Eötvös Loránd University

Faculty Of Science

Algorithmic trading with reinforcement learning

Leonardo Toffalini

Undergraduate Mathematics Thesis

Supervisor:

András Lukács

Department of Computer Science &
AI Research Group

Budapest
2025

Contents

Acknowledgements

First and foremost, I would like to express my gratitude towards my thesis advisor and research coordinator, András Lukács, without whom this thesis would not have been possible. I would also like to acknowledge Lóránt Nagy for contributing his expertise in financial mathematics and for facilitating progress through his relentless dedication and motivation. Last but not least, I would also like to thank Bálint Csanády for his insightful ideas that propelled forward our research.

Introduction

Algorithmic trading is a vibrant research area at the interface of academia and industry. Our recent research contained in the present thesis focuses on bridging reinforcement learning theory and quantitative finance to explore trading policies in an artificial financial environment where the price of a single risky asset follows fractional Brownian motion, and in which the phenomenon of market friction is incorporated. Building on results by Guasoni, Nika, and Rásonyi [7] we benchmark artificial trading algorithms which were learned by an agent through reinforcement learning. The work in progress shows that the problem can be formalized in the framework of reinforcement learning, and that for a reasonable horizon, and anti-persistent price dynamics, the theoretical rates are achieved by an artificial agent using the Proximal Policy Optimization algorithm within a few million training steps.

Reinforcement Learning (RL) has rapidly emerged as a cornerstone of modern artificial intelligence, powering landmark achievements that redefine the boundaries of what computers are capable of. From AlphaGo's mastery over the complex game of Go [19] and AlphaGeometry's automated geometry proofs at Olympiad level [22], to superhuman performance in video game environments like that of Atari 2600 [13], Dota 2 [3], and StarCraft II [23], the impact of RL is undeniable. Its principles are also integral to refining large language models through Reinforcement Learning from Human Feedback (RLHF) [4], and Group Relative Policy Optimization (GRPO) [17]. Reinforcement learning also enables increasingly sophisticated autonomous robots to navigate and interact with the physical world.

At its heart, RL formalizes the intuitive process of learning from experience, a mechanism arguably fundamental to how intelligent beings, including humans, adapt and thrive. Much like a toddler iteratively discovers the mechanics of walking through persistent trial and error, without explicit supervised instruction, RL agents learn optimal strategies by interacting with an environment and observing the consequences of their actions. This intrinsic ability to generate its own learning data through experience distinguishes RL from many other machine learning paradigms that rely on precompiled datasets. The mathematical framework of reinforcement learning will be further explored in the first chapter.

This thesis is structured into four main chapters, each building upon the preceding material to develop a rigorous mathematical treatment of reinforcement learning and its application to trading strategies on stylized market models.

The first chapter lays the theoretical foundation by introducing Markov decision processes, which serve as the formal framework for modeling sequential decision-making problems. The chapter proceeds to define value functions, which quantify the expected returns of states or actions under a given policy. Subsequently, the Bellman expectation equation is presented as a fundamental recursive relationship for value functions, followed by the Bellman optimality

equation, which characterizes the optimal value function and forms the basis for many solution methods.

The second chapter surveys the principal algorithmic approaches to solving reinforcement learning problems. It begins with value iteration, a dynamic programming method for computing optimal policies, and Q-learning, a widely used off-policy temporal-difference algorithm. The chapter then discusses policy gradient methods, which optimize parameterized policies directly, and examines the REINFORCE algorithm as a canonical example. Further, it covers the Advantage Actor Critic (A2C) method, which combines value-based and policy-based approaches, and concludes with Proximal Policy Optimization (PPO), a modern algorithm known for its stability and performance in complex environments.

The third chapter shifts its focus to the mathematical modeling of financial time series. It begins with an overview of stochastic processes, providing the necessary background. The discussion then centers on fractional Brownian motion, a generalization of classical Brownian motion that captures long-range dependence and self-similarity, and explores its relevance and challenges in the context of trading strategies.

The fifth and last chapter presents empirical investigations designed to evaluate the practical performance of reinforcement learning algorithms in artificial trading environments. The first section describes experiments involving an autoregressive price model, while the second section addresses trading scenarios as described in the third chapter.

The thesis concludes with a brief summary of the principal findings and outlines possible directions for future research.

Chapter 1

Introduction to Reinforcement Learning

This chapter will explore and formalize the core concepts of reinforcement learning from a computational perspective. While not intended as an exhaustive guide to the vast landscape of RL, the following sections will equip the reader with the foundational knowledge necessary to understand its mechanisms and, critically, to approach and tackle a novel financial investment problem, as will be detailed in later chapters. For a comprehensive guide to the field, readers are encouraged to consult seminal texts such as Sutton and Barto [21].

1.1 Markov decision processes

Markov decision processes conceptualize the notion of an actor with agency interacting with an environment and receiving rewards contingent on its actions. In the continuation of this chapter and the subsequent one, we endeavor to provide solutions to this mathematical system.

Definition 1.1.1. A Markov decision process (MDP) is defined by the tuple $(\mathcal{S},\ \mathcal{A},\ \mathcal{P}_a,\ \mathcal{R}_a)$ where

- $\mathcal{S}$ is the set of all possible states, called the state space.
- $\mathcal{A}$ is the set of all possible actions, known as the action space.
- $\mathcal{P}$ is the state transition probability function, such that $\mathcal{P}^a_{ss'}$ is the probability of transitioning to state s' upon taking action a at state s, expressed as

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s,\ A_t = a]$$

- $\mathcal{R}$ is the reward function, such that $\mathcal{R}^a_s$ represents the reward received after taking action a at state s, expressed as

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s,\ A_t = a]$$

Within the framework of a Markov decision process, an agent exercises control by selecting actions. The function responsible for associating states with corresponding actions is referred to

as the policy function, denoted as π.
A policy may be deterministic, meaning it assigns a specific action to each state.

$$\pi : \mathcal{S} \to \mathcal{A}$$

A policy may also be probabilistic, thereby associating each state with a probability distribution over possible actions.

$$\pi : \mathcal{S} \to P(\mathcal{A})$$

In a Markov decision process, the successor state is solely influenced by the current state, rendering preceding states inconsequential. This indicates that an MDP lacks memory of any previous states, a characteristic known as the *Markov property*. Consequently, the system is fully represented by the state transition probability function $\mathcal{P}$. When constructing a model of a system with *memory*, it is feasible to incorporate all requisite information from prior states within the current state.

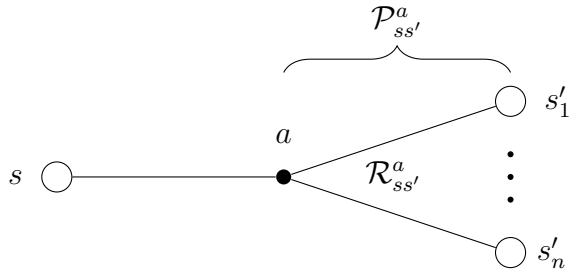


Figure 1.1: Diagram of a single step in a Markov Decision Process, where hollow circles represent states, while full circles represent actions.

1.2 Value function

Value functions play a quintessential role in reinforcement learning by estimating the expected cumulative reward an agent may obtain from a given state or state-action pair, under a specific policy. These functions provide a quantitative measure for assessing the quality of decisions which may inform the agent's strategy as it interacts with its environment.

Definition 1.2.1. The return G_t is defined as the total discounted reward from time step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.1}$$

where $\gamma \in [0, 1]$ is the *discount rate* and R_t is the random variable that denotes the reward at time step t.

Remark 1.2.2. The closer γ is to 1, the more future rewards matter; the closer γ is to 0, the more immediate rewards matter.

It is clear to see that the return at time step t is directly related to the return at time step $t+1$, as demonstrated by equation (1.2).

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{1.2}$$

Definition 1.2.3. The state-value function $v : \mathcal{S} \to \mathbb{R}$ is a function that maps states to values. The value of a state is defined as the expected discounted return starting from state s and following policy π.

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{1.3}$$

Notice that in a Markov decision process, the state-value function depends on the policy due to the agent's capacity to influence its trajectory through the environment. Consequently, the state-value function essentially provides an estimate of the expected return obtainable from state s when adhering to policy π.

Definition 1.2.4. The action-value function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a function that maps state-action pairs to values. The value of an action at a given state is the expected discounted return starting from state s and taking action a and from then onward following policy π.

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s,\ A_t = a \right] \tag{1.4}$$

Analogous to the state-value function, the action-value function provides an estimate of the expected return from a specific state s subsequent to executing action a, while adhering to the policy π.

1.3 Bellman expectation equation

In this section, we will establish foundational results concerning the previously defined value functions. Upon concluding this section, we will derive equations for both the state-value and action-value functions that connect the value of the present state to that of its successor states, thereby providing a system of equations applicable to all Markov decision processes.

Proposition 1.3.1.

$$v(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s \right] \tag{1.5}$$

Proof.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1}] \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned} \tag{1.6}$$

In the penultimate equality, we use the law of total expectation to eliminate the inner conditional expectation. □

An analogous equation for the action-value function can be stated by reversing the order of operations in the previous proposition.

Proposition 1.3.2.

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1},\ A_{t+1}) \mid S_t = s,\ A_t = a] \tag{1.7}$$

Proof.

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1}, A_{t+1}] \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \end{aligned} \tag{1.8}$$

□

In the following, we provide an equation that relates the state-value function to the action-value function, and vice versa. Through these bidirectional relationships, we will derive the Bellman expectation equation.

First we provide a relationship between the state-value function and the action-value function. Figure 1.2a depicts a one-step look-ahead, which provides visual aid for equation (1.9).
To express the value of the present state, we take the weighted sum of the values of the state-action pairs that are reachable from s.

Proposition 1.3.3.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) \tag{1.9}$$

Proof.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\mathbb{E}_\pi[G_t, \mid S_t, A_t], \mid S_t = s] \\ &= \mathbb{E}_\pi[q_\pi(s, A_t) \mid S_t = s] = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) \end{aligned} \tag{1.10}$$

□

Figure 1.2b mirrors Figure 1.2a, albeit in a reversed order, offering an alternative one-step look-ahead with respect to potential transitions that may incur from state s upon executing action a. Equation (1.11) states that the action-value of the present state-action pair consists of the immediate reward upon taking action a at state s and the discounted weighted sum of the values of the successor states s'.

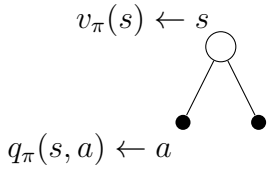
Proposition 1.3.4.

