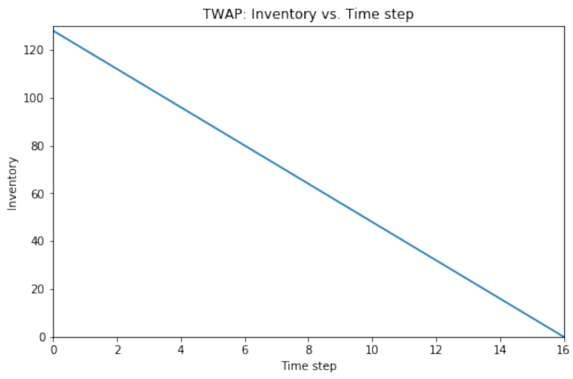
	ϵ	α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
DDQN	0.0005	False	False	False	True	-0.0296	-0.0292	-0.2959
PPO	0.0005	False	True	False	True	-0.0293	-0.0292	-0.0805
DDQN	0.05	False	True	False	True	0.0180	-0.0297	38.2431
PPO	0.05	False	True	True	True	0.0232	-0.0292	42.0496

TABLE 6. Environment 3, training results ($\epsilon = 0.05$)

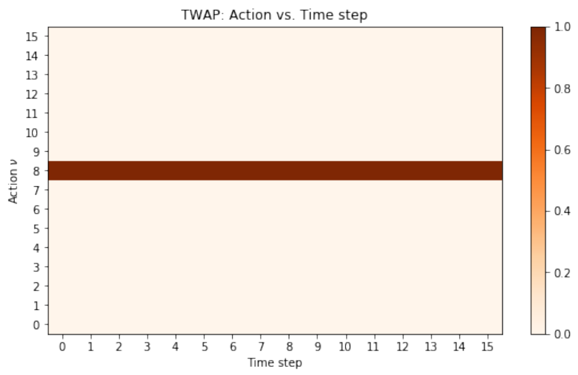
	α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
DDQN	False	True	True	True	0.0173	-0.0295	37.4862
PPO	False	True	True	True	0.0228	-0.0292	41.6968

TABLE 7. Environment 4, training results

	α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
DDQN	False	True	True	True	0.4197	0.2641	121.9571
PPO	False	True	True	True	0.4499	0.2638	145.8177

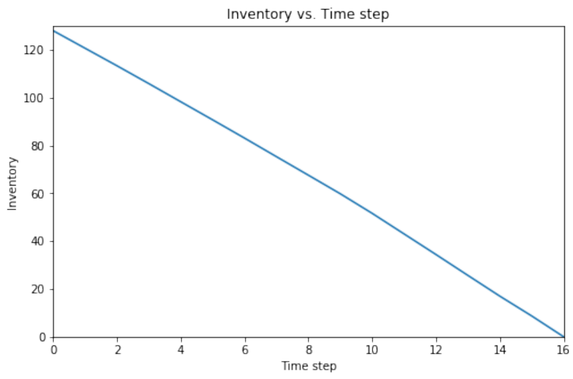


(A) Inventory per time step

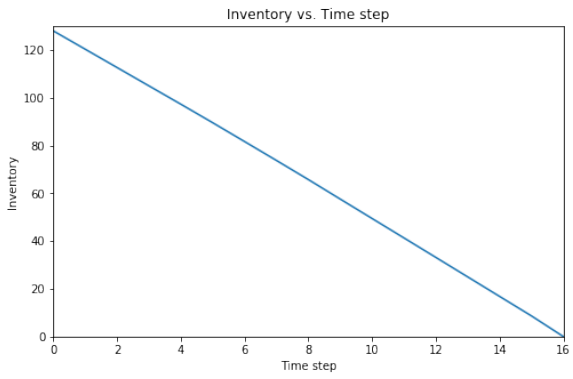


(B) Action taken per time step

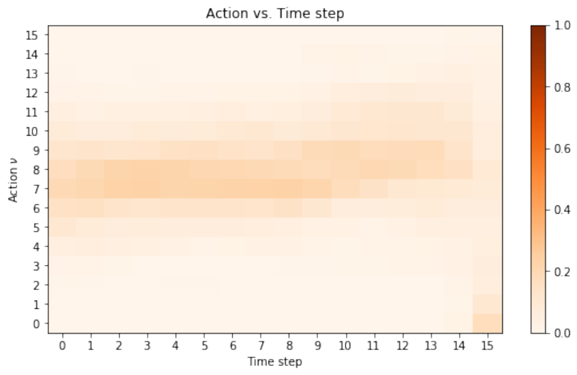
FIGURE 4. TWAP benchmark



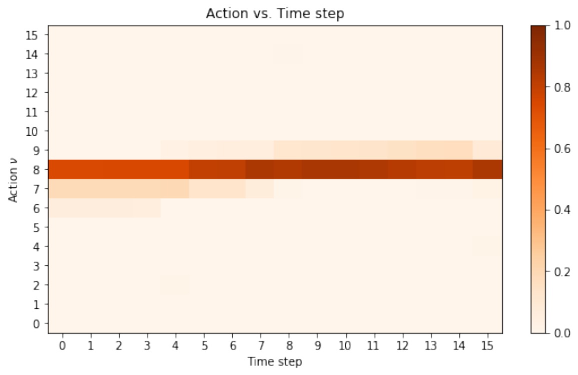
(A) Remaining inventory for DDQN



(B) Remaining inventory for PPO

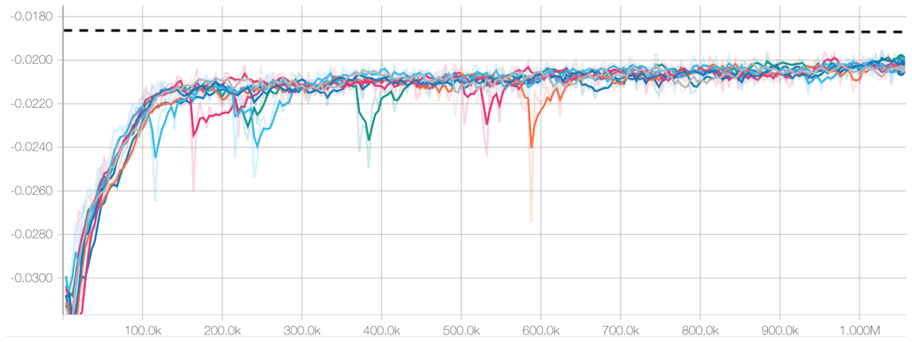


(C) Average action taken per timestep for DDQN

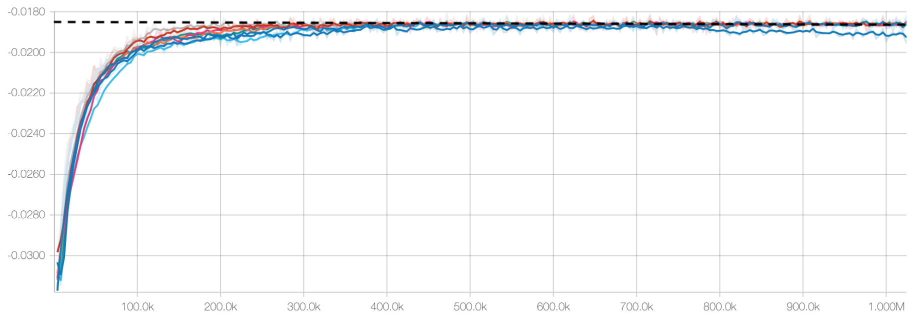


(D) Average action taken per timestep for PPO

FIGURE 5. Environment 1

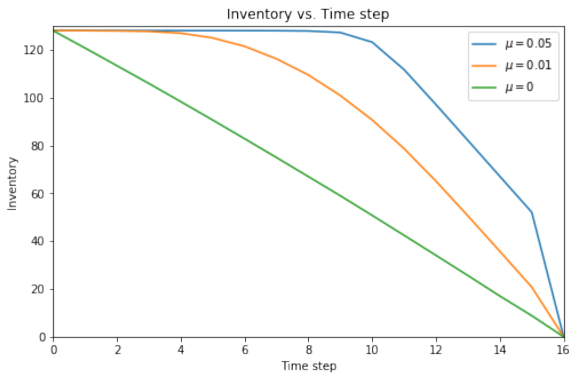


(A) DDQN

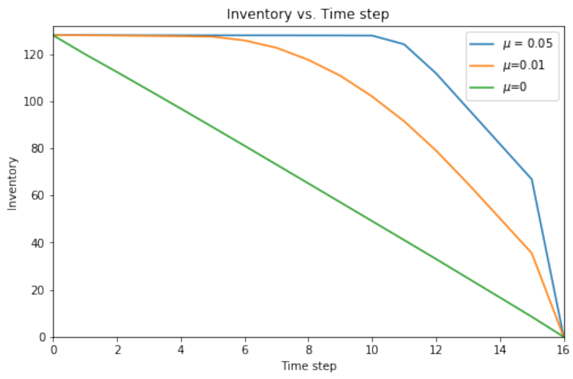


(B) PPO

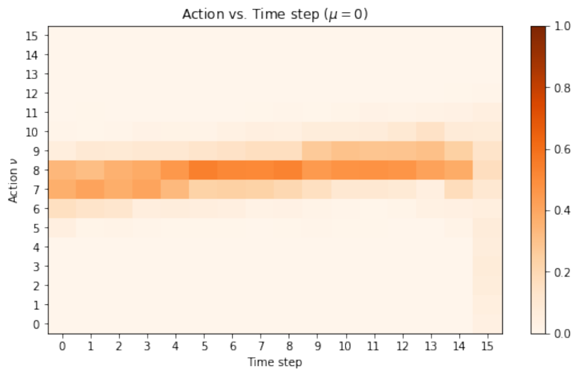
FIGURE 6. Reward convergence environment 1



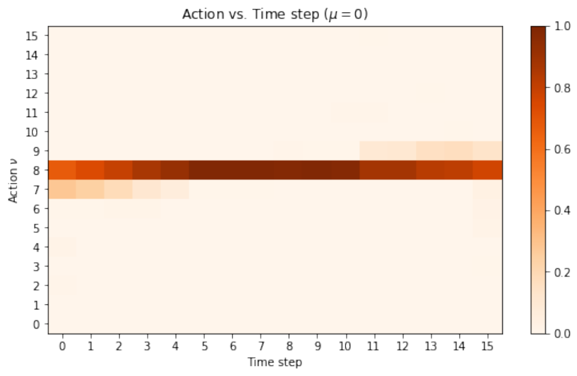
(A) Remaining inventory for DDQN



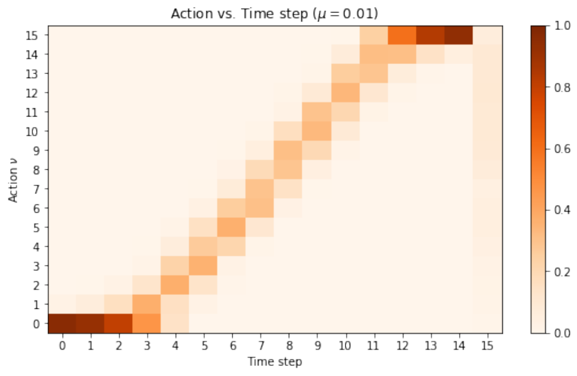
(B) Remaining inventory for PPO



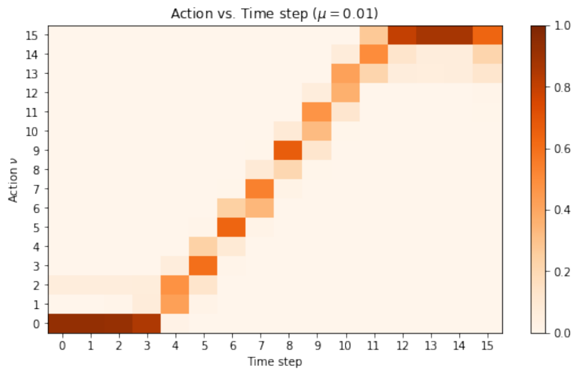
(C) Average action taken per timestep for DDQN $\mu = 0$



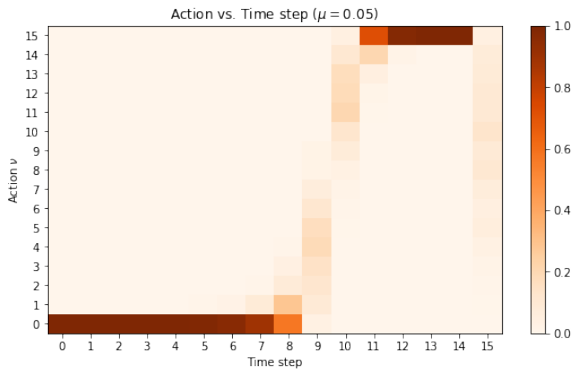
(D) Average action taken per timestep for PPO $\mu = 0$



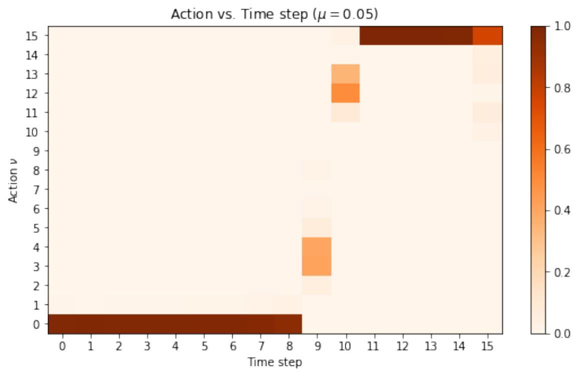
(E) Average action taken per timestep for DDQN $\mu = 0.01$