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \tag{1.11}$$

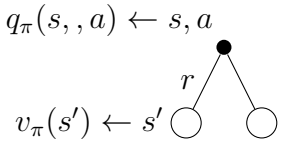
Proof.

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[G_t \mid S_t = s,\ A_t = a] = \mathbb{E}[R_t + \gamma G_{t+1} \mid S_t = s,\ A_t = a] \\ &= \mathbb{E}\left[R_t + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s] \mid S_t = s,\ A_t = a\right] \\ &= \mathbb{E}\left[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a\right] \\ &= \mathcal{R}_s^a + \gamma \mathbb{E}\left[v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a\right] \\ &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \end{aligned} \tag{1.12}$$

□



(a) Backup diagram for equation (1.9).



(b) Backup diagram for equation (1.11).

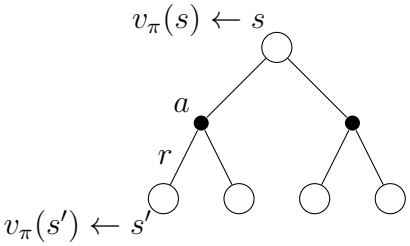
Figure 1.2: Backup diagrams for equations (1.9) and (1.11), where the hollow circles represent states and the full circles represent actions.

Taking the two one-step look-ahead diagrams, we derive the composite diagram 1.3a. Writing equation (1.9) and substituting for $q_\pi(s,a)$ using (1.11) we get the recursive equation (1.13) that relates the value of the current state to that of the successor states. The stated equation is called the Bellman expectation equation for the state-value function.

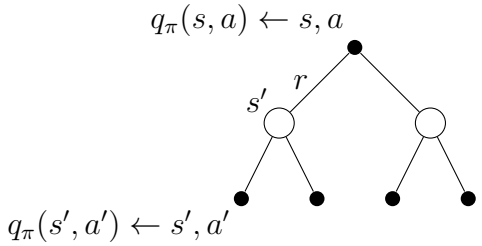
$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right) \tag{1.13}$$

Reversing the order of the two diagrams, we get a new composite diagram 1.3b that relates the action-value of the present state-action pair to that of the possible successor state-action pairs. Same again, writing equation (1.11) and substituting $v_\pi(s')$ using (1.9) we get the following recursive equation for q_π. Equation (1.14) is called the Bellman expectation equation for the action-value function.

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' \mid s') q_\pi(s', a') \tag{1.14}$$



(a) Backup diagram for equation (1.13).



(b) Backup diagram for equation (1.14).

Figure 1.3: Backup diagrams for equations (1.13) and (1.14), where the hollow circles represent states and the full circles represent actions.

1.4 Bellman optimality equation

In the preceding section, we formulated two equations: the first establishes a relationship between the state-value function of the current state and the weighted average of the state-value of its successor states (1.13), while the second connects the action-value of the existing state-action pair to that of the subsequent states and actions (1.14). Nevertheless, the aforementioned equations are applicable to all policies. Our objective, however, is to identify an equation that is

specifically applicable to the optimal policy, which would enable us to solve it and consequently derive the optimal policy.

In this section, we define what an optimal policy and an optimal value function are. We will provide an equation that resolves the Markov decision process by presenting a formulation for the optimal state-value and action-value functions, utilizing the equations established in the preceding section.

Definition 1.4.1. The optimal *state-value* function $v_*(s)$ is the maximum state-value function over all policies.

$$v_*(s) = \max_\pi v_\pi(s) \quad \forall s \in \mathcal{S}$$

Definition 1.4.2. The optimal *action-value* function $q_*(s, a)$ is the maximum action-value function over all policies.

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

In order to define the optimal policy, we ought to define a partial ordering on the set of all policies.

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

This implies that one policy is considered *greater* to another if it yields a greater expected discounted return for all states. Given that the state-value function is bounded by above, there exists a policy π_* such that

$$\pi_* \geq \pi \quad \forall \pi$$

Notice that π_* is not unique, as distinct policies may achieve the same discounted return. Nevertheless, all optimal policies are denoted by π_*. Moreover, by definition, it follows that if $\pi = \pi'$ then $v_\pi = v_{\pi'}$. Consequently, all optimal policies induce the same state-value function, namely $v_{\pi_*} = v_*$.
The same property holds for the action-value function, that is, $q_{\pi_*} = q_*$.

Through the newly established definitions of optimal state-value functions, action-value functions, and policies, we are now equipped to delineate our objective in solving a Markov Decision Process (MDP): the derivation of the optimal policy π_*.

In the preceding section, we derived an equation that relates the value of the current state to the weighted average value of its successor states. By amending equation (1.13), it can be restructured to represent the optimal policy, thus eliminating the need to compute the average in favor of determining the maximum value among the successor states.

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right) \tag{1.15}$$

In summary, the aforementioned equation conveys that the value of a state under the optimal policy must equal the immediate reward and the weighted average value of the successor states corresponding to the optimal action.

The same can be applied to the Bellman expectation equation for the action-value function (1.14).

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a') \tag{1.16}$$

The above two equations are the Bellman optimality equation for the state-value function and for the action-value function, respectively.

The Bellman optimality equations, when solved, yield the optimal state-value or action-value function, subsequently leading to the derivation of the optimal policy. Nevertheless, a closed-form solution does not exist for these equations in general. Moreover, the resolution of the equations necessitates knowledge about the dynamics of the environment $\mathcal{P}$, which may not always be available. The subsequent chapter will present two iterative methods for resolving the Bellman optimality equations: Value iteration and Q-learning.

Chapter 2

Reinforcement Learning Algorithms

In the previous chapter, we established the foundational framework of reinforcement learning, formalizing the problem setting through Markov Decision Processes (MDPs) and introducing the core concepts that underpin learning from interaction. Building on this groundwork, the focus of this chapter shifts to the central question of how does an agent learn an optimal policy within an MDP?

The ultimate objective in reinforcement learning is for an agent to discover a policy that maximizes the expected cumulative (discounted) reward by navigating through a complex environment solely through experience. Achieving this requires algorithmic strategies that iteratively improve the agent's behavior based on feedback from its interactions.

In this chapter, we will explore a wide variety of reinforcement learning algorithms, beginning with fundamental iterative methods that progressively refine value estimates. Each algorithm we present addresses limitations of its predecessors, advancing in complexity and capability. Our exploration of reinforcement learning algorithms culminates with the derivation of Proximal Policy Optimization (PPO), an algorithm that remains a state-of-the-art standard due to its balance of sample efficiency, stability, and ease of implementation.

2.1 Value iteration

The Bellman optimality equation for the state-value function (1.15) describes a system of equations that yield the optimal state-value function, which implies the optimal policy. Nevertheless, a general solution does not exist in general. This section provides one way of solving this equation through the iterative improvement of an approximation of the optimal state-value function.

Given that the Bellman equation is inherently a recursive equation relating the value of the present state to that of the successor states, it can be transformed into an iterative procedure. Iterative algorithms of this nature are prevalent in numerical methods employed for solving systems of equations, wherein their convergence is frequently contingent upon principles of contraction mapping.

In this section, we will not demonstrate the convergence of the Value iteration algorithm to the optimal state-value function. Nonetheless, it suffices to assert that by establishing a complete

metric space defined over the state-value function and employing a contraction mapping to articulate the update rule used in Value iteration, it consequently follows that the iterative application of this mapping leads to a unique fixed point, which is precisely v^*.

Algorithm 1 Value Iteration

while $\Delta > \theta$ **do**
  $\Delta \leftarrow 0$
  $v_{\text{prev}} \leftarrow v$
  for $s \in S$ **do**
    $v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\text{prev}}(s') \right)$
    $\Delta \leftarrow \max\{\Delta,\ |v(s) - v_{\text{prev}}(s)|\}$
  end for
end while

In this context, θ serves as the convergence criterion; it is a hyperparameter that indicates when the algorithm has achieved convergence. The convergence criterion manages the trade-off between computational cost and accuracy; a smaller θ value results in a more precise value function at the expense of increased computational time.

During each iteration, the largest change in state-values is computed and stored in Δ, and convergence is deemed to occur when the largest change in state-value throughout a complete iteration is less than θ. The essential component of the algorithm is represented by the following statement, which corresponds to equation (1.15) but without referencing the optimal policy.

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\text{prev}}(s') \right) \tag{2.1}$$

The objective of the value iteration algorithm is to learn the optimal state-value function, v_*. However, our primary interest lies in determining the optimal policy, π_*. To derive the policy from the evaluated state-value function, the policy is defined to act greedily, selecting actions that maximize the estimated return.

$$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) \tag{2.2}$$

It should be noted that in order to calculate the new value function, knowledge of the state transition probability function $\mathcal{P}$ is required. Generally, this knowledge cannot be presumed, as in most scenarios, we lack access to an accurate world model.

2.2 Q-learning

In value iteration, the update of the current value function requires an understanding of the environmental dynamics to compute the new value of the state. This approach is referred to as *model based* RL, in contrast to *model free* RL, where the agent operates without access to the system's underlying dynamics. This requirement is most often a limitation, as in real-world applications, we cannot presume the availability of an accurate world model.

Q-learning mitigates this limitation by directly learning from interactions, utilizing the observed successor state and the received reward to update its action-value estimates. In Q-learning, a *Q-function* is maintained, which, similar to the learned value function in value iteration, approximates the optimal action-value function. The signature of the Q-function is represented as $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Importantly, in scenarios where both state and action spaces are finite, the Q-function can be thought of as a table with $|\mathcal{S}|$ rows and $|\mathcal{A}|$ columns; hence, it is often referred to as the Q-table.

Q-learning, like value iteration, is a value-based method. Consequently, the agent first learns an action-value function, from which a policy is derived post-training. The trivial extraction of policy from a learned Q-function involves greedily selecting the optimal action at each state based on the learned action-values.

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \tag{2.3}$$

where Q^* is the optimal Q-function that induces the optimal policy π^*.

Algorithm 2 Q-learning

for $i \leftarrow 1$, num episodes **do**
  $\varepsilon \leftarrow \varepsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g. ε-greedy)
    Take action A_t and observe the reward and state that follows, R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow (1 - \alpha)Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right)$
    $t \leftarrow t + 1$
  until S_t is terminal
end for
return Q

The update rule is derived through a modification to the Bellman optimality equation for the action-value function. Direct application of the Bellman equation within the algorithm is infeasible, as it explicitly requires the state probability transition function. Consequently, the Q-value is updated on the basis of the observed successor state and reward, thereby mitigating the requirement for the state transition probability function.

$$Q(S_t, A_t) \leftarrow (1 - \alpha)Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right) \tag{2.4}$$

During the sampling of actions from the Q-function in the training phase, it is possible to introduce some randomness by refraining from selecting the current optimal action prescribed by the Q-function, opting instead to select randomly with a probability of ε. This approach, known as ε-greedy, facilitates the agent to *explore* rather than merely *exploiting* the prevailing optimal action. Q-learning is characterized as an *off-policy* algorithm, on account of the difference between the final policy it generates and the exploratory policy utilized during the accumulation of experience in training.

In Reinforcement Learning, we must balance exploration versus exploitation. It is clear that at the early stages of training, it is evident that the agent has not experienced most of the environment

yet, thus it would be beneficial for it to explore more and collect experience. Conversely, when the agent has found a strategy that has high return, it should exploit it.

A common approach to balance the exploration-exploitation trade-off in Q-learning is to start training with a high ε value and applying an annealing function throughout the training process. This approach ensures that as training advances, the value is gradually reduced and the rate of exploration diminishes.

2.3 Policy gradient methods

So far we have only discussed *value based* RL algorithms, where we indirectly learn the optimal policy by learning the optimal value function. An alternative approach to solving the reinforcement learning control problem is to directly learn a policy by iteratively improving on the current policy, thus leading to an incremental improvement at each step; these types of RL algorithms are called *policy based*.

In this section, our focus will be directed on *policy gradient* algorithms, which are a subclass of policy-based algorithms. In a policy gradient algorithm, the policy is parameterized, and an *objective function* is employed to measure the performance of the policy. When improving the policy after each step of the algorithm, we compute the objective function and adjust the parameters of the policy in the direction of the gradient of the objective function. By following the gradient, we ensure that the change in parameters will be that of the greatest ascent (or descent).

The primary advantage of policy gradient algorithms lies in their ability to effectively generalize to continuous action spaces. It should be noted that in the value iteration and Q-learning algorithms, calculating a maximum over all actions is required. This task becomes prohibitively expensive when applied to continuous action spaces. However, for policy gradient algorithms, the computational burden of determining the gradient remains consistent, regardless of whether the action space is of high or low dimensionality.

Among the most common policy gradient algorithms are REINFORCE, Advantage Actor Critic, Trust Region Policy Optimization, and Proximal Policy Optimization. These methods will be covered in the following sections. Nonetheless, a comprehensive grasp of policy gradient algorithms is indispensable for the comprehension of the algorithms that follow.

Let us introduce the following objective function for continuing tasks, that measures the average value of all states weighted by their stationary distribution.

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi}(s) v_{\pi}(s) \tag{2.5}$$

where d^{π} represents the stationary distribution of the Markov chain over the state space by adhering to π, and v_{π} denotes the *true* value function under π. This objective function ensures that states frequently visited by the policy bear greater significance.

For episodic tasks, the objective function can be defined as the value of the initial state under policy π.

$$J(\theta) = v_{\pi}(s_0) \tag{2.6}$$

The episodic and continuing cases define different objective functions $J(\theta)$, necessitating that they be treated separately. Nevertheless, they ultimately lead to the same result, with the major part of the derivation being identical for both cases. Therefore, we focus on the episodic case and propose the continuing case as an exercise for the reader.

Directly calculating the gradient of the objective function might seem to be terribly challenging, given that the parameter affects the policy, which in turn affects the stationary distribution. Furthermore, it is also dependent on the successor states after selecting an action, which is controlled by the environment. In situations where complete information about the environment is unavailable, direct computation of the objective function or its gradient is not feasible. Fortunately, the gradient can be reformulated to resemble an expectation, allowing it to be approximated through sampling methods.

Theorem 2.3.1. *Policy gradient theorem*

$$\nabla_\theta J(\theta) \propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} q_\pi(s,a) \nabla_\theta \pi_\theta(a \mid s) \tag{2.7}$$

Proof. To keep the notation clean and readable, we omit writing θ in all cases, and leave it implicit that the gradients are with respect to θ and that policy π is a function of θ.

By equation (1.9), we can rewrite the state value function.

$$\nabla v_\pi(s) = \nabla \left[\sum_a \pi(a \mid s) q_\pi(s,a) \right] \tag{2.8}$$

We can apply the sum and the product rule of the gradient.

$$\nabla v_\pi(s) = \sum_a \Big[\nabla\pi(a \mid s) q_\pi(s,a) + \pi(a \mid s) \nabla q_\pi(s,a) \Big] \tag{2.9}$$

By equation (1.11), we can rewrite the action-value function.

$$\nabla v_\pi(s) = \sum_a \left[\nabla\pi(a \mid s) q_\pi(s,a) + \pi(a \mid s) \nabla \sum_{s'} \mathcal{P}^a_{ss'} \left(r + v_\pi(s') \right) \right] \tag{2.10}$$

Since the reward received does not depend on θ, its gradient is zero.

$$\nabla v_\pi(s) = \sum_a \left[\nabla\pi(a \mid s) q_\pi(s,a) + \pi(a \mid s) \sum_{s'} \mathcal{P}^a_{ss'} \nabla v_\pi(s') \right] \tag{2.11}$$

It appears that the gradient of the value of the current state depends on the gradient of the value of the successor states. With this recursive formula, we can unroll the equation by substituting the formula for ∇v_π into $\nabla v_\pi(s')$.
Through sufficient expansion of this recursive relationship, it is ensured that all states of the state space are visited, thereby resulting in the subsequent equation.

$$\nabla v_\pi(s) = \sum_{x\in\mathcal{S}} \sum_{k=0}^{\infty} \mathbb{P}(s \to x, k, \pi) \sum_a \nabla\pi(a \mid x) q_\pi(x,a) \tag{2.12}$$

where $\mathbb{P}(s \to x, k, \pi)$ is the probability of reaching state x starting from state s, after k steps, by following policy π.

Let $\eta(s) = \sum_{k=0}^{\infty} \mathbb{P}(s_0 \to s, k, \pi)$ and $\phi(s) = \sum_a \nabla\pi(a \mid s) q_\pi(s, a)$. With these shorthand notations in place, we can rewrite the latest form of the equation.

$$\begin{aligned}
\nabla J(\theta) = \nabla v_\pi(s_0) &= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \mathbb{P}(s_0 \to s, k, \pi) \sum_{a \in \mathcal{A}} \nabla\pi(a \mid s) q_\pi(s, a) \\
&= \sum_{s \in \mathcal{S}} \eta(s)\phi(s) = \sum_{s \in \mathcal{S}} \left(\frac{\sum_s \eta(s)}{\sum_s \eta(s)} \right) \eta(s)\phi(s) \\
&= \left(\sum_{s \in \mathcal{S}} \eta(s) \right) \sum_{s \in \mathcal{S}} \frac{\eta(s)}{\sum_s \eta(s)} \phi(s) \propto \sum_s \frac{\eta(s)}{\sum_s \eta(s)} \phi(s) \\
&= \sum_{s \in \mathcal{S}} d^\pi(s)\phi(s)
\end{aligned} \tag{2.13}$$

Given that $\sum_{s \in \mathcal{S}} \eta(s)$ is a constant, and $\frac{\eta(s)}{\sum_s \eta(s)}$ is precisely $d^\pi(s)$, we finally arrive at the desired result after resubstituting for our shorthand notation of $\phi(s)$.

$$\nabla J(\theta) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla\pi(a \mid s) q_\pi(s, a) \tag{2.14}$$

□

Observe that the expression on the right-hand side of the policy gradient theorem consists of a weighted sum over all states, with each state weighted according to its likelihood under policy π. Consequently, adhering to policy π results in encountering states in these proportionalities. Given that d^π is a probability distribution over $\mathcal{S}$, and $\phi(s)$ constitutes a function over $\mathcal{S}$, the right-hand side of the policy gradient theorem embodies an expectation.

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} \nabla\pi(a \mid s) q_\pi(s, a) \right] \tag{2.15}$$

The intention is to employ the policy gradient theorem as an update rule within a gradient based algorithm. Nevertheless, the current formulation remains unsampleable due to the necessity of summing over the entirety of actions. To transform this summation into an expectation, it is requisite to multiply each action by its corresponding selection probability, which precisely constitutes the policy π.

$$\begin{aligned}
\nabla J(\theta) &= \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s, a) \frac{\nabla\pi(a \mid s)}{\pi(a \mid s)} \right] \\
&= \mathbb{E}_\pi \left[\mathbb{E}_\pi \left[q_\pi(s, a) \frac{\nabla\pi(a \mid s)}{\pi(a \mid s)} \right] \right] \\
&= \mathbb{E}_\pi \left[q_\pi(s, a) \frac{\nabla\pi(a \mid s)}{\pi(a \mid s)} \right]
\end{aligned} \tag{2.16}$$

Recall that $\nabla \log f = \frac{\nabla f}{f}$ by the chain rule. Thus, we derive the following update rule, which will serve as the foundational update rule for the algorithms that follow.

$$\nabla J(\theta) = \mathbb{E}_\pi \left[q_\pi(s, a) \nabla \log \pi(a \mid s) \right] \tag{2.17}$$

2.4 REINFORCE

In the preceding section, we derived the generic update rule applicable to policy gradient methods (2.17). The principal challenge remaining with this update rule arises from the absence of the true action value function q_π. Thus, an estimate or a substitution for the action-value function is required that does not incur bias. Recall that $q_\pi(s,a) = \mathbb{E}\left[G_t \mid s, a\right]$, which implies that the return is an unbiased estimator of the action-value. Therefore, the action-value function can be substituted for its definition in the generic policy gradient update.

$$\nabla J(\theta) = \mathbb{E}_\pi\left[\mathbb{E}_\pi\left[G_t \mid S_t = s,\ A_t = a\right] \cdot \nabla \log \pi(a \mid s)\right] \tag{2.18}$$

By the law of total expectation, we can eliminate the inner expectation.

$$\nabla J(\theta) = \mathbb{E}_\pi\left[G_t \cdot \nabla \log \pi(a \mid s)\right] \tag{2.19}$$

With the aforementioned update rule, we present the REINFORCE algorithm [25].

Algorithm 3 REINFORCE

Initialize a differentiable policy $\pi_{\boldsymbol{\theta}}$ with parameter $\boldsymbol{\theta} \in \mathbb{R}^d$
repeat
  Sample an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following policy $\pi_{\boldsymbol{\theta}}$
  for each step of the episode $t = 0, \ldots, T-1$ **do**
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t, S_t)$
  end for
until π converges

However, the REINFORCE algorithm is only applicable for episodic environments, as the full return of an episode is required for a single update. For episodic environments, the sampled update serves as an unbiased estimator of the true policy gradient update. Nevertheless, we must also consider the variance of the sample alongside the bias.
In order to control the variance of the sample, we can introduce a baseline function $B(s)$, dependent solely on the state rather than the action.