(F) Average action taken per timestep for PPO $\mu = 0.01$

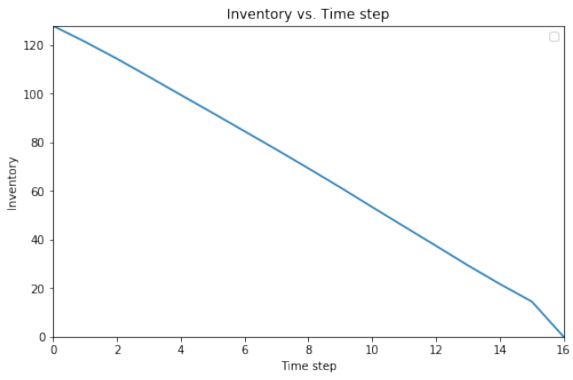


(G) Average action taken per timestep for DDQN $\mu = 0.05$

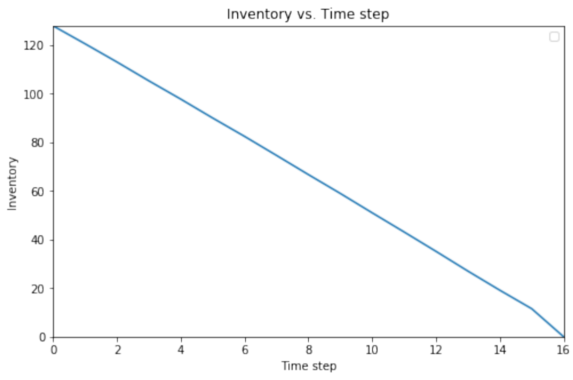


(H) Average action taken per timestep for PPO $\mu = 0.05$

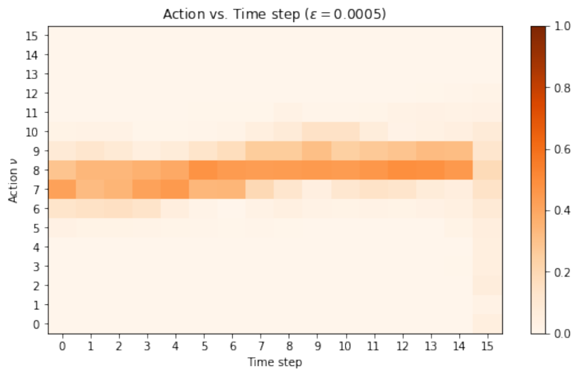
FIGURE 7. Environment 2



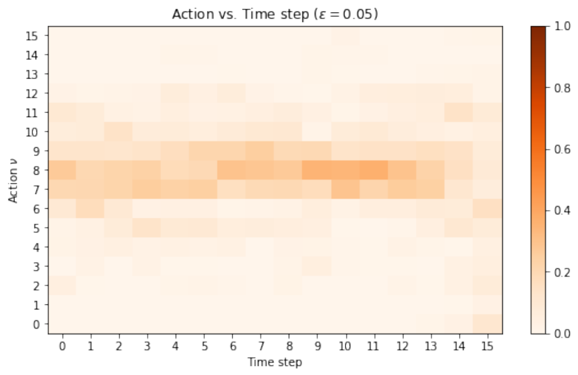
(A) Remaining inventory for DDQN



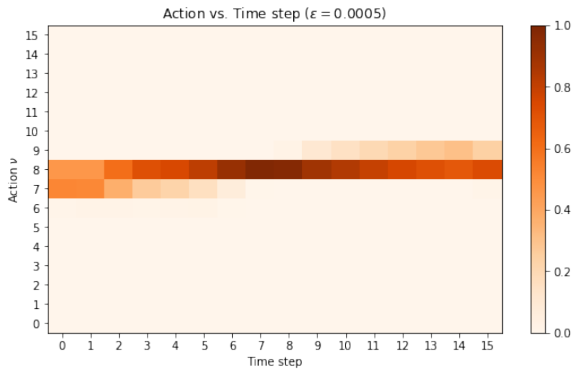
(B) Remaining inventory for PPO



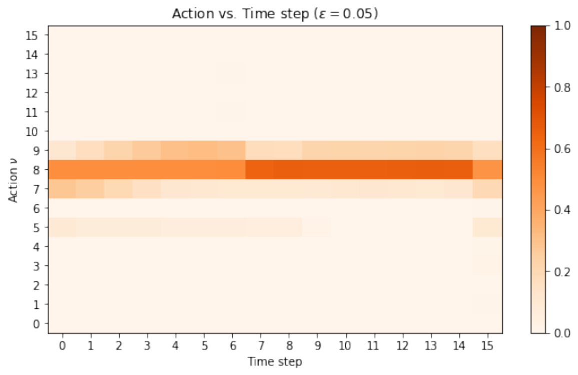
(C) Average action taken per timestep for DDQN $\epsilon = 0.0005$



(D) Average action taken per timestep for PPO $\epsilon = 0.0005$

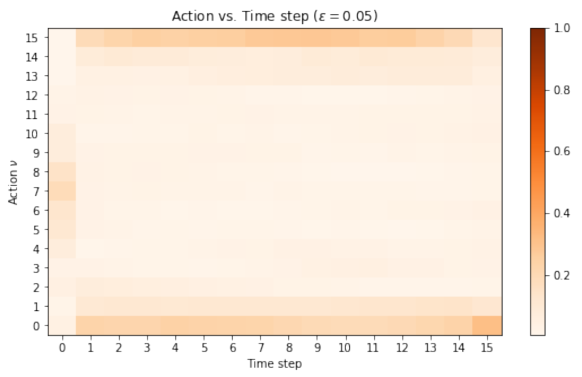


(E) Average action taken per timestep for DDQN $\epsilon = 0.05$

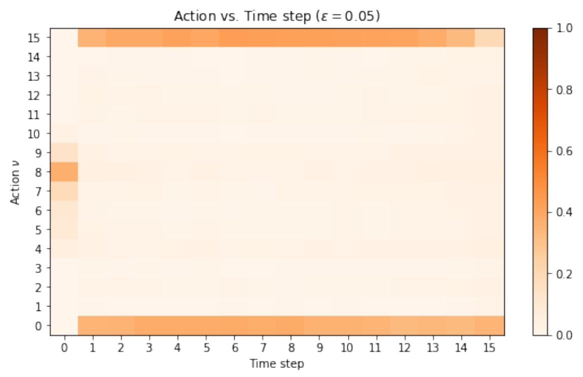


(F) Average action taken per timestep for PPO $\epsilon = 0.05$

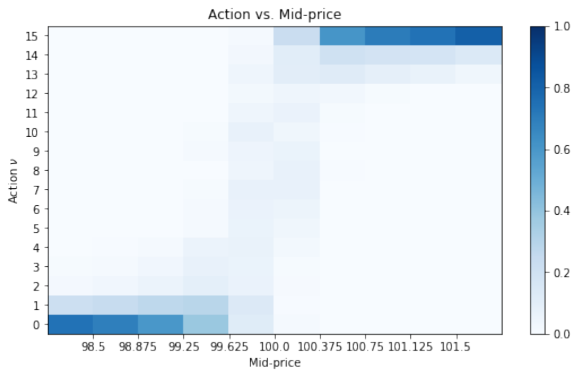
FIGURE 8. Environment 3



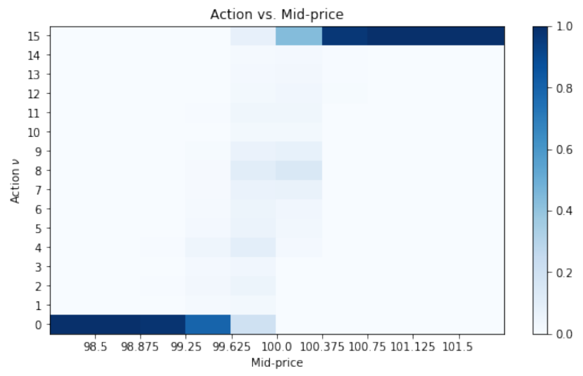
(A) Action taken given time step for DDQN



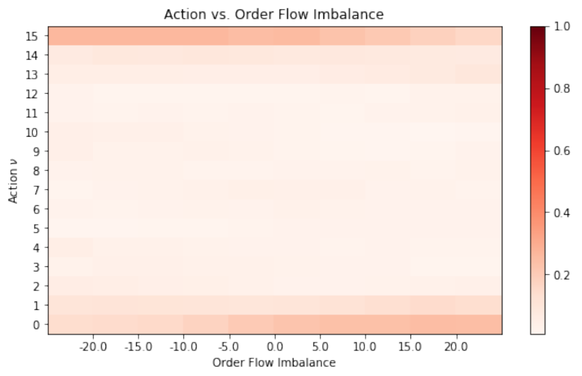
(B) Action taken given time step PPO



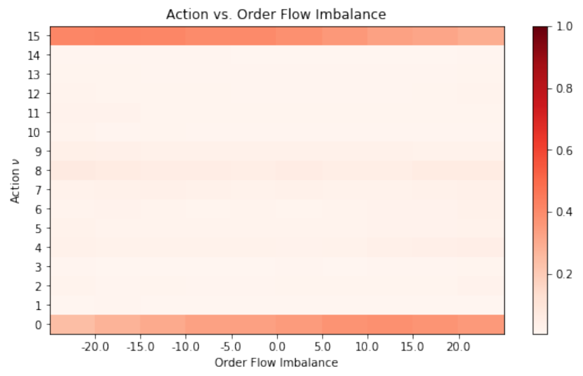
(C) Action taken given mid-price for DDQN



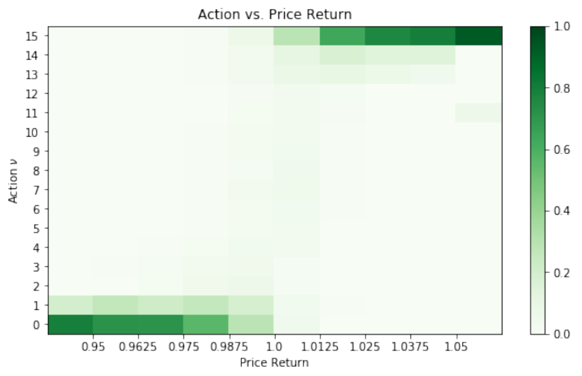
(D) Action taken given mid-price for PPO



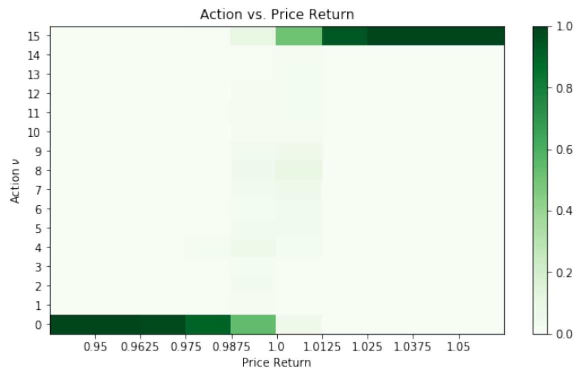
(E) Action taken given order flow imbalance for DDQN