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \Big(q_\pi(s,a) - b(s)\Big) \nabla_\theta \pi_\theta(a \mid s) \tag{2.20}$$

It is clear to see that the introduction of the baseline does not alter the validity of the policy gradient theorem, as the introduced part simplifies to zero.

$$\sum_{a \in \mathcal{A}} b(s) \nabla_\theta \pi_\theta(a \mid s) = b(s) \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) = b(s) \nabla_\theta 1 = 0 \tag{2.21}$$

By subtracting a baseline function, we can differentiate between actions more clearly for different states.
Consider a scenario where, in state s_1, one action holds a value of 11 while another action possesses a value of 12. Similarly, in state s_2, one action is valued at 1, and its alternative at 2. In

these examples, although the difference between the two actions remains constant across both states, the update magnitude at state s_1 will be significantly larger, attributed to the higher overall value of the optimal action compared to the best action at state s_2. To rectify this discrepancy, it is proposed to establish the state's value as the baseline, ensuring that the updates for states s_1 and s_2 are equivalent.

2.5 Advantage Actor Critic

Previously, we have derived the policy gradient theorem, which enables the update of the policy by sampling trajectories. Nonetheless, the equation directly references the action-value function $q(s, a)$, the exact form of which is unknown to us. The REINFORCE algorithm employs an approximation of the action-value function; however, this approximation does not undergo refinement. Actor Critic methods improve on this idea by *learning* the action-value function and incorporating the learned action-value into the update rule. In this framework, the actor is responsible for learning the policy, whereas the critic is tasked with learning the action-value. Consequently, Actor Critic methods are categorized as both value-based and policy-based.

This section is dedicated to the development of an Actor Critic method that is designed to approximate the so-called *advantage*, and learns a policy based on the input provided by the advantage estimate.

In the preceding section, we argued that the subtraction of the state's value as a baseline during each update does not compromise the validity of the policy gradient theorem, provided the baseline remains independent of the action. Nevertheless, this approach may contribute to a reduction in variance by centering the sampled values. The policy gradient update incorporating the state value baseline is expressed as follows.

$$\nabla J(\theta) = \mathbb{E}_\pi \left[(q_\pi(s, a) - v_\pi(s)) \nabla \log \pi(a \mid s) \right] \tag{2.22}$$

The inner part comes up frequently enough that it has a name; it is called the *advantage*.

$$A_\pi(s, a) := q_\pi(s, a) - v_\pi(s) \tag{2.23}$$

Should one endeavor to formulate an algorithm based on the aforementioned update rule, it may be necessary to estimate both the state-value function and the action-value function. However, a more efficient approach can be adopted by solely estimating the state-value function, provided that the advantage function is appropriately reformulated. By employing the definition of the action-value function and partially unrolling the Bellman equation, one can deduce the following outcomes.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a] = \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a] \tag{2.24}$$

Rewriting the definition of the advantage function with the derived formulation of the action-value function, we get the following.

$$A_\pi(s, a) = \overbrace{\mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a]}^{q_\pi(s,a)} - v_\pi(s) \tag{2.25}$$

Let us introduce the temporal difference (TD) error to tighten up our notation.

$$\delta_\pi := R_t + \gamma v_\pi(S_{t+1}) - v_\pi(S_t) \tag{2.26}$$

By considering the expectation of the temporal difference (TD) error conditioned on the state and action, it becomes evident that the TD error serves as an unbiased estimator of the advantage function when the true value function is employed.

$$\mathbb{E}_\pi \left[\delta_\pi \mid S_t = s,\ A_t = a\right] = \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a] - v_\pi(s) = A_\pi(s, a) \tag{2.27}$$

Thus, the TD error can be employed to calculate the update for the policy gradient.

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\nabla \log \pi(a \mid s)\delta_\pi\right] \tag{2.28}$$

Having established this reformulation, it becomes sufficient to approximate the state value function to facilitate the completion of a policy gradient update. Numerous methods exist for learning the state value function, although we will not delve into the specifics of these various approaches. Among the simplest methods is TD Learning. The update rule for a single-step temporal difference method, commonly referred to as TD(0), is presented as follows.

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.29}$$

This update rule implies that following each action undertaken, the one-step TD error, $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, is computed based on our present estimation of $V(S_t)$, and the estimate is incrementally adjusted using the sampled value of the TD error.

2.6 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [16] represents the culmination of all our previous efforts at improving RL algorithms. PPO is an actor critic algorithm distinguished by two key ideas: the generalized advantage estimate (GAE) and the clipped surrogate objective.

Definition 2.6.1. Let L^{CPI} denote the conservative policy iteration (CPI) objective function.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta)\hat{A}_t\right] \tag{2.30}$$

where the ratio function r_t is defined as follows:

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

Our objective is to avoid large changes to the current policy in one step, as they might impact convergence. One method to limit the change in policy is to impose a penalty based on the Kullback-Leibler (KL) divergence [10] of the policy, which is the key idea of TRPO [14].

However, PPO employs a different approach by clipping the change in policy rather than penalizing it. The following equation defines the clipped surrogate objective for PPO. Additionally, Figure 2.1 offers an illustration of the clipped surrogate objective function.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),\ 1-\varepsilon,\ 1+\varepsilon\big)\hat{A}_t\Big] \tag{2.31}$$

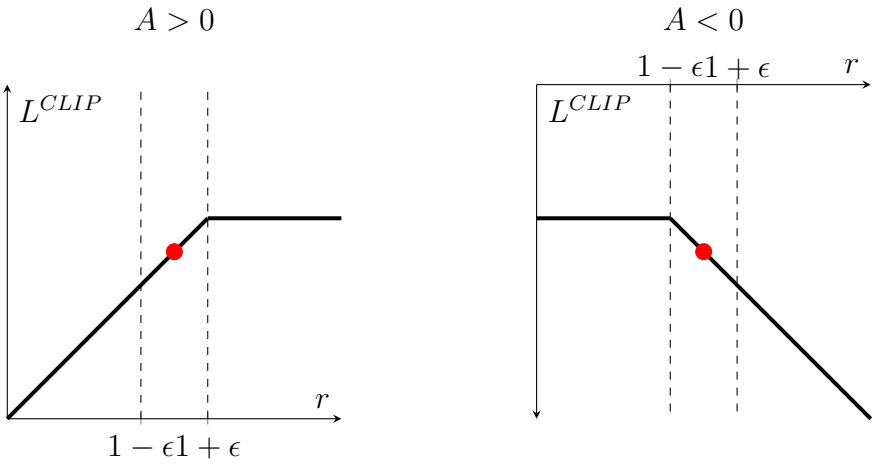


Figure 2.1: Diagrams depicting the clipped surrogate L^{CLIP} as a function of the policy probability ratio r.

Proximal Policy Optimization is an actor critic method, meaning that it is responsible for learning both the policy and the value function. For the algorithm to learn the value function, we introduce the squared-error loss L^{VF} for the value function.

$$L^{VF} = \big(V_\theta(s_t) - V_t^{\text{target}}\big)^2 \tag{2.32}$$

In addition to the policy loss and the value loss, PPO further augments the loss function by incorporating an entropy bonus, which aids the agent in exploration $S[\pi_\theta](s_t)$.

$$S[\pi_\theta](s) = -\sum_{a\in\mathcal{A}} \pi(a \mid s)\log\pi(s,a) \tag{2.33}$$

Combining the previously established components, the final PPO objective function is presented as follows.

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

Where the coefficients c_1 and c_2 are hyperparameters controlling the significance of the value loss and the entropy bonus, respectively.

In practical applications, automatic differentiation software is typically utilized to circumvent the manual computation of derivatives. In such scenarios, it is necessary to construct the clipped surrogate objective $L_t^{CLIP+VF+S}(\theta)$ and perform optimization steps on it, such as stochastic gradient descent (SGD) or ADAM [9].

So far we have only defined the objective, but we have not discussed how to estimate the advantage function. Recall how we estimated the advantage in the advantage actor critic algorithm; we argued that the true TD error is an unbiased estimator of the advantage. However, we do not know the true value function v^*, only an estimate V^{π_θ} which may be biased. Thus, if the value estimate has high bias, so will the advantage estimate.
To reduce the bias introduced by the value function, we can look at more steps, reducing the contribution of the value function in the estimate.

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+1} = -V^{\pi_\theta}(S_t) + R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^k V^{\pi_\theta}(s_{t+k}) \tag{2.34}$$

Introducing more steps does lead to a reduced bias, because the coefficient of $V^{\pi_\theta}(S_{t+k})$ becomes smaller as k increases. The reduction of bias does not come free, as with increasing the sample length, the variance increases.
To summarize, $A_t^{(1)}$ has high bias but low variance, while $A_t^{(k)}$ has low bias but high variance. How do we pick k such that we get the lowest bias and lowest variance? The generalized advantage estimator GAE(γ, λ) [15] is defined as the exponentially-weighted average of these k-step estimators.

$$\begin{aligned}
\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} :=& (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots\right) \\
=& (1-\lambda)\left(\delta_t + \lambda\left(\delta_t + \gamma\delta_{t+1}\right) + \lambda^2\left(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2}\right) + \ldots\right) \\
=& (1-\lambda)\left(\delta_t\left(1 + \lambda + \lambda^2 + \ldots\right) + \gamma\delta_{t+1}\left(\lambda + \lambda^2 + \lambda^3 + \ldots\right)\right. \\
& \left. + \gamma^2\delta_{t+2}\left(\lambda^2 + \lambda^3 + \lambda^4 + \ldots\right) + \ldots\right) \\
=& (1-\lambda)\left(\delta_t\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}\left(\frac{\lambda^2}{1-\lambda}\right) + \ldots\right) \\
=& \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}
\end{aligned} \tag{2.35}$$

As the result of the previous derivation, we can see that the generalized advantage estimator resolves in an astonishingly simple formula of the discounted sum of TD errors. To control the bias-variance trade-off, we can change the values of γ and λ.
The only problem remaining is that GAE(γ, λ) is defined as an infinite sum, which is problematic to compute on finite machines. To resolve this problem, we can sample a trajectory of length T and calculate the truncated sum for each step of the sample trajectory. [12]

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l} \tag{2.36}$$

An additional concept utilized by PPO, which has not been previously discussed, involves executing multiple actors in parallel to speed up training [12]. Integrating the previously established components, the final PPO algorithm in pseudocode is presented as follows.

Algorithm 4 Proximal Policy Optimization (PPO)

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T time steps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L with respect to θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Chapter 3

Trading fractional Brownian motion

3.1 Stochastic processes

Definition 3.1.1. Let $(\Omega, \mathcal{A}, P)$ be a probability space and (S, Σ) a measurable space. A *stochastic process* is a collection of S-valued random variables, which can be written as

$$\{X(t) : t \in \mathbb{T}\} \tag{3.1}$$

for some index set $\mathbb{T}$.

Typically, the index set $\mathbb{T}$ represents time, making $\mathbb{T}$ a subset of the real numbers. In scenarios where the index set signifies time, it becomes possible to model a variety of real-world phenomena, such as the population dynamics of a group of animals, the trajectory of a particle suspended in a medium, or the behavior of financial markets. In the ensuing sections, a concrete model for a financial market will be delineated, and the methodology to realize asymptotically optimal trading results will be discussed.

Definition 3.1.2. For every $t \in \mathbb{T}$, let $\mathcal{F}_t$ be a sub-σ-algebra of $\mathcal{A}$. Then $(\mathcal{F}_t)_{t\in\mathbb{T}}$ is called a *filtration*, if $\mathcal{F}_k \subseteq \mathcal{F}_l$ for all $k \leq l$.

Definition 3.1.3. A stochastic process X_t is *adapted* to the filtration $\mathcal{F}_t$ if the random variable $X_t : \Omega \to S$ is a $(\mathcal{F}_t, \Sigma)$ measurable function for all $t \in \mathbb{T}$.

A filtration with an adapted stochastic process effectively models the accumulation of knowledge concerning past and present events. At a given timestep $t \in \mathbb{T}$, all relevant information about the past and present is encapsulated within $\mathcal{F}_t$. The designation of $\mathcal{F}_t$ as a filtration signifies that the aggregation of our knowledge expands as time progresses. In the event that the stochastic process X_t is adapted to this filtration, then at each timestep $t \in \mathbb{T}$ the sub-σ-algebra $\mathcal{F}_t$ contains sufficient information to determine the value of all random variables X_s where $s \leq t$.

From here on out, we focus our attention on Gaussian processes, with particular emphasis on a specific process known as fractional Brownian motion. Multiple findings have established that financial prices can be modeled with stochastic processes of long memory. A collection of such results can be found at [2]. We focus on fractional Brownian motion as it exhibits the property of long memory and it has been used in the past to model financial prices [6].

Definition 3.1.4. A time continuous stochastic process X_t is Gaussian if for every finite set of indices $\{t_1, t_2, \ldots, t_k\} \subseteq \mathbb{T}$

$$X_{t_1,t_2,\ldots,t_k} = (X_{t_1}, X_{t_2}, \ldots, X_{t_k}) \tag{3.2}$$

is a multivariate Gaussian random variable.

Definition 3.1.5. A time continuous Gaussian process $B_H(t)$ on $[0, T]$ is a *fractional Brownian motion* (fBm), if the following properties are satisfied:

$$B_H(0) = 0 \tag{3.3}$$

$$\mathbb{E}[B_H(t)] = 0 \quad \forall t \in \mathbb{T} \tag{3.4}$$

$$\mathbb{E}\left[B_H(t)B_H(s)\right] = \frac{1}{2}\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right) \tag{3.5}$$

where $H \in (0, 1)$.

The parameter H is called the *Hurst parameter*, and it characterizes the type of fractional Brownian motion. In the case of $H < 1/2$, the increments of the process are negatively correlated, while in the case of $H > 1/2$, the increments are positively correlated. When $H = 1/2$, we recover the usual Brownian motion with independent increments. Figure 3.1 shows realizations of the fBm process for different Hurst parameters.

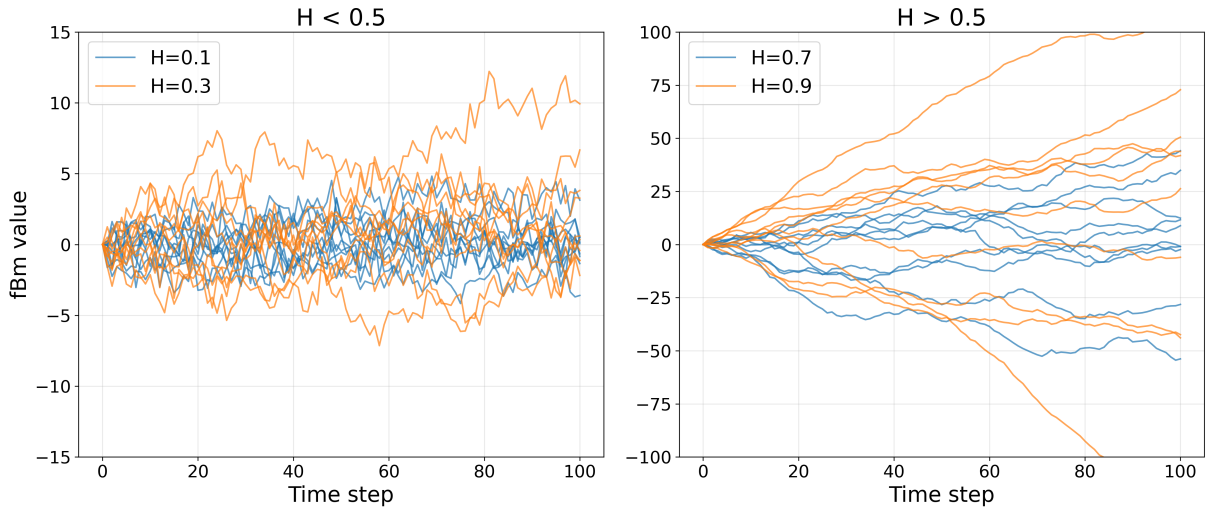


Figure 3.1: Typical fBm realizations of length 100. Hurst parameters less than 0.5 shown on the left subplot, while Hurst parameters greater than 0.5 shown on the right subplot.

3.2 Trading fractional Brownian motion

Within this section, a financial model will be presented, including a discussion on optimal trading strategies within the scope of this model. The results shown are with direct reference to [7]. Consider a modeled financial market wherein the price of a risky asset adheres to an adapted process S_t, where $t \in [0, T]$. The trader may trade at finite rates on the risky asset, though

they incur a temporary, nonlinear price impact as a consequence. Subsequently, we proceed to delineate the family of feasible strategies available to the trader.

$$\mathcal{S}(T) := \left\{ \phi : \phi \text{ is a } \mathbb{R}\text{-valued, optional process, } \int_0^T |\phi_u| \ du < \infty \text{ a.s.} \right\} \tag{3.6}$$

The trader's initial asset position is represented by $z = (z^0, z^1)$, indicating possession of z^0 units of the riskless asset and z^1 units of the risky asset. The quantity of shares in the risky asset at time $t \in [0, T]$, after following strategy ϕ, is specified as

$$X_t^1(\phi) := z^1 + \int_0^t \phi_u \ du \tag{3.7}$$

The aggregate position in the riskless asset is defined in a comparable manner, albeit incorporating the effect of price impact. The trader incurs a superlinear penalty associated with the trading speed, as determined by parameters $\alpha > 1$ and $\lambda > 0$, thereby establishing the position in the riskless asset at time $t \in [0, T]$ as

$$X_t^0(\phi) := z^0 - \int_0^t \phi_u S_u \ du - \int_0^t \lambda \left|\phi_u\right|^\alpha \ du \tag{3.8}$$

In the subsequent discussions, we assume a starting portfolio of $(z^0, z^1) = (0, 0)$, meaning that the trader starts with zero initial capital allocation in either asset.

Let $\mathcal{A}(T)$ be the family of feasible strategies starting with zero initial capital allocation, and with the final position composed exclusively of the riskless asset, and a well-defined notion of expected terminal riskless asset position. Formerly stated as

$$\mathcal{A}(T) := \{\phi \in \mathcal{S}(T) : X_T^1 = 0, \mathbb{E}[X_T^0(\phi)_-] < \infty\} \tag{3.9}$$

where $x_- = -\min\{x, 0\}$.

Building upon the preceding detailed framework, we can now define the objective of our problem: to identify the strategy $\phi \in \mathcal{A}(T)$ that realizes maximal expected profits within the riskless asset.

Proposition 3.2.1. *For each $T \in \mathbb{R}^+$, $u(T) < \infty$ and there exists $\phi^* \in \mathcal{A}(T)$ such that*

$$u(T) = \mathbb{E}[X_T^0(\phi^*)] \tag{3.10}$$

where

$$u(T) := \sup_{\phi \in \mathcal{A}(T)} \mathbb{E}[X_T^0(\phi)] \tag{3.11}$$

In essence, the aforementioned proposition asserts the existence of an optimal strategy for any time horizon T that achieves maximal returns.

Proposition 3.2.2. *There exists a market bound*

$$Q(T) = \int_0^T \mathbb{E}|S_t|^{\frac{\alpha}{\alpha-1}} \ dt \tag{3.12}$$

that dominates the returns of any strategy $\phi \in \mathcal{A}(T)$

$$\mathbb{E}|X_T^0(\phi)| \leq CQ(T) \tag{3.13}$$

where C *is some constant.*

Proof. The existence of the market bound is asserted by the proof of Proposition 3.1. in [7]. □

Theorem 3.2.3. *Let* $\lambda > 0, \alpha > 1$*, and* $H \in (1,2) \setminus \{0.5\}$*. Then for* $S_t := B_t^H$*,*

(i) Maximal expected profits satisfy

$$\limsup_{T\to\infty} \frac{u(T)}{T^{H(1+1/(\alpha-1))+1}} < \infty \tag{3.14}$$

(ii) For each $0 < \kappa < 1/(\alpha-1)$*, the strategies*

$$\phi_t(T,\kappa) := \begin{cases} \operatorname{sgn}\left(S_t(H-1/2)\right)|S_t|^\kappa, & t \in [0, T/2), \\ -\frac{1}{T/2}\int_0^{T/2} \phi_s \, ds, & t \in [T/2, T] \end{cases} \tag{3.15}$$

satisfy

$$\lim_{T\to\infty} \frac{EX_T^0(\phi(T,\kappa))}{T^{H(1+\kappa)+1}} > 0 \tag{3.16}$$

The first part of the theorem states that the ratio

$$\frac{u(T)}{T^{H(1+1/(\alpha-1))+1}} \tag{3.17}$$

does not grow without bound. Meaning that for sufficiently large time horizons T, there exists C such that

$$u(T) \leq C \cdot T^{H(1+1/(\alpha-1))+1} \tag{3.18}$$

The second part of the theorem describes a family of strategies that exhibit at least as steep a slope as $T^{H(1+\kappa)+1}$. In the limiting case, where $\kappa \to 1/(\alpha-1)$, the two parts of the theorem combined establish that the described strategy achieves asymptotically optimal returns. A depiction of the asymptotic optimality of the described strategy is presented in Figure 3.2.