(F) Action taken given order flow imbalancep for PPO

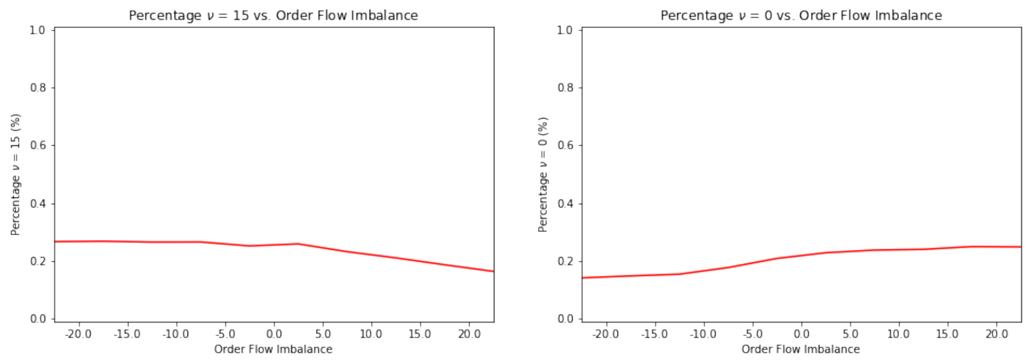


(G) Action taken given price return for DDQN

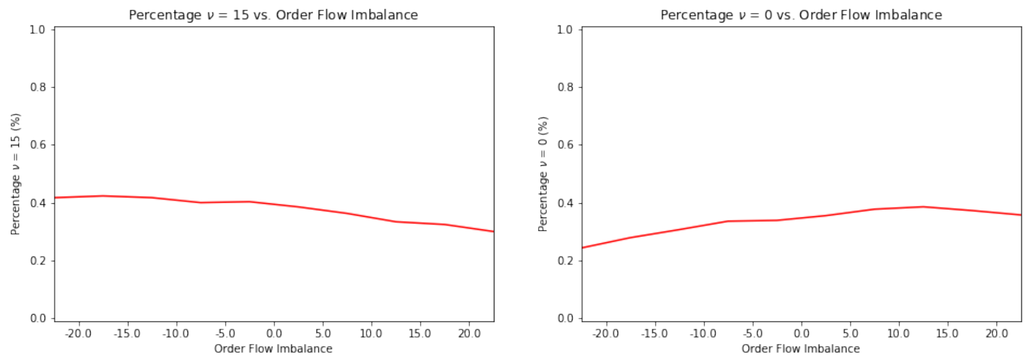


(H) Action taken given price return for PPO

FIGURE 9. Environment 3

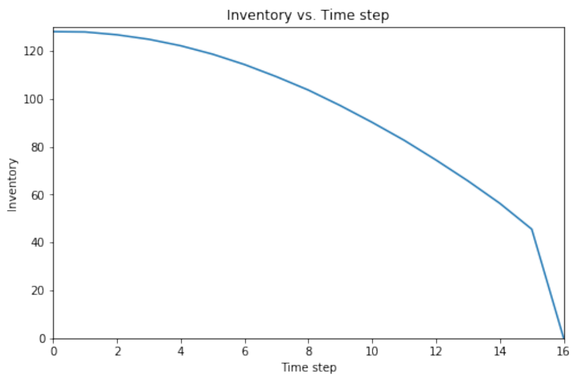


(A) DDQN

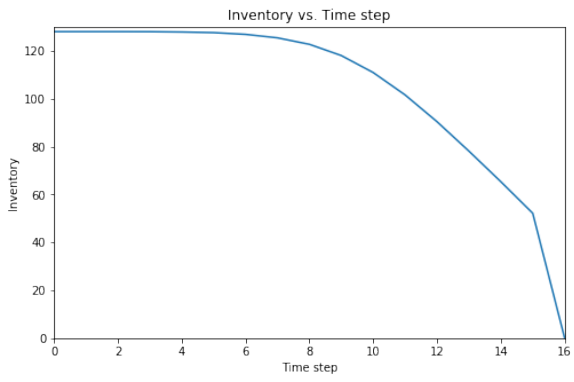


(B) PPO

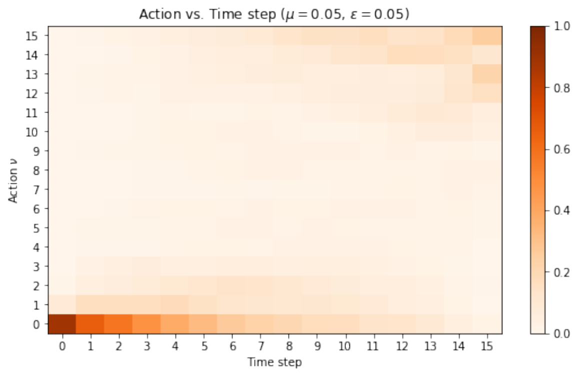
FIGURE 10. Environment 3. Percentage of trading 15 and 0 lots based on OFI ($\epsilon = 0.05$).



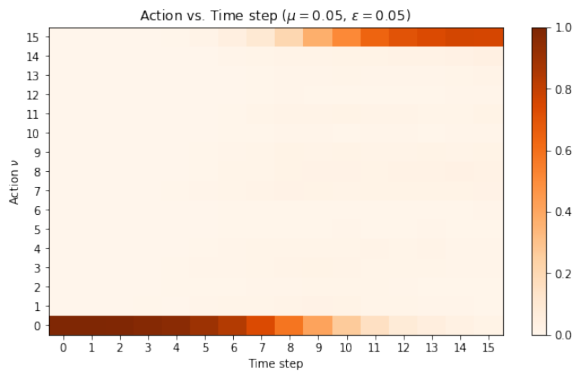
(A) Remaining inventory for DDQN



(B) Remaining inventory for PPO



(C) Average action taken per timestep for DDQN



(D) Average action taken per timestep for PPO

FIGURE 11. Environment 4

Algorithm 1 DDQN

```
Require:
  $\eta \in (0,1]$: Learning rate.
  $\varepsilon \in (0,1]$: exploration rate
  Initialize weights in $Q_1(s,a)$ and $Q_2(s,a)$
  while stopping criteria not met do
      Choose an action $A$ from $S$ based on $Q_1$ and $Q_2$
      Take action $A$, observe $R$, $S_{t+1}$
      Choose UPDATE(A) or UPDATE(B) randomly
      if UPDATE(A) then
          $A^*_{t+1} = \text{argmax}_a Q_1(S_{t+1}, a)$
          $Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \eta(R_{t+1} + \gamma Q_2(S_{t+1}, A^*_{t+1}) - Q_1(S_t, A_t))$
      else if UPDATE(B) then
          $A^*_{t+1} = \text{argmax}_a Q_2(S_{t+1}, a)$
          $Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \eta(R_{t+1} + \gamma Q_1(S_{t+1}, A^*_{t+1}) - Q_2(S_t, A_t))$
      end if
  end while
```

Algorithm 2 PPO Training Algorithm

```
Require: Number of iterations $I$, actors $N$, steps $T$, minibatch size $M$, epochs $K$ and clipping size $\epsilon$
  for iteration in $1 \dots I$ do
      for actor in $1 \dots N$ do
          Run policy $\pi_{\theta_{old}}$ in environment for $T$ steps
          Compute advantage estimates $\hat{A}_1 \dots \hat{A}_T$
      end for
      for epoch in $1 \dots K$ do
          Optimize objective $L^{PPO}$ w.r.t. $\theta$ with minibatch size $M \le NT$
      end for
      $\theta_{old} = \theta$
  end for
```

APPENDIX B. PREPROCESSING

In an attempt to give the different models' neural networks parameters of equal scaling the inputs were transformed to enter the range [-1, 1]. The transformed values where then passed on as state in place of the inital values. Similarly the reward was preprocessed to give similar scalar values for different initial inventory q_0 and total time steps T.

The elapsed time was transformed as follows:

$$t_n^{'} = 2\frac{t_n}{T} - 1,$$

where $t_n = 0$ at the initial time step and $t_n = T$ at the terminal time step.

The agent's inventory was transformed as follows:

$$q_n^{'} = 2\frac{q_n}{q_0} - 1,$$

where $q_n = q_0$ is the initial inventory and when $q_n = 0$ the trade execution is complete.

The stock's mid-price was transformed as follows:

$$s_n^{'} = c\frac{s_n - s_0}{\sigma\sqrt{\Delta t(T-1)}}, \qquad \text{where } c = \frac{1}{2.575},$$

where s_0 is the initial mid-price, Δt is the time interval and T is the terminal time. Scaling with c causes $s_T^{'}$, which has the highest variance, to have a 99% probability to remain in the interval [-1,1]. To see the derivation of the transformation see Appendix B.2.

Recall that order flow imbalance is:

$$ofi_n = M_n^+ - M_n^-,$$

where $M_n^{\pm}$ is the arrival rate buy (sell) market orders, simulated using a Poisson distribution ($M_n^{\pm} \sim Po(\lambda^{\pm})$). Using a normal approximation assumption of ofi_n with $\lambda^{\pm} \geq 15$ ($ofi_n \sim Po(\lambda^{\pm}) - Po(\lambda^{\pm}) \approx N(0, \sqrt{2\lambda^{\pm}})$) we can thus transform order flow imbalance as follows:

$$ofi_n^{'} = c\frac{ofi_n - 0}{\sqrt{2\lambda^{\pm}}}, \qquad \text{where } c = \frac{1}{2.575}.$$

Similar to transformation of mid-price, scaling with c causes $ofi_n^{'}$ to have a 99% probability to remain in the interval [-1,1].

Recall that price return is:

$$r_n^{\%} = \frac{s_n}{s_{n-1}}.$$

It was preprocessed as follows:

$$r_n^{\%'} = r_n^{\%} - 1,$$

to have its move its mean around 0 instead of 1 to ensure its within the interval [-1, 1] with a high probability.

B.1. **Reward.** To enable training of the different models for different problem configurations, as in different starting inventories q_0 and number of time steps T, the reward was scaled to give similar values for the different settings if using similar strategies. The scaling was:

$$r_n^{'} = \frac{T}{q_0^2} r_n,$$

where q_0 is the initial inventory and T is the total number of time steps. To see the derivation of the scaling see Appendix B.3

B.2. **Mid-price transformation.** The transformation is derived from the stochastic variable S_T, which is the mid-price at the termination time step, since its variance is the highest given its a sum of all stochastic variables S_t up till T. Recall the stochastic variable S_t:

$$S_t = S_{t-1} + \sigma\sqrt{\Delta t} Z_t^S, \qquad \text{where } S_0 = s_0.$$

Normalization of S_T is performed through subtraction of its mean and division with its standard deviation:

$$\begin{aligned}
S_T &= s_0 + \sigma\sqrt{\Delta t}\sum_{t=1}^{T-1} Z_t, \qquad \text{where } Z_t = N(0,1),\\
E(S_T) &= E(\sigma\sqrt{\Delta t}\sum_{t_1}^{T-1} Z_t) = s_0,\\
V(S_T) &= V(\sigma\sqrt{\Delta t}\sum_{t_1}^{T-1} Z_t) = \sigma^2 \Delta t \sum_{t=1}^{T-1} V(Z) = \sigma^2 \Delta t (T-1).
\end{aligned}$$

Which in turn enables the following normalization:

$$S_T^{'} = \frac{S_T - s_0}{\sigma\sqrt{\Delta t(T-1)}} \sim N(0,1).$$

However, we wish to transform the variable to most often stay in the interval [-1,1] thus we look for a constant c to scale $S_T^{'}$ with to make that probable:

$$\begin{aligned}
P\Big(-1 \leq cS_T^{'} \leq 1\Big) &= 0.99,\\
P\Big(\frac{-1}{c} \leq S_T^{'} \leq \frac{1}{c}\Big) &= 0.99,
\end{aligned}$$

where $S_T^{'} \sim N(0,1)$ gives us $c = \frac{1}{2.575}$. Which yields the final transformation:

$$s_n^{'} = c\frac{s_n - s_0}{\sigma\sqrt{\Delta t(T-1)}}, \qquad \text{where } c = \frac{1}{2.575}.$$

B.3. **Reward scaling.** Recall that a strategy is the trading rate v_t the agent places at time steps $\{0 \leq t \leq T\}$. This scaling was derived from the scenario that an agent uses TWAP in two different problem configurations:

$$\begin{aligned}
&q_1 \neq q_2,\\
&T_1 \neq T_2,\\
&v_{n_1} = \frac{q_1}{T_1} \text{ and } v_{n_2} = \frac{q_2}{T_2}.
\end{aligned}$$

With the assumption that the price movement has 0 volatility and no permanent market impact ($b = 0$ in $g(v_n)$) the reward per time step is:

$$r_{n_i} = (s_n - kv_n)v_n - s_0 v_n = // \ s_n = s_0 \text{ assuming 0 volatility } // = -kv_n^2 = -k\Big(\frac{q_i}{T_i}\Big)^2,$$

where the agent i uses the TWAP strategy. We want to multiply the rewards yielded per time step with a scaling factor c_i where the agents using same strategies in different problem configurations yield the same total rewards:

$$\begin{aligned}
&c_1 \sum_{n_1=0}^{T_1} r_{n_1} = c_2 \sum_{n_2=0}^{T_2} r_{n_2},\\
&c_1 \sum_{n_1=0}^{T_1} -k\Big(\frac{q_1}{T_1}\Big)^2 = c_2 \sum_{n_2=0}^{T_2} -k\Big(\frac{q_2}{T_2}\Big)^2,\\
&-kc_1 T_1\Big(\frac{q_1}{T_1}\Big)^2 = -kc_2 T_2\Big(\frac{q_2}{T_2}\Big)^2,\\
&c_1 \frac{q_1^2}{T_1} = c_2 \frac{q_2^2}{T_2},\\
&\frac{c_1}{c_2} = \frac{q_2^2/T_2}{q_1^2/T_1},\\
&c_i = \frac{T_i}{q_i^2},
\end{aligned}$$

Thus, we scale the reward with $c = \frac{T}{q_0}$. Which yields the final scaling:

$$r_n' = \frac{T}{q_0^2}\, r_n.$$

Appendix C. Optimal Trade Execution Strategy in Discrete Time

To confirm that TWAP is the optimal solution for Environment 2 as derived in financial background, while also exploring the relationship between k and b for the optimal solution, we will solve the maximization of cash in discrete time. This under the scenario of selling a fixed amount of assets q_0 during a fixed amount of time T. We first observe the mid-price process:

$$S_t = S_{t-1} - bv_{t-1} + \sigma Z_t \sqrt{\Delta t}, \qquad \text{where } S_0 = s_0,$$

where Z_t is independent standard normal distributions $Z_t \sim N(0,1)$. Given that the price process is martingale we assume the volatility of the asset σ is zero and with linear permanent price impact we observe the mid-price process:

$$S_t = S_{t-1} - bv_{t-1} = s_0 - bv_{t-1}.$$

Which in turn yields the execution price:

$$\hat{S}_t = s_0 - bv_{t-1} - kv_t,$$

where v_t is the amount traded. The cash earned from a trade at time step t is then:

$$X_t = \hat{S}_t v_t = (s_0 - bv_{t-1} - kv_t)v_t.$$

We consider minimizing the execution cost through maximizing the terminal cash over the trade horizon T. With the constraint that all q_0 assets must be sold over the time period. We can then set up the optimization problem:

$$\begin{aligned}
\max(f(X)) &= \sum_{t=1}^{T} X_t,\\
\text{subject to } &\sum_{t=1}^{T} v_t = q_0.
\end{aligned}$$

Including the constraint into the objective function through v_T :

$$v_T = q_0 - \sum_{t=1}^{T-1} v_t.$$

Gives the final objective function:

$$\max(f(X)) = \sum_{t=1}^{T-1} X_t + (s_0 - kq_0 - bq_0 + k\sum_{t=1}^{T-1} v_t + b\sum_{t=1}^{T-1} v_t)(q_0 - \sum_{t=1}^{T-1} v_t).$$

With the constraint incorporated into the objective function we can simply calculate the gradient to find the extreme values:

$$\nabla f(X) = \begin{bmatrix} q_0(2k-b) - 2(2k-b)x_1 - (2k-b)\sum_{t\in(1,..,T)\setminus 1} v_t \\ q_0(2k-b) - 2(2k-b)x_2 - (2k-b)\sum_{t\in(1,..,T)\setminus 2} v_t \\ \vdots \\ q_0(2k-b) - 2(2k-b)x_T - (2k-b)\sum_{t\in(1,..,T)\setminus T} v_t \end{bmatrix}$$

To find an extreme value all rows of the gradient $\nabla f(X)$ must be equal to zero, which happens when:

$$v_t^* = \frac{q_0}{T},$$

where v_t^* is the TWAP strategy. Thus, if this is the maximum of the objective function we have found the optimal trading strategy. To find wether this is extreme value is the maximum or minumum we explore the Hessian:

$$H_{f(X)} = \begin{bmatrix} -2(2k-b) & -(2k-b) & \dots & -(2k-b) \\ -(2k-b) & -2(2k-b) & \dots & -(2k-b) \\ \vdots & \vdots & \ddots & \vdots \\ -(2k-b) & -(2k-b) & \dots & -2(2k-b) \end{bmatrix}$$

where if it is positive (semi) definite the extreme value is the target function's minimum and if it is negative (semi) definite it is a maximum. This can be determined by its eigenvalues λ, where all eigenvalues λ_i are:

$$\lambda_i = -(2k-b) = b - 2k,$$

where if all eigenvalues are positive the Hessian matrix $H_{f(X)}$ is positive definite and if they are negative $H_{f(X)}$ is negative definite. In other words if:

- $b < 2k$: $H_{f(X)}$ is negative definite, thus TWAP is maximum and thus the best strategy.
- $b > 2k$: $H_{f(X)}$ is positive definite, thus TWAP is minimum and thus the worst strategy.
- $b = 2k$: If we look at the gradient we can see that all strategies that exists are extreme values, or rather, all strategies are equally good/bad since $f(X)$ is independent of X.