The proposed strategy is predicated on the Hurst parameter of the fBm process. It executes trades counter to the price direction in the anti-persistent scenario, while aligning with the price direction under persistent conditions. The second half of the trading period is exclusively dedicated to linear liquidation, minimizing the impact of trading speed while liquidating its entire position in the risky asset. Figure 3.4 aims to provide a depiction of the distribution of actions for each time step $t \in [0, T]$, while Figure 3.3 portrays representative trajectories of the optimal strategy for four different Hurst parameters.

A key metric economists measure when assessing the risk associated with the return of investing strategies is the *Sharpe ratio*. In essence, the Sharpe ratio aims to quantify the reliability of excess returns. The Sharpe ratio is defined as the expected returns in the riskless asset divided by the standard deviation of the returns in the riskless asset (3.19). Figure 3.5 depicts the empirically measured Sharpe ratio of the optimal strategy for different Hurst parameters.

$$\frac{\mathbb{E}[X_T^0 - X_0^0]}{\sqrt{\operatorname{var}(X_T^0 - X_0^0)}} \tag{3.19}$$

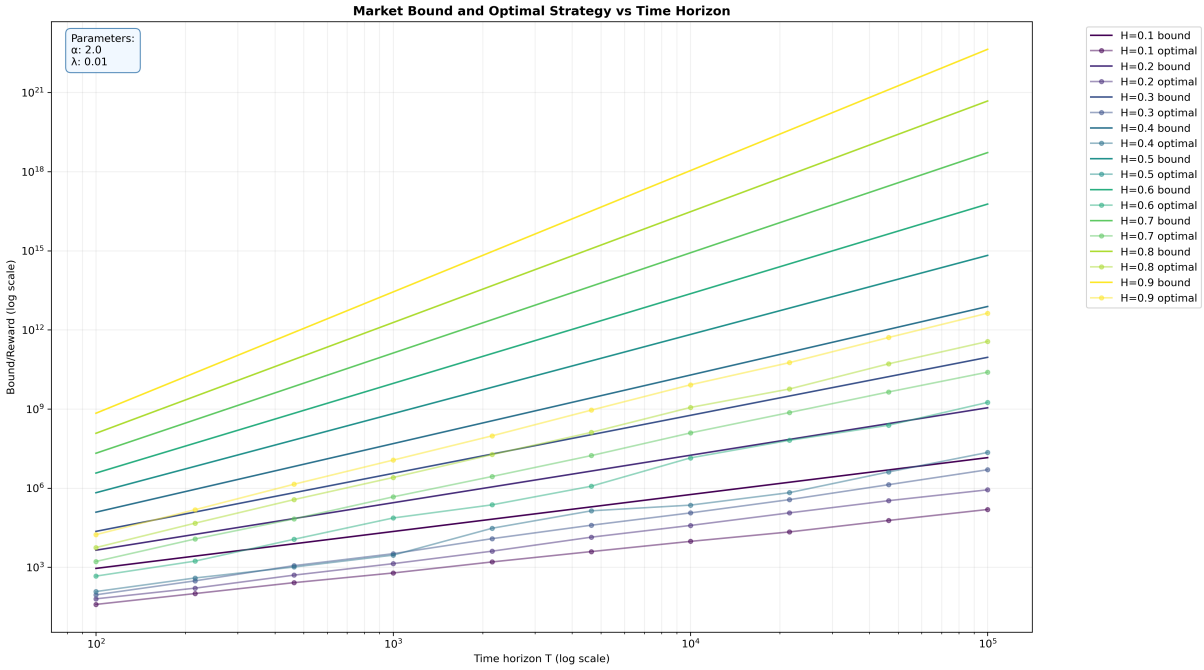


Figure 3.2: Expected profits (vertical axis in logarithmic scale) against time horizon (horizontal axis in logarithmic scale) with the market bound. For each Hurst parameter and each time horizon the expected returns of the optimal strategy are empirically calculated from 500 samples.

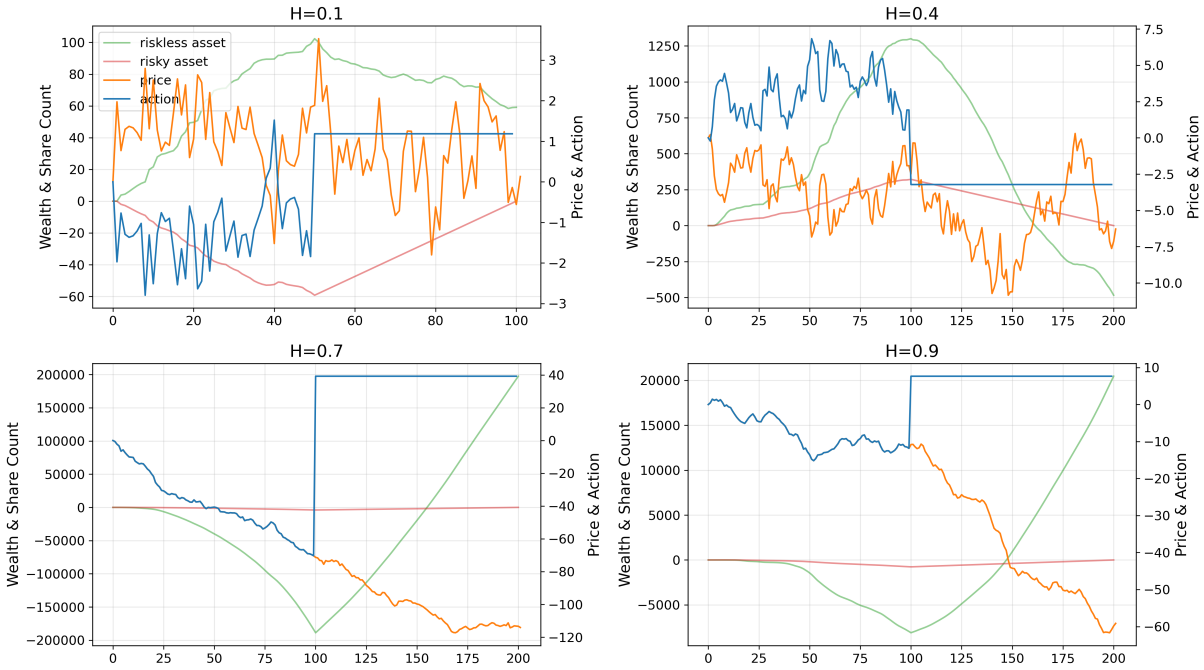


Figure 3.3: Typical realizations for the optimal strategy for different Hurst parameters.

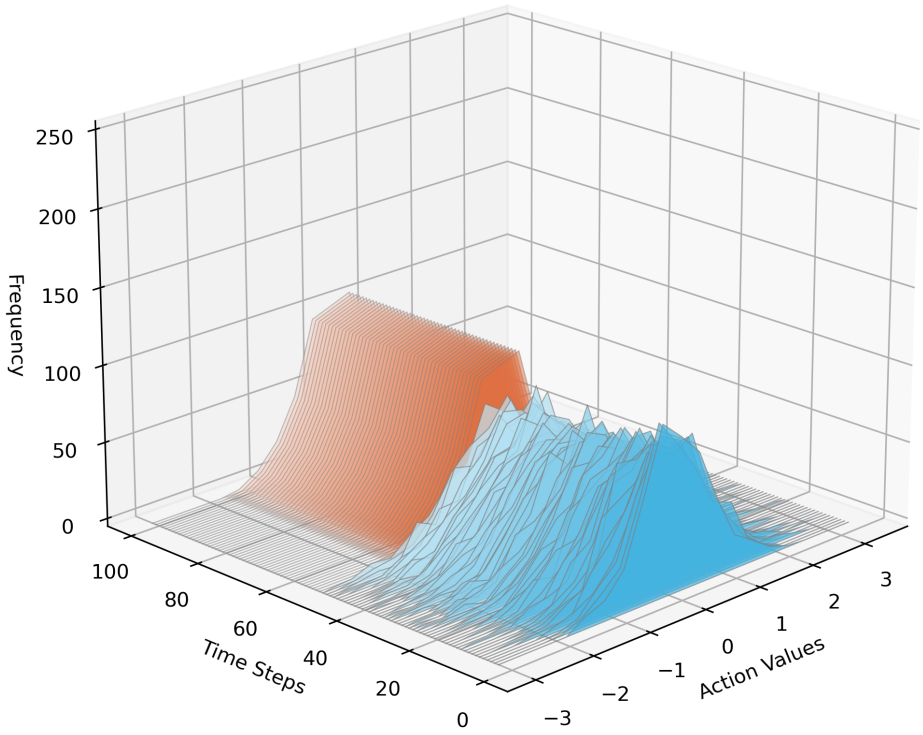


Figure 3.4: Distributions of actions for 500 simulated trading sequences for the optimal strategy for every time step. The active trading half of the action distributions in blue, with the liquidation half of the trading period in orange.

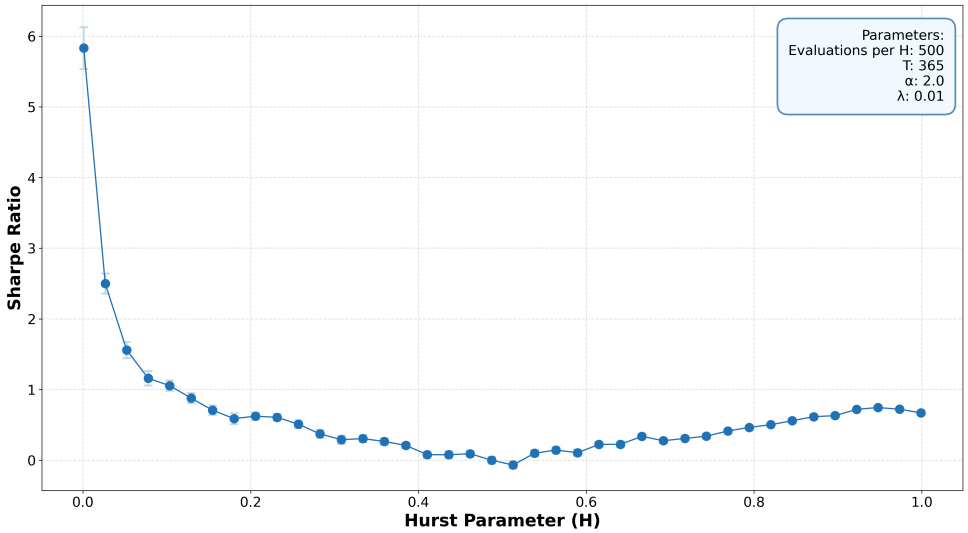


Figure 3.5: Sharpe ratio against Hurst parameter for optimal strategy, each data point calculated from 500 simulations.