If we observe the case that TWAP is the worst strategy ($b > 2k$) we can see that the optimal strategy is then to sell all inventory q_0 at one time step and 0 at the remaining time steps.

Also, if we consider the case without permanent price impact ($b = 0$) it then fulfills the constraint $b < 2k$ (since k > 0) and thus TWAP is the optimal strategy once again.

Appendix D. Input Features

Table 8. Input features for DDQN and PPO

Category	**Private state**		**Market state**		
Input features	t_n	q_n	s_n	ofi_n	$r_n^{\%}$
Environment 1	x	x	-	-	-
Environment 2	x	x	-	-	-
Environment 3	x	x	x/-	x/-	x/-
Environment 4	x	x	x	x	x

Appendix E. Environment parameters

Table 9. Parameters used in the different simulation environments

Parameter:	**Env. 1:**	**Env. 2:**	**Env. 3:**	**Env. 4:**
Volatility (σ)	8.3×10^{-3}	8.3×10^{-4}	8.3×10^{-4}	8.3×10^{-4}
Drift (μ)	0	[0, 0.01, 0.05]	0	0.05
Temporary price impact (k)	1.86×10^{-3}	1.86×10^{-4}	1.86×10^{-4}	1.86×10^{-4}
Permanent price impact (b)	-	1.41×10^{-3}	1.41×10^{-3}	1.41×10^{-3}
Mean reversion scale (ζ)	-	-	0.5	0.5
Mean reversion deviation (β)	-	-	0.01	0.01
MO arrival rate ($\lambda^{\pm}$)	100	100	100	100
OFI impact ($\epsilon^{\pm}$)	-	-	[0.0005, 0.05]	0.05

TABLE 10. Hyparameters for DDQN

Parameters	**Description**	**Grid Search**
γ	Discount factor	Initial testing 0.99
ε-annealing	Fraction of training period ε is annealed	Initial testing 0.1
ε	Final epsilon after annealing	Initial testing 0.02
Hidden layers	# of neurons per layer × # of layers	[(20 × 6), (32 × 4), (64 × 3), (128 × 2)]
η	Learning rate ADAM	$[10^{-4}, 5\times10^{-4}, 10^{-3}, 5\times10^{-3}, 10^{-2}]$
ρ_1	Decay rate for moment estimate	Suggested default 0.9
ρ_2	Decay rate for moment estimate	Suggested default 0.999
M	Train batch size	[32, 64]

TABLE 11. Hyparameters for PPO

Parameters	**Description**	**Grid Search**
Hidden layers	# of neurons per layer × # of layers	[(20 × 6), (32 × 4), (64 × 3), (128 × 2)]
λ	General advantage estimator	[0.8, 0.9, 1]
η	Learning rate SGD	$[1\times10^{-5}, 5\times10^{-5}, 1\times10^{-4}]$
M	Mini batch size SGD	[128, 256]
c_1	Coefficient of the value function	Iniital testing 1
c_2	Coefficient of the entropy regularizer	Initial testing 0
ϵ	Clip parameter for loss function	[0.1, 0.2, 0.3]

TABLE 12. DDQN Environment 1

η	M	Hidden layers	Time	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{bps}
0.00005	32	[20, 20, 20, 20, 20, 20]	1.00	-0.0210	-0.0189	-1.6570
0.00005	32	[64, 64, 64]	1.00	-0.0217	-0.0187	-2.4046
0.00005	64	[20, 20, 20, 20, 20, 20]	0.87	-0.0216	-0.0185	-2.4631
0.00005	32	[32, 32, 32, 32]	1.00	-0.0216	-0.0184	-2.5148
0.00005	64	[32, 32, 32, 32]	0.87	-0.0224	-0.0189	-2.8328
0.00005	64	[64, 64, 64]	0.73	-0.0227	-0.0188	-3.0846
0.00050	64	[32, 32, 32, 32]	0.87	-0.0230	-0.0183	-3.7652
0.00050	64	[64, 64, 64]	0.87	-0.0237	-0.0189	-3.8550
0.00050	32	[128, 128]	0.87	-0.0236	-0.0187	-3.8561
0.00050	64	[128, 128]	0.47	-0.0237	-0.0188	-3.9977
0.00050	32	[20, 20, 20, 20, 20, 20]	0.60	-0.0241	-0.0186	-4.4301
0.00005	32	[128, 128]	0.73	-0.0243	-0.0187	-4.4797
0.00050	32	[64, 64, 64]	0.73	-0.0249	-0.0189	-4.7500
0.00005	64	[128, 128]	0.47	-0.0260	-0.0186	-5.9295
0.00050	32	[32, 32, 32, 32]	0.73	-0.0277	-0.0190	-6.9935
0.00050	64	[20, 20, 20, 20, 20, 20]	0.47	-0.0293	-0.0189	-8.3371
0.00500	32	[20, 20, 20, 20, 20, 20]	0.20	-0.0306	-0.0193	-9.1075
0.00500	64	[128, 128]	0.47	-0.0314	-0.0185	-10.3140
0.00500	64	[32, 32, 32, 32]	0.87	-0.0320	-0.0188	-10.5631
0.01000	64	[128, 128]	0.33	-0.0325	-0.0187	-11.0148
0.00500	32	[32, 32, 32, 32]	0.73	-0.0341	-0.0187	-12.3912
0.00500	64	[20, 20, 20, 20, 20, 20]	0.33	-0.0349	-0.0190	-12.7489
0.00500	32	[64, 64, 64]	0.20	-0.0366	-0.0187	-14.3474
0.00500	64	[64, 64, 64]	0.33	-0.0365	-0.0184	-14.4796
0.01000	64	[64, 64, 64]	0.20	-0.0373	-0.0186	-14.9630
0.00500	32	[128, 128]	0.20	-0.0377	-0.0188	-15.0929
0.01000	32	[64, 64, 64]	0.47	-0.0377	-0.0187	-15.2212
0.01000	64	[32, 32, 32, 32]	0.33	-0.0373	-0.0183	-15.2719
0.01000	64	[20, 20, 20, 20, 20, 20]	0.47	-0.0382	-0.0190	-15.4112
0.01000	32	[128, 128]	0.47	-0.0386	-0.0190	-15.7123
0.01000	32	[32, 32, 32, 32]	0.20	-0.0441	-0.0191	-19.9988
0.01000	32	[20, 20, 20, 20, 20, 20]	0.33	-0.0466	-0.0187	-22.3314

TABLE 13. DDQN Environment 2, with no drift ($\mu = 0$)

η	M	Hidden layers	Time	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{bps}
0.00001	32	[16, 16, 16, 16, 16, 16, 16, 16]	1.00	-0.0306	-0.0293	-0.9803
0.00001	32	[32, 32, 32, 32, 32, 32]	1.00	-0.0305	-0.0292	-1.0411
0.00001	32	[20, 20, 20, 20, 20, 20]	0.84	-0.0310	-0.0293	-1.3357
0.00001	32	[32, 32, 32, 32]	1.00	-0.0311	-0.0292	-1.5030
0.00005	32	[32, 32, 32, 32, 32, 32]	0.76	-0.0319	-0.0293	-2.0950
0.00005	32	[20, 20, 20, 20, 20, 20]	0.52	-0.0321	-0.0290	-2.5326
0.00001	32	[64, 64, 64]	0.76	-0.0324	-0.0291	-2.6671
0.00005	32	[16, 16, 16, 16, 16, 16, 16, 16]	0.44	-0.0326	-0.0292	-2.7267
0.00010	32	[16, 16, 16, 16, 16, 16, 16, 16]	0.44	-0.0329	-0.0294	-2.7917
0.00005	32	[64, 64, 64]	0.84	-0.0323	-0.0288	-2.7979
0.00010	32	[20, 20, 20, 20, 20, 20]	0.44	-0.0328	-0.0291	-2.9165
0.00005	32	[32, 32, 32, 32]	0.60	-0.0328	-0.0290	-3.0291
0.00010	32	[64, 64, 64]	0.84	-0.0333	-0.0289	-3.5231
0.00010	32	[32, 32, 32, 32, 32, 32]	0.44	-0.0338	-0.0293	-3.6442
0.00010	32	[32, 32, 32, 32]	0.60	-0.0340	-0.0291	-3.9384

TABLE 14. DDQN Environment 3 with small mean reversion ($\epsilon = 0.0005$)

α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
False	False	False	True	-0.0296	-0.0292	-0.2959
True	False	False	False	-0.0296	-0.0292	-0.3327
False	False	False	False	-0.0296	-0.0292	-0.3511
True	False	False	True	-0.0296	-0.0292	-0.3512
False	False	True	False	-0.0296	-0.0292	-0.3591
True	False	True	False	-0.0297	-0.0292	-0.3978
True	False	True	True	-0.0297	-0.0292	-0.4025
False	True	False	False	-0.0297	-0.0292	-0.4041
True	True	True	True	-0.0297	-0.0292	-0.4483
True	True	False	True	-0.0298	-0.0292	-0.4581
False	True	True	True	-0.0298	-0.0292	-0.5185
False	True	False	True	-0.0298	-0.0292	-0.5359
False	True	True	False	-0.0298	-0.0292	-0.5367
True	True	True	False	-0.0299	-0.0292	-0.5800
True	True	False	False	-0.0299	-0.0292	-0.6216
False	False	True	True	-0.0300	-0.0292	-0.6287

TABLE 15. DDQN Environment 3, with large mean reversion ($\epsilon = 0.05$)

α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
False	True	False	True	0.0180	-0.0297	38.2431
False	True	True	True	0.0191	-0.0285	38.1355
False	True	False	False	0.0179	-0.0294	37.9792
False	True	True	False	0.0174	-0.0299	37.9281
True	False	True	True	0.0190	-0.0282	37.8213
True	False	True	False	0.0183	-0.0285	37.5247
True	True	False	False	0.0173	-0.0295	37.5157
True	True	True	False	0.0180	-0.0286	37.3923
True	True	True	True	0.0165	-0.0291	36.5314
True	True	False	True	0.0150	-0.0298	35.9860
True	False	False	True	0.0086	-0.0295	30.5519
False	False	False	True	0.0081	-0.0289	29.6834
False	False	True	True	0.0075	-0.0290	29.2384
True	False	False	False	-0.0300	-0.0293	-0.5938
False	False	False	False	-0.0315	-0.0298	-1.3865
False	False	True	False	-0.0342	-0.0294	-3.8079

TABLE 16. PPO Environment 1, the 20 best results

λ	η	ϵ	Minibatch	Hidden layers	Time	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
0.9	0.00005	0.3	256	[20, 20, 20, 20, 20, 20]	1.00	-0.0186	-0.0186	-0.0416
0.9	0.00005	0.3	256	[32, 32, 32, 32]	1.00	-0.0186	-0.0186	-0.0740
0.9	0.00005	0.2	256	[20, 20, 20, 20, 20, 20]	1.00	-0.0187	-0.0185	-0.1489
1.0	0.00005	0.3	128	[20, 20, 20, 20, 20, 20]	0.87	-0.0188	-0.0186	-0.1647
0.9	0.00005	0.2	256	[32, 32, 32, 32]	0.87	-0.0188	-0.0186	-0.1866
0.9	0.00005	0.3	128	[20, 20, 20, 20, 20, 20]	1.00	-0.0187	-0.0184	-0.2124
0.9	0.00005	0.3	128	[64, 64, 64]	0.87	-0.0188	-0.0185	-0.2452
0.9	0.00005	0.3	256	[64, 64, 64]	0.87	-0.0189	-0.0185	-0.2527
0.9	0.00005	0.3	128	[32, 32, 32, 32]	0.87	-0.0189	-0.0185	-0.3022
0.9	0.01000	0.2	128	[20, 20, 20, 20, 20, 20]	1.00	-0.0191	-0.0187	-0.3157
1.0	0.00005	0.2	128	[20, 20, 20, 20, 20, 20]	0.87	-0.0190	-0.0186	-0.3315
0.9	0.00005	0.2	128	[20, 20, 20, 20, 20, 20]	0.87	-0.0190	-0.0186	-0.3340
0.9	0.00050	0.3	256	[64, 64, 64]	1.00	-0.0191	-0.0187	-0.3360
0.9	0.00005	0.3	256	[128, 128]	0.87	-0.0190	-0.0186	-0.3419
1.0	0.00005	0.3	128	[32, 32, 32, 32]	0.87	-0.0191	-0.0186	-0.3575
0.9	0.00005	0.3	128	[128, 128]	0.73	-0.0190	-0.0185	-0.4006
0.9	0.00005	0.2	256	[128, 128]	0.87	-0.0190	-0.0185	-0.4064
1.0	0.00005	0.3	128	[128, 128]	1.00	-0.0190	-0.0185	-0.4068
1.0	0.00005	0.3	256	[32, 32, 32, 32]	0.87	-0.0190	-0.0185	-0.4078
0.9	0.00050	0.2	256	[20, 20, 20, 20, 20, 20]	0.60	-0.0191	-0.0186	-0.4462

TABLE 17. PPO Environment 1, the 20 worst results

λ	η	ϵ	Minibatch	Hidden layers	Time	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
1.0	0.005	0.3	256	[20, 20, 20, 20, 20, 20]	0.33	-0.0219	-0.0185	-2.6493
1.0	0.005	0.3	128	[20, 20, 20, 20, 20, 20]	0.33	-0.0219	-0.0185	-2.6607
0.9	0.010	0.2	256	[32, 32, 32, 32]	0.20	-0.0219	-0.0186	-2.6808
1.0	0.005	0.2	128	[20, 20, 20, 20, 20, 20]	0.20	-0.0218	-0.0185	-2.6833
1.0	0.010	0.3	256	[64, 64, 64]	0.33	-0.0219	-0.0185	-2.7103
1.0	0.005	0.2	256	[32, 32, 32, 32]	0.33	-0.0218	-0.0184	-2.7185
0.9	0.005	0.2	128	[64, 64, 64]	0.20	-0.0219	-0.0185	-2.7311
1.0	0.005	0.3	256	[64, 64, 64]	0.47	-0.0219	-0.0184	-2.7550
0.9	0.005	0.2	256	[64, 64, 64]	0.20	-0.0220	-0.0186	-2.7556
1.0	0.010	0.2	128	[64, 64, 64]	0.20	-0.0220	-0.0185	-2.7719
1.0	0.010	0.2	256	[64, 64, 64]	0.33	-0.0220	-0.0185	-2.7805
1.0	0.005	0.2	256	[20, 20, 20, 20, 20, 20]	0.20	-0.0221	-0.0185	-2.8714
0.9	0.005	0.3	256	[128, 128]	0.33	-0.0221	-0.0184	-2.9212
1.0	0.005	0.3	128	[32, 32, 32, 32]	0.20	-0.0224	-0.0185	-3.1207
1.0	0.010	0.2	256	[128, 128]	0.33	-0.0224	-0.0184	-3.1606
0.9	0.010	0.2	256	[128, 128]	0.33	-0.0227	-0.0186	-3.2699
1.0	0.010	0.3	128	[64, 64, 64]	0.20	-0.0226	-0.0185	-3.2699
1.0	0.005	0.2	256	[64, 64, 64]	0.20	-0.0228	-0.0186	-3.3318
1.0	0.010	0.3	256	[32, 32, 32, 32]	0.20	-0.0227	-0.0185	-3.3981
0.9	0.010	0.2	128	[128, 128]	0.20	-0.0249	-0.0185	-5.1404

TABLE 18. PPO Environment 2, with no drift ($\mu = 0$)

λ	η	ϵ	Minibatch	Hidden layers	Time	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
0.9	0.00005	0.3	256	[16, 16, 16, 16, 16, 16, 16, 16]	0.92	-0.0293	-0.0291	-0.1281
0.9	0.00005	0.3	256	[20, 20, 20, 20, 20, 20]	0.84	-0.0293	-0.0291	-0.1567
0.9	0.00005	0.3	256	[32, 32, 32, 32]	0.92	-0.0295	-0.0292	-0.2360
0.9	0.00010	0.3	256	[32, 32, 32, 32]	0.92	-0.0295	-0.0292	-0.2692
0.9	0.00001	0.3	256	[32, 32, 32, 32, 32, 32]	0.76	-0.0296	-0.0292	-0.3851
0.9	0.00010	0.3	256	[16, 16, 16, 16, 16, 16, 16, 16]	0.76	-0.0296	-0.0291	-0.3871
0.9	0.00010	0.3	256	[20, 20, 20, 20, 20, 20]	0.76	-0.0297	-0.0291	-0.4478
0.9	0.00010	0.3	256	[64, 64, 64]	0.68	-0.0302	-0.0292	-0.8161
0.9	0.00005	0.3	256	[32, 32, 32, 32, 32, 32]	0.44	-0.0301	-0.0290	-0.8690
0.9	0.00001	0.3	256	[64, 64, 64]	0.52	-0.0303	-0.0292	-0.8920
0.9	0.00005	0.3	256	[64, 64, 64]	0.52	-0.0302	-0.0290	-0.9446
0.9	0.00010	0.3	256	[32, 32, 32, 32, 32, 32]	0.44	-0.0306	-0.0291	-1.1404
0.9	0.00001	0.3	256	[16, 16, 16, 16, 16, 16, 16, 16]	0.28	-0.0315	-0.0292	-1.8415
0.9	0.00001	0.3	256	[32, 32, 32, 32]	0.20	-0.0317	-0.0292	-2.0037
0.9	0.00001	0.3	256	[20, 20, 20, 20, 20, 20]	0.20	-0.0322	-0.0292	-2.4705