Chapter 4

Experiments

4.1 Artificial trading on autoregressive prices

This chapter details the experiments conducted to train an artificial agent, using reinforcement learning, to discover optimal trading strategies within the framework defined in Section 3.2.

The initial step in addressing a problem with reinforcement learning involves defining an environment that encapsulates the problem's key characteristics. Our research involved experimenting with various investment settings, leading to the development of a base investment environment from which different specific settings are derived. The base investment environment is characterized by the price process generator and the length of the trading period, T. Each specific investment setting is defined by these components, alongside a unique utility function and the implementation of a single trading step.

An implementation of the Proximal Policy Optimization algorithm, based on the CleanRL single-file implementation [8], was used as the training algorithm. Across all experiments, both the actor and the critic were implemented using a multilayer perceptron (MLP). A list of all the hyperparameters used can be found in Table 4.1.

Initially, we sought to replicate the results presented by [5], which outline an investment problem based on autoregressive prices. However, the exponential utility function defined as:

$$u(x) = -e^{-x} \tag{4.1}$$

was deemed computationally impractical. To address this, we explored substituting a linear utility function, with the aim of achieving satisfactory performance in the original setting after training in the modified setting. However, this approach proved unsuccessful. The original exponential utility heavily penalized even minor losses, overshadowing positive returns. Figure 4.1 illustrates the evaluation results during training, comparing the linear utility (blue) and the certainty coefficient (orange). The certainty coefficient for n realizations is calculated as the inverse utility of the average utility:

$$u^{-1}\left(\frac{\sum u(x_i)}{n}\right) \tag{4.2}$$

Following these initial experiments, we shifted our focus to the investment setting detailed in Section 3.2, which employs the identity function as the utility function. This setting presents

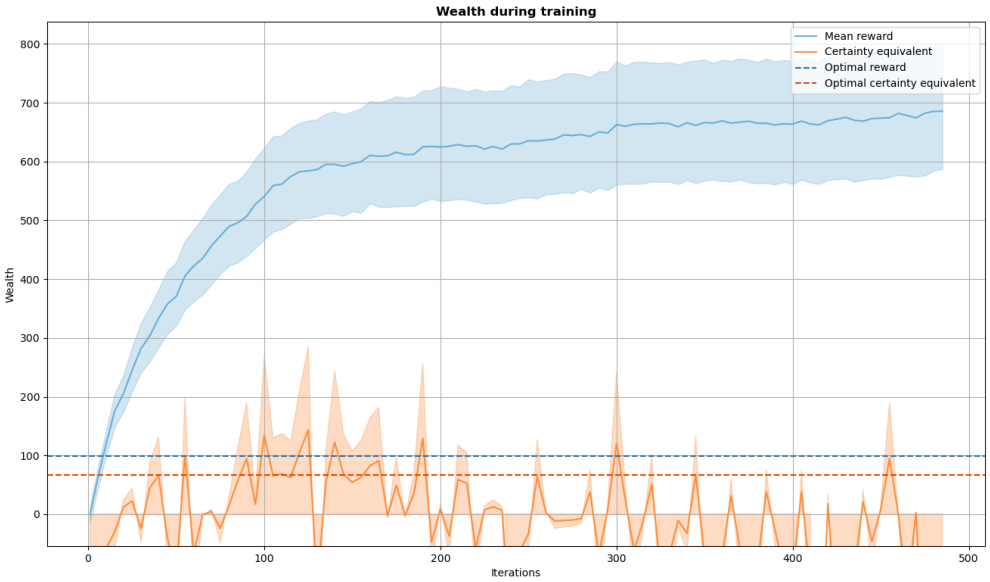


Figure 4.1: Evaluation during training for autoregressive process against the optimal strategy. At each iteration, the current model is evaluated on 5000 sample episodes. The average terminal cash holdings of the current model are shown in solid blue, while the shaded area depicts the interquartile range of outcomes. The certainty coefficient of the current model is shown in solid orange, while the 95% confidence interval of the certainty coefficient is shown in shaded orange.

challenges due to the superlinear friction imposed on the trading speed, coupled with the complexity of the fBm process compared to the autoregressive process.

4.2 Artificial trading on fractional Brownian motion

Similar to the autoregressive environment, we implemented the trading step for this investment problem, incorporating the friction on trading speed and utilizing a prepackaged Python implementation for simulating fBm realizations. It is important to note that Theorem 3.2.3 applies to continuous-time trading, which is infeasible on computers. Consequently, we discretized the investment problem by restricting trading on the fBm process to positive integers. The definitions of the risky and riskless assets were adapted to accommodate this discrete-time trading framework. The quantity of shares in the risky asset is defined as:

$$H_t = \sum_{k=1}^{t} h_k \tag{4.3}$$

The aggregate position in the riskless asset is defined as:

$$V_t = -\sum_{k=1}^{t} S_k h_k - \sum_{k=1}^{t} \lambda |h_k|^\alpha \tag{4.4}$$

Fractional Brownian motion has been extensively studied as a way to model asset prices since its first introduction by Mandelbrot [11]. However, whether it is truly an accurate model has been deeply debated, as studies contending whether it displays long memory have been inconclusive or rejected. Among the many studies, which aim to model asset prices with fractional Brownian motion, Gatheral et al. [6] estimated the smoothness of the log-volatility of assets represented in the Oxford-Man dataset. Their findings led them to model the log-volatility with a fractional Brownian motion with Hurst parameter $H < 0.5$. Furthermore, estimating the Hurst parameter of the selected assets using their model shows that it lies between 0.08 and 0.2. Therefore, we concentrated our efforts on training an agent on fBm with Hurst parameter of this range. To examine the asymptotic properties of the reinforcement learning model, we trained five distinct models across increasing time horizons, ranging from 128 to 2048. Figure 4.2 illustrates the performance of the trained model against the analytically optimal model, while Figure 4.3 presents sample trading realizations executed by the trained model for fixed time horizon $T = 128$, and Figure 4.4 shows the plot of evaluations for each iteration during training for time horizon $T = 128$.

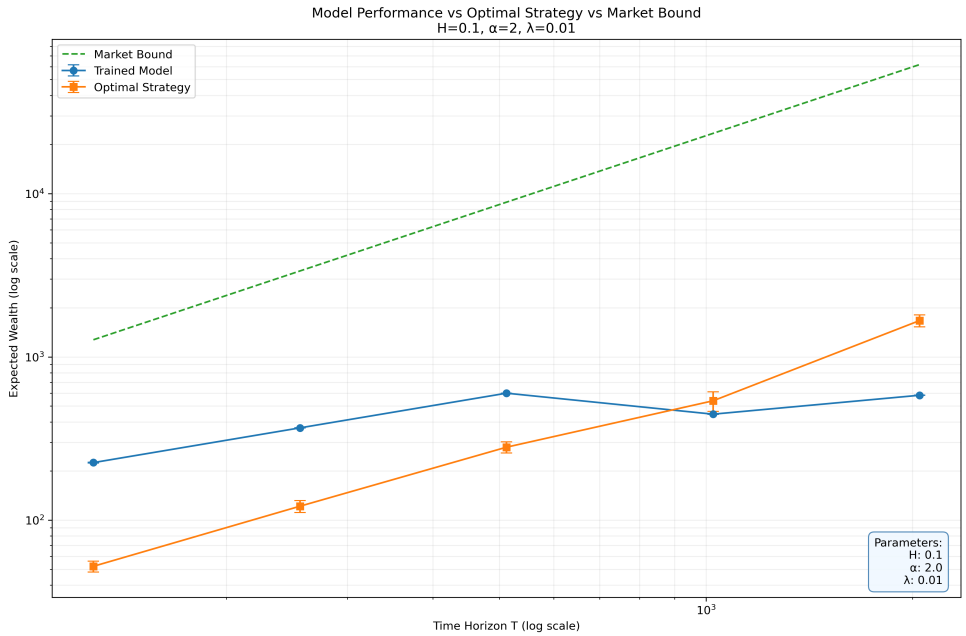


Figure 4.2: Asymptotic model evaluation for fixed Hurst parameter $H = 0.1$, trading period against model performance and market bound. Trained RL model in blue against analytical strategy of Theorem 3.2.3 in orange against theoretical market bound in dashed green.

The results indicate that the trained model outperforms the analytical strategy for time horizons up to 512. However, the performance of the reinforcement learning framework degrades as the time horizon increases. This issue is attributed to the credit assignment problem, a common challenge in reinforcement learning where longer episodes make it difficult to accurately attribute outcomes to specific actions.

As previously discussed, the Hurst parameter for indices within the Oxford-Man Institute of Quantitative Finance Realized Library ordinarily spans a range from 0.08 to 0.2. A more comprehensive exposition of this data is available in Table B.2 of [6]. Consequently, we trained a model using a Hurst parameter drawn from a uniform distribution on the interval (0.05, 0.15). Figure 4.5 presents the evaluation of this model.

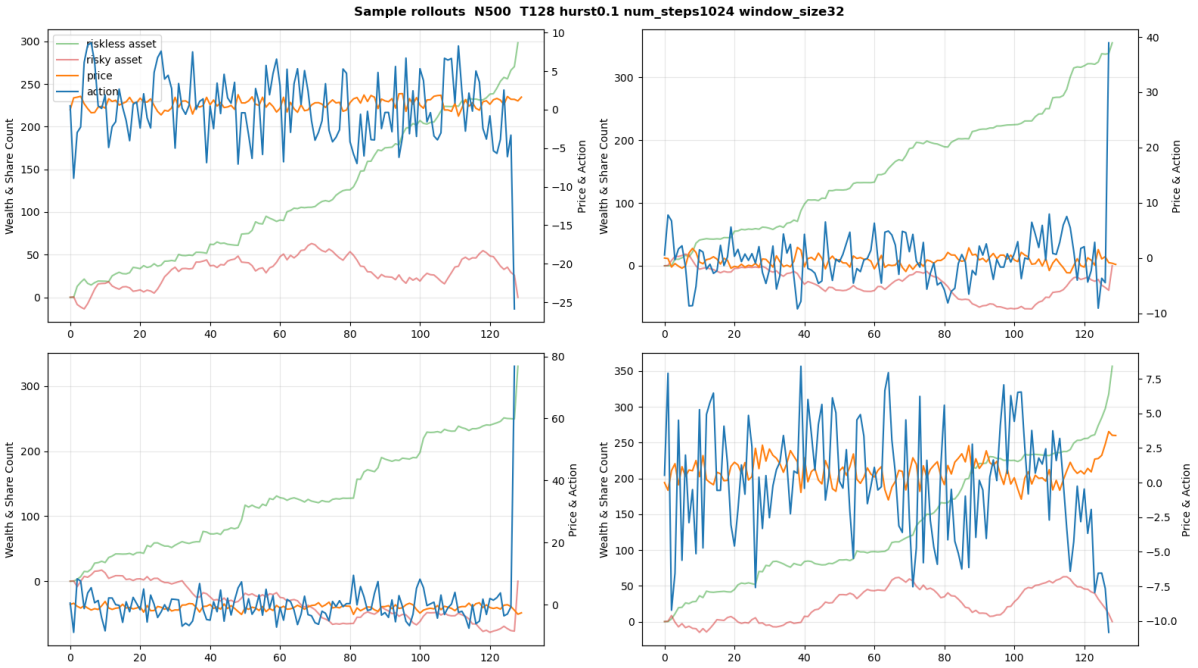


Figure 4.3: Four typical episodes for the trained agent, with parameters $H = 0.1$, $T = 128$. Actions are shown in blue, prices are shown in orange, riskless asset holdings are shown in purple, risky asset holdings are shown in red. The wealth (green) is computed as the sum of the riskless asset position and the value of the held risky assets.