TABLE 19. PPO Environment 3, with small mean reversion ($\epsilon = 0.0005$)

α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
True	True	False	False	-0.0292	-0.0292	-0.0444
True	False	False	False	-0.0293	-0.0292	-0.0718
True	False	False	True	-0.0293	-0.0292	-0.0746
False	True	False	True	-0.0293	-0.0292	-0.0805
False	True	False	False	-0.0293	-0.0292	-0.0835
False	True	True	False	-0.0293	-0.0292	-0.0842
False	False	True	False	-0.0293	-0.0292	-0.1044
False	False	True	True	-0.0293	-0.0292	-0.1114
True	False	True	True	-0.0293	-0.0292	-0.1180
True	True	True	True	-0.0293	-0.0292	-0.1184
True	True	False	True	-0.0294	-0.0292	-0.1665
False	False	False	False	-0.0294	-0.0292	-0.1683
True	True	True	False	-0.0294	-0.0292	-0.1753
False	False	False	True	-0.0294	-0.0292	-0.1769
False	True	True	True	-0.0295	-0.0292	-0.2200
True	False	True	False	-0.0298	-0.0292	-0.5053

TABLE 20. PPO Environment 3, with large mean reversion ($\epsilon = 0.05$)

α_n	s_n	OFI	$R_{\%}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
False	True	True	True	0.0232	-0.0292	42.0496
True	False	True	False	0.0226	-0.0298	42.0418
True	True	False	False	0.0234	-0.0290	41.9904
True	True	False	True	0.0233	-0.0290	41.9854
True	True	True	False	0.0228	-0.0292	41.6796
True	True	True	True	0.0227	-0.0292	41.5851
True	False	True	True	0.0223	-0.0296	41.5831
False	True	True	False	0.0220	-0.0292	41.0591
False	True	False	False	0.0222	-0.0290	41.0365
False	True	False	True	0.0222	-0.0288	40.8446
True	False	False	True	0.0172	-0.0294	37.3609
False	False	True	True	0.0152	-0.0294	35.7130
False	False	False	True	0.0149	-0.0289	35.1317
True	False	False	False	0.0029	-0.0288	25.3759
False	False	False	False	-0.0295	-0.0290	-0.3712
False	False	True	False	-0.0299	-0.0294	-0.4304

TABLE 21. DDQN Environment 1, results from all 16 seeds

$\max \overline{v}_{agent}$	$\overline{v}_{agent}$	$\min \overline{v}_{agent}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
10.9762	7.9251	4.2540	-0.0199	-0.0186	-1.0123
11.0996	7.8571	3.6335	-0.0201	-0.0187	-1.1392
11.3095	7.8673	4.1667	-0.0201	-0.0186	-1.2087
11.0720	7.6932	3.0760	-0.0199	-0.0184	-1.2117
11.4741	7.7848	3.4064	-0.0202	-0.0186	-1.2241
11.2629	7.8041	4.2311	-0.0202	-0.0186	-1.3375
11.6520	7.8605	4.0840	-0.0205	-0.0188	-1.3821
11.2151	7.8040	3.7888	-0.0205	-0.0187	-1.3974
11.3625	7.8003	3.7610	-0.0203	-0.0186	-1.4001
10.8810	7.8891	3.1786	-0.0204	-0.0186	-1.4068
11.3745	7.8437	4.1793	-0.0201	-0.0184	-1.4182
11.2470	7.7570	3.9721	-0.0206	-0.0187	-1.4833
12.0239	7.8355	3.4980	-0.0205	-0.0187	-1.5008
11.6440	7.8102	3.7600	-0.0204	-0.0185	-1.5058
11.6996	7.9146	3.7075	-0.0207	-0.0186	-1.7041
11.4252	7.9508	3.6969	-0.0210	-0.0188	-1.8390

TABLE 22. PPO Environment 1, results from all 16 seeds

$\max \overline{v}_{agent}$	$\overline{v}_{agent}$	$\min \overline{v}_{agent}$	$\overline{G}_{agent}$	$\overline{G}_{TWAP}$	I_{BPS}
8.004	7.9965	7.940	-0.0188	-0.0188	0.0022
8.004	7.9995	7.988	-0.0187	-0.0187	0.0008
8.000	7.9995	7.992	-0.0186	-0.0186	0.0004
8.000	8.0000	8.000	-0.0184	-0.0184	0.0000
8.000	8.0000	8.000	-0.0187	-0.0187	0.0000
8.000	7.9972	7.956	-0.0185	-0.0185	-0.0017
8.008	7.9965	7.940	-0.0185	-0.0185	-0.0019
8.008	7.9975	7.952	-0.0190	-0.0190	-0.0031
8.000	7.9935	7.900	-0.0187	-0.0187	-0.0033
8.016	7.9890	7.808	-0.0188	-0.0188	-0.0065
8.004	7.9812	7.700	-0.0187	-0.0186	-0.0149
8.000	7.9805	7.688	-0.0188	-0.0188	-0.0368
9.012	7.9940	6.864	-0.0190	-0.0187	-0.2460
10.084	7.9633	6.260	-0.0191	-0.0186	-0.4680
8.032	7.6640	6.672	-0.0196	-0.0190	-0.5350
9.904	7.7930	4.636	-0.0201	-0.0188	-1.0032

REFERENCES

[AC01] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

[ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017.

[BCP$^+$16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. 2016.

[BL98] Dimitris Bertsimas and Andrew Lo. Optimal control of execution costs. *Journal of Financial Markets*, 1(1):1–50, 1998.

[BM95] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

[CJP15] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[DBK$^+$17] 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*, 28(3):653–664, March 2017.

[EH89] Charles Engel and James D Hamilton. Long swings in the exchange rate: are they in the data and do markets know it? Technical report, National Bureau of Economic Research, 1989.

[Fel16] David Fellah. Active learning in trading algorithms - reinforcement learning in algorithmic execution. 2016. Global Derivatives Conference, Budapest.

[FL09] Henri Faure and Christiane Lemieux. Generalized halton sequences in 2008: A comparative study. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 19(4):15, 2009.

[Gro04] Jeffrey Gropp. Mean reversion of industry stock returns in the us, 1926–1998. *Journal of Empirical Finance*, 11(4):537–551, 2004.

[Gui18] T. Guida. *Big Data and Machine Learning in Quantitative Investment*. Wiley Finance. Wiley, 2018.

[HMvH$^+$17] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[HW14] Dieter Hendricks and Diane Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2104 IEEE Conference on*, pages 457–464. IEEE, 2014.

[JDO$^+$17] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[Kyl85] Albert S Kyle. Continuous auctions and insider trading. *Econometrica: Journal of the Econometric Society*, pages 1315–1335, 1985.

[LJR$^+$18] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning, 2018.

[MBM$^+$16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[MNW$^+$17] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889, 2017.

[NFK06] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[NLJ18] Brian Ning, Franco Ho Ting Ling, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *arXiv preprint arXiv:1812.06600*, 2018.

[PS88] James M Poterba and Lawrence H Summers. Mean reversion in stock prices: Evidence and implications. *Journal of financial economics*, 22(1):27–59, 1988.

[Rie05] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sch97] Eduardo S Schwartz. The stochastic behavior of commodity prices: Implications for valuation and hedging. *The journal of finance*, 52(3):923–973, 1997.

[SFSK18] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. Market making via reinforcement learning. *CoRR*, abs/1804.04216, 2018.

[Sil16] David Silver. Reinforcement learning, 2016.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[SML$^+$15] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[SWD$^+$17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[vHGS15] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[VHGS16] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WMG$^+$17] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.