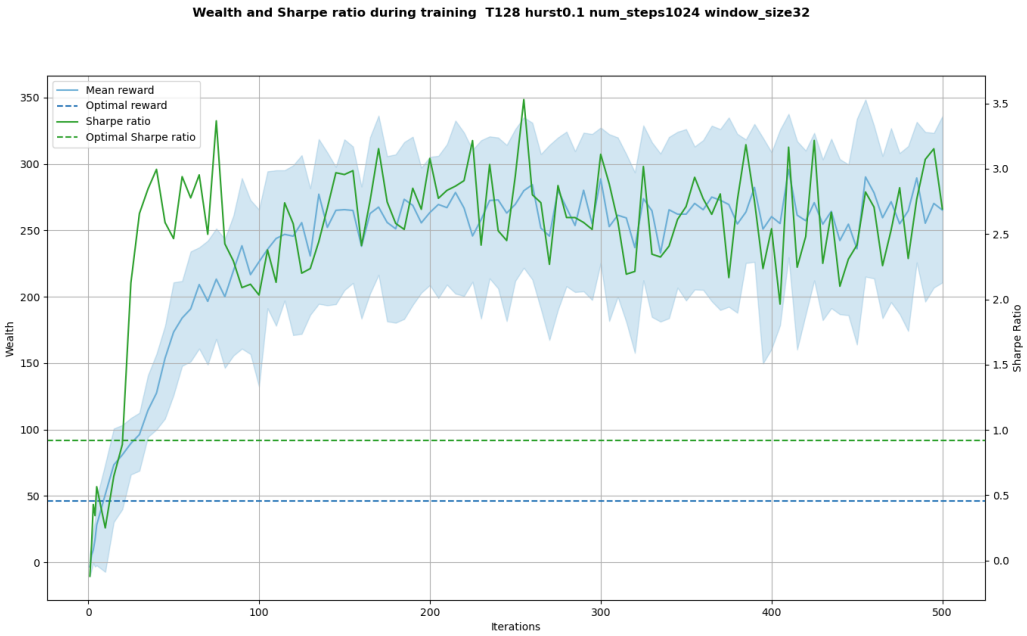


Figure 4.4: Evaluation during training for fixed Hurst parameter $H = 0.1$, iterations against wealth and Sharpe ratio. At each iteration, the current model is evaluated on 500 sample episodes. The average terminal cash holdings of the current model are shown in solid blue, while the shaded area depicts the interquartile range of outcomes. Calculated returns for the analytical strategy of Theorem 3.2.3 are shown in dashed blue, while calculated Sharpe ratio is shown in dashed green.

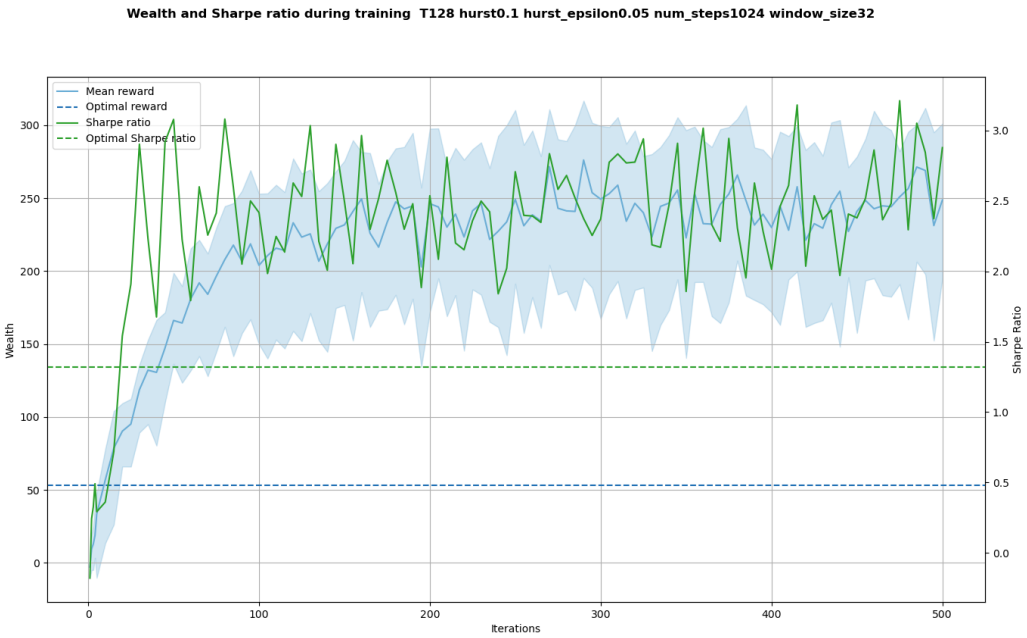


Figure 4.5: Evaluation during training for random Hurst parameter $H \sim U(0.05,\ 0.15)$, iterations against wealth and Sharpe ratio. At each iteration, the current model is evaluated on 500 sample episodes. The average terminal cash holdings of the current model are shown in solid blue, while the shaded area depicts the interquartile range of outcomes. Calculated returns for the analytical strategy of Theorem 3.2.3 are shown in dashed blue, while calculated Sharpe ratio is shown in dashed green.

Hyperparameter	Value
seed	1
torch_deterministic	True
cuda	True
save_model	False
eval_train	True
num_eval_train	100
num_envs_eval	100
action_train	False
printing	True
env_id	FBMEnv
env_kwargs.T	128
env_kwargs.hurst	0.1
env_kwargs.hurst_epsilon	0.05
env_kwargs.random_hurst	True
env_kwargs.friction_parameter	2
env_kwargs.illiquidity	0.01
env_kwargs.penality_version	0
env_kwargs.reward_type	delta bankroll
env_kwargs.window_size	32
learning_rate	0.0003
num_envs	32
num_steps	8T
anneal_lr	True
gamma	1.0
gae_lambda	0.95
num_minibatches	32
update_epochs	10
norm_adv	True
clip_coef	0.2
clip_vloss	True
ent_coef	0.0
vf_coef	0.5
max_grad_norm	0.5
num_iterations	500

Table 4.1: Hyperparameters used in training for the random uniform Hurst experiment.

Conclusion

In this thesis, we have explored the mathematical framework of reinforcement learning, with the primary objective of applying its methods to find optimal trading strategies in a novel financial market model. By employing the proximal policy optimization algorithm, we were able to demonstrate that in the market model described in Section 3.2, the trained artificial agent was capable of performing on par with the analytically optimal strategy for small enough trading periods and anti-persistent prices.

In conclusion, the results, shown in Figure 4.4 and Figure 4.5, demonstrate that our trained model outperforms the strategy outlined in Theorem 3.2.3 for sufficiently small time horizons. This leads to the conclusion that reinforcement learning can be effectively used to train an agent that surpasses a known analytical strategy for trading periods of limited duration. On the basis of the claims stated in [6], one could find practical applications of the proposed artificial agent by trading on index funds modeling volatility, such as the VIX [1], and expect higher expected returns with smaller risk for trading periods of up to 512 days.

While this work addresses several important aspects of the problem, some questions remain open. In particular, trading on persistent fBm prices is not discussed as the liquidation process complicates the strategy that the agent needs to learn. Future studies could build on these results by extending to persistent fBm prices by limiting the agent to only trade in the first half of the trading period and systematically liquidating the amounted assets linearly, as it can be shown that in this case linear liquidation is optimal. Further extensions are also within reach to adapt different market models to the reinforcement learning framework, even in the case where analytical strategies are not known.

In summary, this thesis contributes to the field of reinforcement learning driven quantitative finance by demonstrating that reinforcement learning is capable of learning optimal trading strategies, and it lays the groundwork for further exploration of financial market models modeled and solved with reinforcement learning.

Bibliography

[1] CBOE Volatility Index (VIX). https://www.cboe.com/tradable_products/vix/. Accessed: 2025-05-23.

[2] Long memory. https://www.long-memory.com. Accessed: 2025-05-23.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[4] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[5] Sándor Deák and Miklós Rásonyi. An explicit solution for optimal investment problems with autoregressive prices and exponential utility. *arXiv preprint arXiv:1501.01506*, 2015.

[6] Jim Gatheral, Thibault Jaisson, and Mathieu Rosenbaum. Volatility is rough. *Quantitative finance*, 18(6):933–949, 2018.

[7] Paolo Guasoni, Zsolt Nika, and Miklós Rásonyi. Trading fractional Brownian motion. *SIAM journal on financial mathematics*, 10(3):769–789, 2019.

[8] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[11] Benoit B Mandelbrot. When can price be arbitraged efficiently? A limit to the validity of the random walk and martingale models. *The Review of Economics and Statistics*, pages 225–236, 1971.

[12] 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. PmLR, 2016.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[14] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

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

[17] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[18] David Silver. Lectures on Reinforcement Learning. URL: `https://www.davidsilver.uk/teaching/`, 2015.

[19] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[20] Thomas Simonini and Omar Sanseviero. The Hugging Face Deep Reinforcement Learning Class. `https://github.com/huggingface/deep-rl-class`, 2023.

[21] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[22] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[23] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[24] Lilian Weng. Policy Gradient Algorithms. `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`, 2018.

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

I, the undersigned Leonardo Toffalini, hereby declare that in the course of preparing my thesis, I used the artificial intelligence-based tools listed below to perform the specified tasks.

Task	**Tool used**	**Place of usage**	**Notes**
Research	Perplexity	Entire thesis	
Review	Perplexity	Entire thesis	
Phrasing	Writefull	Entire thesis	
Programming	Cursor	Experiments	

Beyond those listed above, I did not use any other AI-based tools.